УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

ЗБОРНИК РАДОВА
ФАКУЛТЕТА ТЕХНИЧКИХ НАУКА

Едиција: Техничке науке – зборници
Година: XL
Број: 12/2025

Нови Сад

Едиција: „Техничке науке – Зборници“
Година: XL Свеска: 12
Издавач: Факултет техничких наука Нови Сад

Главни и одговорни уредник Едиције: проф. др Борис Думнић, декан Факултета техничких наука у Новом Саду

Штампање одобрио: Савет за библиотечку и издавачку делатност ФТН, председник, проф. др Селена Самарџић Цвијановић

Штампа: ФТН – Графички центар ГРИД, Трг Доситеја Обрадовића 6, Нови Сад

CIP-Каталогизација у публикацији
Библиотека Матице српске, Нови Сад

378.9(497.113)(082)
62

ЗБОРНИК радова Факултета техничких наука / главни и одговорни уредник
Борис Думнић. – Год. 7, бр. 9 (1974)-1990/1991, бр.21/22 ; Год. 23, бр 1 (2008)-. – Нови Сад : Факултет техничких наука, 1974-1991; 2008-. – илустр. ; 30 цм. – (Едиција: Техничке науке – зборници)

Месечно

ISSN 0350-428X

COBISS.SR-ID 58627591

ПРЕДГОВОР

Поштовани читаоци,

Пред вама је десета овогодишња свеска часописа „Зборник радова Факултета техничких наука“.

Часопис је покренут давне 1960. године, одмах по оснивању Машинског факултета у Новом Саду, као „Зборник радова Машинског факултета“, а први број је одштампан 1965. године. Након осам публикованих бројева у шест година, пратећи прерастање Машинског факултета у Факултет техничких наука, часопис мења назив у „Зборник радова Факултета техничких наука“ и 1974. године излази као број 9 (VII година). У том периоду у часопису се објављују научни и стручни радови, резултати истраживања професора, сарадника и студената ФТН-а, али и аутора ван ФТН-а, тако да часопис постаје значајно место презентације најновијих научних резултата и достигнућа. Од броја 17 (1986. год.), часопис почиње да излази искључиво на енглеском језику и добија поднаслов *Publications of the School of Engineering*.

Наставно-научно веће ФТН-а је одлучило да од новембра 2008. год. у облику пилот пројекта, а од фебруара 2009. год. као сталну активност, уведе презентацију најважнијих резултата свих мастер радова студената ФТН-а у облику кратког рада у „Зборнику радова Факултета техничких наука“.

Поред студената мастер студија, часопис је отворен и за студенте докторских студија, као и за прилоге аутора са ФТН или ван ФТН-а.

Зборник излази у два облика – електронском на веб-страници Факултета техничких наука (www.ftn.uns.ac.rs) и штампаном, који је пред вама. Обе верзије публикују се сваки месец, у оквиру промоције дипломираних мастера.

Известан број кандидата објавили су радове на некој од домаћих научних конференција или у неком од часописа. Њихови радови нису штампани у Зборнику радова ФТН-а.

У свесци са редним бројем 12 објављени су радови из области машинског инжењерства и електротехничког и рачунарског инжењерства.

Континуираним радом и унапређењем квалитета часописа, план је да часопис постане препознатљив међу ауторима, чиме ће значајно допринети да се оствари мото Факултета техничких наука:

„Високо место у друштву најбољих“

Уредништво

Садржај

Машинско инжењерство

Електротехничко и рачунарско инжењерство

Зборник радова Факултета техничких наука, Нови Сад

UDK: 621
DOI: https://doi.org/10.24867/33AM01Cvetkovic

БЕЗБЕДНОСТ И ЗАШТИТА НА РАДУ СА МАШИНАМА ЗА ВЕРТИКАЛНИ ТРАНСПОРТ ЛИЦА И ТЕРЕТА

SAFETY AND PROTECTION AT WORK WITH MACHINES FOR THE VERTICAL TRANSPORT OF PERSONS AND GOODS

Зоран Цветковић, *Факултет техничких наука, Нови Сад*

Област – МАШИНСКО ИНЖЕЊЕРСТВО

Кратак садржај – *У раду је дат историјски преглед лифтова као машина за вертикални транспорт лица и терета. Дат је опис лифтова на електрични погон, њихове предности и недостаци у односу на хидрауличне. Дат је преглед основних елемената лифта са сврхом и начином рада, као и безбедносне компоненте електричних лифтова. Наведене су најчешће и најзначајније инцидентне ситуације које се могу појавити код експлоатације лифтова са поступком за њихову превенцију и отклањање. На крају, приказан је детаљан поступак прегледа и провере електричног путничког лифта са издатим Извештајем.*

Кључне речи: *Електрични лифт, безбедносне компоненте, инцидентне ситуације, преглед и провера лифта*

Abstract – *This thesis describes a historical overview of elevators as machines for vertical transport of people and cargo. There is a description of electrical elevators, their advantages and disadvantages compared to hydraulic ones. An overview of the basic elements of the elevator with its purpose and mode of operation, as well as the safety components of electrical elevators, is given. The most common and most significant incident situations that can occur during the operation of elevators are listed with the procedure for their prevention and elimination. Finally, a detailed procedure of inspection and verification of the electrical passenger elevator with the issued Report is shown.*

Keywords: *Electrical elevator, safety components, incident situations, inspection and verification of the elevator*

1. УВОД

Лифтови су вероватно један од најважнијих проналазака у историји човечанства, а њихова вертикална кретања на електрични погон су дуго била саставни део у стварању грађевинских прича доступних и практичних за изградњу. Лифт се као изум показао као један од најкориснијих изума са аспекта транспорта терета на велике висине, јер скраћује време транспорта и нема алтернативу у том домену. Функција лифтова је транспорт терета и људи. Уобичајено је да се лифтови или лифтовске платформе крећу уз помоћ електромотора или помоћу хидрауличне пумпе.

Лифтови представљају врсту машина за подизање и спуштање људи и/или терета у кабини, периодичним дејством, тако што се кабина креће дуж крутих, праволинијских вођица паралелно уграђених, са углом нагиба према вертикали до 15º. Чињеница је да је лифт једно од најважнијих и најмасовнијих средстава путничког транспорта у великим градовима. Његова примена непрекидно расте, што је несумњиво у функцији са објективном тенденцијом ка повећању спратности у грађевини.

Сви лифтови се могу поделити [1]:

1) према намени (функцији),

2) према врсти погонског механизма за дизање,

3) према конструкцији преносног механизма за кретање кабине,

4) према начину преноса кабине,

5) према брзини кретања кабине,

6) према шеми намотавања вучних ужади,

7) према постојању машинске просторије,

8) према положају машинске просторије,

9) према начину погоњења погонског добоша,

10) према тачности заустављања кабине лифта.

2. ЛИФТОВИ НА ЕЛЕКТРИЧНИ / ХИДРАУЛИЧНИ ПОГОН

2.1. Хидраулични лифтови

Хидраулични лифтови користе уље под притиском и хидроцилинре уз чију помоћ долази до кретања лифта. Постоје хидраулични лифтови који поред хидрауличког погона користе и додатне ужетњаче и челична ужад за покретање кабине. Најважнија специфичност код хидрауличких лифтова је то што се обично не примењује противтег, већ се погон врши само у једном правцу (на горе) док се у другом правцу користи сила земљине теже уз контролу вентила за кретање којим се регулише брзина кретања лифта. Ако дође до критичне ситуације, на сцену ступају сигурносни вентили који ће осигурати безбедно коришћење лифта.

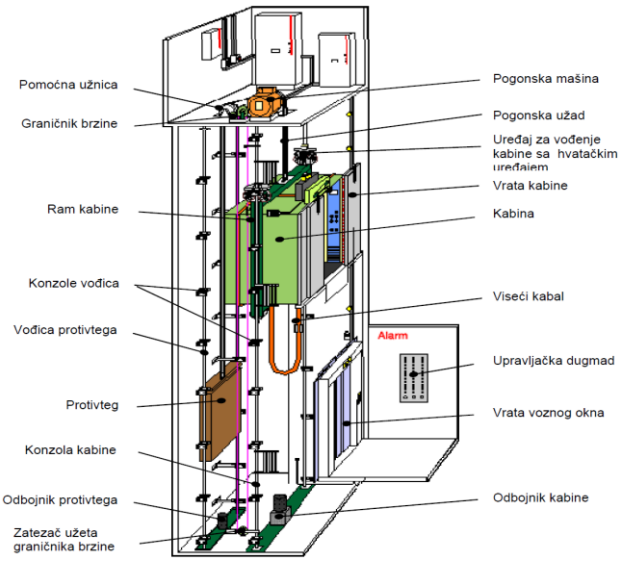
НАПОМЕНА:
Овај рад је проистекао из матер рада чији ментор је био др Радомир Ђокић, ванр. проф.

Лифтови са хидрауличним погоном примењују се за вертикални транспорт лица и терета до максималне висине од 30 m, док се у пракси користе дизања од 20-24 m.

Ови лифтови обезбеђују миран рад, без трзаја и буке као и велику тачност пристајања од ±2 mm, која се код лифтова са вучом преко ужади не може обезбедити. Не захтевају додатни простор на врху конструкције и једноставније се спуштају кабине током квара, а такође им је неосетан полазак као и кочење лифта. Мана им је што се могу користити само за објекте мање спратности, мала брзина (око 1 m/s), а такође имају проблеме цурења уља. Носивост лифтова на хидраулични погон може бити веома велика и до 7 t (теретни лифтови на хидраулични погон за дизање моторних возила).

2.2. Електрични лифтови

Електрични лифтови су најзаступљенији, а брзина кретања им је од 0,2 m/s до 17 m/s. Како им је функција превоз људи, могу да се користе и за превоз терета где им је носивост изнад 10.000 kg (10 t), наравно, при мањим брзинама.



Слика 1. *Лифт са горњим положајем машинске просторије*

Код електричних лифтова, кабина је обешена носећим ужетом (челичном сајлом) око ужетњаче која је у директној вези са погонским делом. Тежина кабине је балансирана контратегом, чија је маса једнака маси кабине и 50% од максималне дозвољене масе. Сврха контратега је да одржава готово константном укупну потенцијалну енергију система, да обезбеди лакши рад моторног дела и елиминише трзаје. Обртањем вратила мотора, покреће се погонска ужетњача, а трењем ужади о њене жљебове остварује се кретање кабине и противтега по њиховим вођицама. Смерови кретања кабине и противтега су супротни. Одговарајућим управљањем рада погонске машине и осталих делова лифтовског постројења реализује се кретање кабине између свих станица које лифт опслужује, отварање и затварање врата лифта, а тиме и транспорт путника и терета.

2.3. Конструкција лифта – уређаји, распоред и деловање елемената лифта

Основну конструкцију лифта представља механизам за дизање на електрични или хидраулични погон, са одговарајућим витлом и системом намотавања ужади за вучу кабине.

Конструктивни делови:

Вођице, возно окно, јама возног окна, машинска просторија, противтег, хватачки уређај, граничник брзине, погонски механизам, одбојници кабине и противтега, врата лифта, кабина лифта [2].

2.4. Предности и недостаци електричних лифтова у односу на хидрауличне лифтове

Електрични лифтови доносе низ предности у поређењу са другим лифтовима, али свакако да постоје и одређени недостаци.

Предности:

• Брзина и носивост: обично пружају бржи и ефикаснији транспорт у поређењу са хидрауличним или другим врстама лифтова, што може бити корисно у зградама са великим бројем спратова или високим фреквенцијама коришћења. Као што је већ наведено у тачки 2.2, брзина кретања им је од 0,2 m/s до 17 m/s. Како им је функција превоз људи, могу се користити и за превоз терета, где носивост прелази 10.000 kg (10 t), наравно, при мањим брзинама. Хидраулични лифтови су такође подесни за превоз терета и транспорт људи у нижим стамбеним зградама, а брзина кретања им је од 0,25 m/s до максимално 1 m/s. Носивост теретних хидрауличних лифтова се креће од неколико стотина килограма до више десетина тона.

• Поузданост: електрични лифтови имају високу поузданост и мање су подложни кваровима и то их чини пожељним избором за комерцијалне и стамбене објекте.

• Ниска потрошња енергије: модерни електрични лифтови су дизајнирани са ефикасним моторима и системима управљања енергијом, што доприноси смањењу потрошње енергије.

• Како је већ наведено у тачки 2.2, код електричних лифтова кабина је обешена носећим ужетом око погонске ужетњаче која је у директној вези са погонском машином. Тежина кабине је балансирана контратегом, чија је маса једнака маси кабине и 50% од максималне дозвољене масе. Сврха контратега је да одржава готово константном укупну потенцијалну енергију система, да обезбеди лакши рад моторног дела и елиминише трзаје. Хидраулични лифтови, такође, имају кабину, возно окно и машинску просторију. Они немају противтег. Крећу се по вођицама. Погон се реализује хидрауличним флуидом под притиском, кога покреће електромотор и пумпа, потопљена у резервоар са уљем. Поред наведеног, у опрему хидрауличног лифта спадају и командни вентили за подизање и спуштање кабине, неповратни вентил, вентил сигурности, челичне цеви и гумена црева.

Слика 2. *Противтег лифта произвођача „OTIS“, тип „GeN2 Premier“*

Недостаци:

• Потреба за електричном енергијом: ови лифтови захтевају стално снабдевање електричном енергијом како би радили, па квар у напајању може довести до застоја или губитка функционалности.

• Трошкови инсталације: инсталација може бити скупља у поређењу са хидрауличним или другим врстама лифтова, а посебно у случају комплексних или високих инсталација.

• Потребно одржавање: иако су електрични лифтови обично поуздани, они захтевају редовно одржавање како би се осигурало да раде исправно и безбедно, што може додатно повећати трошкове током времена.

• Потенцијал за кварове електронике: електрични лифтови садрже комплексне електронске компоненте које су подложне кваровима или оштећењима услед разних електричних проблема.

3. ИНЦИДЕНТНЕ СИТУАЦИЈЕ КОЈЕ СЕ МОГУ ПОЈАВИТИ КОД ЛИФТОВА

Лифтови су статистички најбезбедније превозно средство. Упркос страху од заглављивања или слободног пада у лифту, вожња лифтом је заправо безбеднија од вожње аутомобилом. Просечно 26 људи погине сваке године у несрећама са лифтовима (и то углавном техничари који раде на лифту, а не путници), док 26 људи погине сваког сата у саобраћајним несрећама. Лифтови се користе у хотелима, тржним центрима, стамбеним зградама, канцеларијама, болницама, аеродромима и многим другим локацијама. Власници зграда/објеката су дужни да своје лифтове одржавају на безбедан начин и постављају знак упозорења када се не смеју користити. Ови власници се могу сматрати немарним када не испуне ове обавезе према станарима, гостима и купцима [3].

Неке од главних ситуација када се крши ова дужност:

• Прикљештења

Једна озбиљна повреда код лифта је заглављивање у покретним деловима или између њих. Ово може бити између врата лифта. Ако су врата лифта неисправна, могу се отворити или затворити изненада или пребрзо и заробити људе који покушавају да уђу или изађу из лифта између врата. Врата такође могу да се не отворе до краја или лифт може да почне да се креће док је неко ухваћен вратима - и једно и друго може бити изузетно опасно. Мала деца су у повећаном ризику да им се руке, ципеле или одећа захвате у лифтовима, али и одећа одраслих такође може бити заробљена. Ако се неки одевни предмет ипак ухвати, може се брзо увући невероватном снагом и сваке године постоје бројни извештаји о деци која су изгубила прсте на рукама и ногама због ових незгода.

• Брзина

Ако се лифт креће великом брзином, односно прекорачење брзине као последица слободног пада, то је често узроковано противтегом или контролним системима који не функционишу. Лифт се може трзати горе-доле и бацити појединце у зидове лифта и на под.

• Пад у окно лифта

Једна од најсмртоноснијих врста незгода у лифту је она у којој кабина лифта или појединац у њој падне у окно. Може се појавити у бројним ситуацијама, као што су:

- Када се лифт заустави,
- Када се врата лифта отворе између спратова и особа у лифту изађе из њега и падне у окно,
- Када неко покуша да незаконито отвори врата возног окна,
- Када необучено особље покуша да спасе особу заробљену у лифту.

• Општи кварови

Неисправан лифт може бити узрок озбиљних повреда, па чак и смрти. Кварови се могу манифестовати на много различитих начина, укључујући изненадне промене брзине, неправилна ломљења или неисправне кочионе системе. Непоштовање одредаба правних аката којима је регулисано питање правилног прегледа, одржавања и поправке лифта. Жртве могу да претрпе тешке телесне повреде и повреде са смртним исходом.

3.1. Поступак у случају спашавања путника из лифта

1. Овлашћено лице треба да комуницира са особом или особама које су заглављене унутар кабине лифта. Сазнати колико особа је унутар кабине и упутити их да остану мирни и да се удаље од врата кабине.

2. Овлашћено лице искључује електрични довод на главној командној табли у машинској просторији, за лифт у којем су лица заглављена.

3. Отворити врата возног окна са специјалним кључем, на спрату испод или изнад места где је лифт заглављен.

4. Ако се кабина лифта налази на нивоу више од ± 600 mm до најближе станице (спрата), онда не вадити путнике. Спасилачке радове треба да врши техничко (стручно) лице, јер постоји опасност да поступци спасавања могу довести у ризик од фаталне несреће заглављене путнике у виду пада кроз отвор у возном окну. Ако кабина лифта није у нивоу да би путници безбедно изашли напоље, онда се понекад саветује да се врата отворе за око 200 mm да би свеж ваздух могао да уђе у кабину и да би се путници осећали удобније док чекају стручно лице. Стручна лице су обучена за ручно довођење кабине лифта до најближег нивоа

(станице), тако што једна особа лагано отпушта кочницу лифта, а друга окретањем замајца на електромотору спушта или подиже кабину, у зависности шта је потребно. НАПОМЕНА: ОВО ЈЕ ОПЕРАЦИЈА КОЈУ МОРА ДА УРАДИ САМО ОБУЧЕНО ТЕХНИЧКО ЛИЦЕ.

4. ПРЕГЛЕД И ПРОВЕРА ЛИФТА

4.1. Редовни преглед лифта

Власник лифта обезбеђује редован преглед лифта.

Редован преглед лифта се обавља најмање једном годишње, а обавља га Именовано тело за преглед лифта.

Лице које обавља послове одржавања лифта присуствује редовном прегледу лифта.

У поступку редовног прегледа лифта проверава се:

• исправан рад опреме за безбедност и заштиту,
• исправност друге опреме која би могла да утиче на безбедност,
• да ли су настале промене на лифту које могу да утичу на безбедност,
• да ли су настале промене у окружењу које могу да утичу на безбедност,
• да ли долази до промена код употребе лифта које могу да утичу на безбедност,
• да ли се на лифту налазе све ознаке и упутства за употребу, одржавање и спашавање лица из лифта,
• да ли су у књигу одржавања лифта уписане све промене настале од последњег редовног прегледа,
• да ли су од последњег редовног прегледа уклоњени сви недостаци који су утврђени у Извештају о прегледу.

Након прегледа Именовано тело сачињава Извештај о прегледу, који садржи све евентуалне недостатке на лифту, поступке за отклањање и рок отклањања.

Ако лифт не задовољава горе наведене тачке тако да је нарушена безбедност корисника, Именовано тело ставља лифт ван употребе и обавештава инспектора, одржаваоца и власника лифта. Изузетно, ако безбедност корисника није битно нарушена, Именовано тело може да дозволи експлоатацију лифта у одређеном временском периоду, за које је власник лифта дужан да отклони све документоване недостатке [4].

ПРИМЕДБЕ И НЕДОСТАЦИ
- У књигу одржавања уписати податке о лифту према Правилнику о прегледима лифтова у употреби и водити евиденцију о прегледима, сервисима, замени делова и др. (рок 1 месец). - Није означена носивост монтажних носача у врху возног окна. Поред носача поставити ознаку носивости од стране компетентне особе (рок 1 месец). - Измерена сила која је потребна за спречавање затварања аутоматских врата возног окна и кабине је 430 N пре него што се иста отворе, што је веће од максималне дозвољене силе (150 N). Подесити силу која спречава затварање врата тако да буде у дозвољеним границама (рок 1 месец). - У кабини лифта недостаје упутство за безбедно коришћење лифта. Поставити такво упутство у кабину (рок 1 месец). - Уређај за звучно упозорење на кабини лифта није у функцији. Довести га у функцију (рок 1 месец).
ЗАКЉУЧАК
Контролисањем (прегледом) је утврђено да ПРЕДМЕТНИ ЛИФТ ЗАДОВОЉАВА битне захтеве за здравље и безбедност из Правилника о безбедности лифтова тако да није битно угрожена безбедност корисника. ***Рад лифта се дозвољава уз отклањање недостатака према датим роковима.***

Слика 2. *Примедбе, недостаци и закључак Извештаја о прегледу лифта*

4.2. Ванредни преглед лифта

Ванредни преглед лифта обавља Именовано тело за преглед лифта.

Власник лифта доставља Именованом телу за преглед лифта сву потребну документацију, пре прегледа лифта.

Ванредни преглед лифта мора да обави исто Именовано тело за преглед лифта које је издало негативан Извештај о прегледу.

Ванредни преглед спроводи се у случају:
• насталих основних промена на лифту,
• стављања лифту у употребу после незгода,
• захтев надлежног инспектора.

Именовано тело за преглед лифта сачињава Извештај о прегледу и уписује у књигу одржавања лифта датум када је преглед обављен, као и резултате прегледа. Такође, води евиденцију обављених прегледа са подацима који су истоветни подацима који су уписани у књигу одржавања лифта. Подаци се достављају министарству, односно инспектору на његов захтев [4].

5. ЗАКЉУЧАК

Као и све механичке машине и уређаји, и код лифтова се повремено јавља прекид рада. Лифт који није у функцији може бити резултат нестанка струје, механичког квара или сигурносног система дизајнираног да заустави лифт.

Дакле, стручни преглед лифта је неопходна и одговорна пракса која има за циљ очување живота и здравља корисника, очување функционалности машина, као и поштовање законских прописа и техничких норми.

6. ЛИТЕРАТУРА

[1] С.Б. Тошић, *"Лифтови"*, Универзитет у Београду, Машински факултет, Центар за механизацију, Београд, 2004.
[2] Правилник о безбедности лифтова („Сл. гласник РС", бр. 15/2017 и 21/2020)
[3] Д.Б. Ђорђевић, "Социолошко казивање о лифту (у, испред, иза и около њега – Изводи из азбучника", *Социолошка Луча*, Vol 13, Issue 2, pp. 46-69, 2019.
[4] Правилник о прегледима лифтова у употреби („Сл. гласник РС", бр. 15/2017)

Кратка биографија:

Зоран Цветковић је рођен у Новом Саду 1986. године. Дипломирао је на Факултету техничких наука на смеровима Инжењерство заштите животне средине 2012. године и Инжењерство заштите на раду 2023. године. Мастер рад из области Машинског инжењерства одбранио је 2025. године. Суоснивач је предузећа НС Превент д.о.о. Супруг и родитељ 3 детета.
контакт: cvelisans@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.316.925
DOI: https://doi.org/10.24867/33AM02Radic

ZNAČAJ RAČUNARSKE GRAFIKE U AUTOMATIZOVANOM PROJEKTOVANJU MAŠINA

THE IMPORTANCE OF COMPUTER GRAPHICS IN AUTOMATED MACHINE DESIGN

Nemanja Radić, Radomir Đokić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – MAŠINSKO INŽEWERSTVO

Kratak sadržaj – *U master radu su obrađene tehnike i metode koje su najčešće u primjeni u mašinstvu i koje čine osnovu automatizovanog projektovanja kao što su zapreminsko modelovanje, parametarsko modelovanje i dr. Dati su neki od mnogobrojnih primjera njihove primjene u projektovanju i njihove prednosti i mane. Takođe, u ovom radu su opisane i nove metode koje su tek u početnim fazama implementacije kao što su vještačka inteligencija, virtuelna i proširena realnost i mogućnost njihovog kombinovanja sa tradicionalnim metodama automatizovanog projektovanja.*

Ključne reči: *Računarska grafika, automatizovano projektovanje mašina*

Abstract – *In the master's thesis, the techniques and methods most commonly used in mechanical engineering, which form the basis of automated design, such as solid modeling, parametric modeling, and others, were analyzed. Several examples of their application in design, along with their advantages and disadvantages, were presented. Additionally, the thesis described new methods that are still in the early stages of implementation, such as artificial intelligence, virtual reality, and augmented reality, as well as the possibility of combining them with traditional automated design methods.*

Keywords: *Computer graphics, Automated machine design*

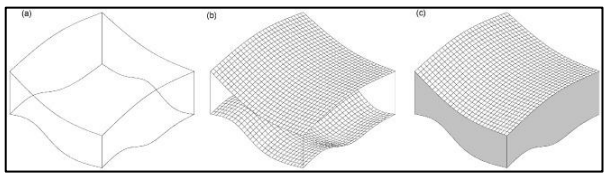
1. UVOD

Računarska grafika i kompjuterski podržano projektovanje (CAD) predstavljaju temelj savremenog mašinskog inženjerstva, mijenjajući način na koji se mehanički elementi i sistemi zamišljaju, vizualizuju i realizuju. Ovaj rad bavi se različitim aspektima inženjerske grafike i CAD-a, pružajući razumijevanje njihovog značaja, tehnika i primjena u mašinskom projektovanju. Tokom ovog istraživanja istaknut je razvoj inženjerske grafike od ručnog crtanja do digitalnog projektovanja, naglašavajući ulogu vizualizacije u efikasnoj komunikaciji u mašinskom projektovanju. Analizirani su osnovni koncepti ortogonalnih projekcija, geometrijskog modeliranja, parametarskog projektovanja, otkrivajući kako ove tehnike unapređuju efikasnost i tačnost projektovanja.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Radomir Đokić, vanr. prof.

1.1. Automatizovano projektovanje

Automatizovano projektovanje mašina predstavlja primjenu računarskih tehnologija i softvera u procesu kreiranja, analize i optimizacije mašinskih konstrukcija. Osnova automatizovanog projektovanja nalazi se u softverskim alatima kao što su CAD (Computer-Aided Design) i CAE (Computer-Aided Engineering). Iako danas postoje brojne različite tehnike automatizovanog projektovanja, osnovu svih tih tehnika čine geometrijski modeli i operacije koje se vrše nad njima, odnosno Boole-ove operacije (unija, razlika i presjek). Postoje dva osnovna tipa geometrijskih modela: dvodimenzionalni model koji se koristi za tehničko crtanje i trodimenzionalni model koji se koristi za računarski potpomognuto projektovanje i proizvodnju. U zavisnosti od reprezentacije objekata, sistemi geometrijskog modelovanja mogu se klasifikovati u tri kategorije, a to su zapreminsko modelovanje, modelovanje površina i modelovanje žičanog modela, slika 1, [1-2].



Slika 1. *Žičani model, površinski model i zapreminski model (slijeva na desno) [1]*

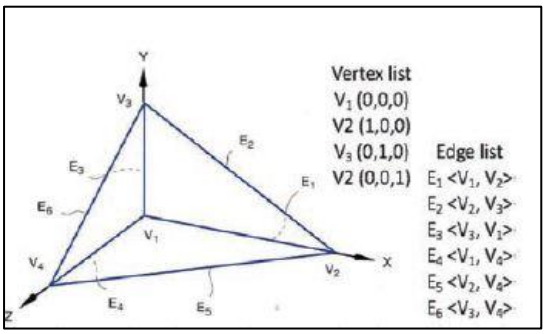
2. TEHNIKE AUTOMATIZOVANOG PROJEKTOVANJA

Tehnike projektovanja predstavljaju niz strategija i metoda koje inženjeri koriste kako bi najučinkovitije razvili nove proizvode, strukture ili sisteme. Projektovanje uključuje brojne korake: od početne ideje i istraživanja, preko modelovanja i simulacija, pa sve do izrade prototipa i testiranja. Svrha ovih tehnika je da olakšaju kreiranje rješenja koja su što bliža zahtjevima funkcionalnosti, estetike, troškova i trajnosti. Automatizovano projektovanje podrazumijeva interaktivan rad između inženjera i računara, gdje se spajaju kreativnost čovjeka i obradna moć softverskih alata. Tehnike projektovanja koje danas imaju najširu primjenu su:

1. Formiranje žičanog modela, 2. Površinsko modeliranje, 3. Zapreminsko modeliranje, 4. Parametarsko modeliranje, 5. Generativni dizajn, 6. Inverzno (reverzibilno) inženjerstvo i 7. tzv. „Kinematičko“ modeliranje. Svaka od ovih tehnika ima posebnu ulogu i pruža specifične mogućnosti koje unapređuju proces projektovanja.

2.1. Formiranje žičanog modela

Žičani model predstavlja jedan od najosnovnijih i najstarijih oblika grafičkog prikaza 3D objekata, koji se široko koristi u mašinstvu. Ova tehnika modeliranja zasniva se na upotrebi linija, tačaka i krivih za definisanje geometrijskih odnosa i oblika objekata, bez potrebe za definisanjem detalja kao što su površina ili zapremina. Tačke predstavljaju koordinate koje definišu ključne dijelove objekta, dok linije i krive povezuju ove tačke i formiraju ivice koje određuju oblik objekta, kao što je to prikazano na slici 2.

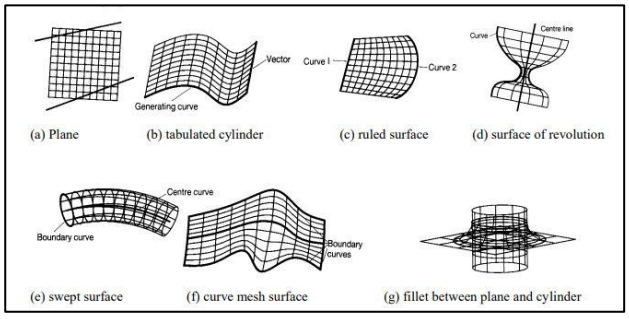


Slika 2. *Žičani model tetraedra [2]*

Iako ne pruža potpunu predstavu o fizičkim svojstvima objekta, žičani model ima važnu ulogu u ranim fazama dizajna i analize, posebno kada je potrebno istražiti osnovne geometrijske odnose i strukturu objekta.

2.2. Modeliranje površina

Modeliranje površina definiše komponentu sa većim matematičkim integritetom, jer modelira površine kako bi dalo preciznije prostorne granice dizajnu. Posebno je korisno za modeliranje objekata koji se mogu modelirati kao ljuske, kao što su paneli karoserije automobila, trupovi aviona ili lopatice ventilatora. Najosnovniji tip površina je ravna površina, prikazana na slici 3(a), koja se može definisati između dvije paralelne prave linije, kroz tri tačke ili kroz liniju i tačku. Drugi tipovi površina koje se obično koriste u CAD-u su: tabelarni cilindar, slika 3(b), loft, slika 3(c), površina dobijena rotacijom, prikazana na slici 3(d), sweep, slika 3(e), skulpturalna ili površina sa mrežom krivih, slika 3(f) i tzv. filet površina kao prelaz sa cilindra na ravan, slika 3(g).



Slika 3. *Tipovi površina u CAD softverima [3]*

Jedna od značajnih prednosti površinskog modeliranja je u tome što omogućava inženjerima da rade u interaktivnom okruženju. Kada se promijeni neka površina, modifikacije se odmah odražavaju na cjelokupnom modelu, što olakšava projektovanje i optimizaciju. Ova sposobnost da se brzo prilagodi i izmijene modeli čini površinsko modeliranje izuzetno korisnim alatom u dinamičnom okruženju kao što je savremena industrija.

2.3. Zapreminsko modeliranje

Zapreminsko modeliranje je jedna od osnovnih tehnika u inženjerskom projektovanju koja se koristi za stvaranje trodimenzionalnih zapreminskih modela. Ova metoda omogućava inženjerima da kreiraju realistične predstave fizičkih objekata, uključujući njihova stvarna fizička svojstva kao što su masa, zapremina, centar gravitacije i moment inercije. Ove informacije su od suštinskog značaja za analizu i optimizaciju dizajna, posebno u složenim projektima. Šeme predstavljanja 3D modela mogu se podijeliti u šest opštih klasa, a to su čista primitivna instanca, generalizovano izvlačenje, prebrojavanje prostorne okupacije, ćelijska dekompozicija, sastavljanje modela iz 3D primitiva (CSG) - model gomerijske konstrukcije i definisanje granica objekta (B-rep) - model granične prezentacije. Šeme predstavljanja koje se najčešće koriste u komercijalnim modelima su CSG i B-rep. Kada je u pitanju grafička dokumentacija, zapreminsko modeliranje u specijalizovanim softverima, kao što su Autodesk Inventor i CATIA V5, olakšava kreiranje tehničkih crteža i specifikaciju delova. Inženjeri mogu automatski generisati 2D crteže iz 3D zapreminskih modela, čime se ubrzava proces kreiranja dokumentacije i smanjuju mogućnosti grešaka.

2.4. Parametarsko modeliranje

Parametarsko modeliranje predstavlja ključnu tehniku u automatizovanom projektovanju. Ova tehnika omogućava inženjerima da na efikasan način definišu i upravljaju osnovnim dimenzijama i karakteristikama geometrijskih modela. Parametarsko modeliranje odnosi se na kreiranje 3D CAD modela u kojima je geometrija definisana i ograničena parametrima. Ovi parametri mogu biti dimenzije, matematički odnosi, osobine materijala i drugo. Ključna stvar je da se, kada se promijene vrednosti parametara, geometrija automatski ažurira prema unapred definisanim pravilima. Ovaj pristup omogućava neviđenu fleksibilnost i automatizaciju u procesu dizajniranja. Zahvaljujući mogućnosti brzih izmena kompletne geometrije promenom parametara, inženjeri mogu brzo raditi na varijantama dizajna, optimizovati komponente i standardizovati sklopove.

2.5. Generativni dizajn

Generativni dizajn je tehnika koja transformiše način na koji inženjeri i dizajneri pristupaju razvoju proizvoda. Ovaj proces omogućava inženjerima da, na osnovu definisanih ulaznih parametara (kao što su materijali, opterećenja, ograničenja prostora i troškovi), generišu veliki broj mogućih rešenja za određeni problem. Ova tehnika je posebno značajna u sektorima gde je smanjenje težine ključno, kao što su avijacija i raketna industrija, gde svako smanjenje mase može dovesti do značajnih ušteda u gorivu i poboljšanja u performansama. U pogledu troškova, generativni dizajn može dovesti do ušteda od 10% do 30% u poređenju sa tradicionalnim metodama, [4].

2.6. Reverzibilno inženjerstvo

Reverzibilno inženjerstvo predstavlja metodologiju koja podrazumeva analizu i dekonstukciju gotovog proizvoda s ciljem sticanja dubljeg razumevanja njegove strukture,

funkcionalnosti i principa rada. Ova tehnika se često koristi u inženjerstvu za razvoj novih proizvoda, reinženjering postojećih rešenja, kao i za usavršavanje procesa proizvodnje. Proces reverzibilnog inženjeringa započinje fizičkim proizvodom, gde inženjeri razgrađuju komponente, analiziraju ih, a zatim koriste te podatke za stvaranje novog dizajna (varijanti) ili za poboljšanje postojećih. Iako reverzibilno inženjerstvo donosi brojne prednosti, kao što su smanjenje troškova, brža proizvodnja i unapređenje postojećih tehnologija, ono sa sobom nosi i određene izazove. Jedan od njih su pravna pitanja koja se odnose na intelektualnu svojinu, jer kopiranje dizajna bez odobrenja može dovesti do pravnih komplikacija.

2.7. „Kinematičko“ modeliranje

„Kinematičko“ modeliranje je tehnika koja se koristi za analizu kretanja komponenti u sistemu, što je od suštinskog značaja u projektovanju mehanizama. Ova metoda omogućava inženjerima da proučavaju kretanje dijelova u mehaničkom sistemu, što je ključno za optimizaciju performansi i funkcionalnosti dizajna. Shvatanje kretanja komponenti pomaže inženjerima da identifikuju oblasti u kojima mogu doći do neusaglašenosti ili neželjenih dejstava tokom rada. Na primjer, u razvoju robota ili automobila, „kinematičko“ modeliranje može pomoći u analizi kako različiti dijelovi utiču na rad mehanizma, čime se osigurava da sve komponente rade sinhronizovano i bez problema.

3. INOVACIJE I BUDUĆNOST AUTOMATIZOVANOG PROJEKTOVANJA

Automatizovano projektovanje mašina predstavlja dinamičnu i inovativnu oblast inženjerstva koja se neprestano razvija, a nove tehnologije i metode značajno utiču na način na koji inženjeri pristupaju dizajnu i razvoju mehaničkih sistema. Napredak u tehnologiji, naročito u oblasti vještačke inteligencije (AI) i mašinskog učenja, dovodi do transformacije tradicionalnih metoda projektovanja, omogućavajući inženjerima da istražuju nove dizajnerske koncepte i optimizuju postojeća rješenja. Koristeći algoritme vještačke inteligencije i modelovanje podataka, inženjeri mogu analizirati velike količine informacija, identifikovati obrasce i predvidjeti ponašanje sistema.

Budućnost automatizovanog projektovanja obećava da će biti obilježena naprednom integracijom tehnologija, povećanom saradnjom među timovima i većom sposobnošću inženjera da brzo reaguju na promjene u zahtjevima tržišta, čime se poboljšavaju kvalitet i efikasnost proizvoda. Neke od novih metoda koje su obrađene u ovom radu su vještačka inteligencija i mašinsko učenje, 3D štampanje i aditivna proizvodnja, oblak i kolaborativne platforme, virtuelna stvarnost i proširena stvarnost.

3.1. Vještačka inteligencija i mašinsko učenje

Oblast vještačke inteligencije (AI) dobija veliku pažnju zbog sposobnosti da efikasno analizira i djeluje na ogromnu količinu prikupljenih podataka. Nagli porast interesovanja se uglavnom pripisuje napretku ostvarenim u podoblasti mašinskog učenja (ML), kao i pratećim faktorima, kao što su skladištenje podataka i računarske snage. Sistemi ove tehnologije postaju moćni alati koji se mogu koristiti i mogu dovesti do bržeg, racionalnijeg donošenja odluka, kao i efikasnijeg rada. Vještačka inteligencija (AI) je oblast računarskih nauka koja se može koristiti za razvoj inteligentnih računara koji mogu djelovati, razmišljati i donositi odluke slično ljudima. Vještačka inteligencija je pokazana kada mašina ima sposobnosti slične ljudskim, kao što su učenje, razmišljanje i rješavanje problema. S druge strane, mašinsko učenje (ML) je oblast vještačke inteligencije (AI) i računarskih nauka koja se fokusira na korišćenje podataka i algoritama kako bi se imitirao način na koji ljudi uče i postepeno poboljšala preciznost.

Nauka o podacima je rastuća disciplina, a mašinsko učenje predstavlja njen ključni segment. U inicijativama za rudarenje podataka, algoritmi za klasifikaciju i predviđanje se obučavaju korišćenjem statističkih metoda, pružajući važne uvide. Ove informacije zatim olakšavaju donošenje odluka unutar aplikacija i organizacija, sa idealnim uticajem na ključne mjere rasta. Vještačka inteligencija i mašinsko učenje predstavljaju ključne elemente u budućnosti automatizovanog projektovanja. Ove tehnologije ne samo da poboljšavaju kvalitet i efikasnost dizajna, već i otvaraju nove mogućnosti za inovacije.

3.2. 3D štampanje i aditivna proizvodnja

3D štampanje i aditivna proizvodnja su postali ključni alati u oblasti automatizovanog projektovanja, doprinoseći značajnom unapređenju u načinu na koji inženjeri razvijaju i proizvode složene komponente i sisteme. Ove tehnologije ne samo da omogućavaju brzu izradu prototipa, već i unapređuju mogućnosti dizajna, čime se smanjuju troškovi i vreme potrebno za razvoj. 3D štampanje je proces koji podrazumeva stvaranje fizičkog objekta iz digitalnog modela. Ovaj proces se izvodi sloj po sloj, gde se materijali dodaju jedan na drugi sve dok se ne postigne željeni oblik. Aditivna proizvodnja omogućava proizvodnju komponenti složenih oblika koje bi bilo teško ili nemoguće napraviti tradicionalnim metodama obrade. Na primer, inženjeri mogu stvoriti detaljne geometrije, unutrašnje šupljine i strukturne obrasce koji poboljšavaju funkcionalnost komponenti. Nove tehnike, kao što su 4D štampanje, koje podrazumeva proizvodnju objekata koji se mogu prilagođavati promenljivim uslovima, otvaraju nove mogućnosti za inovaciju. Takođe, integracija sa tehnologijama kao što su veštačka inteligencija može dovesti do još složenijih i pametnijih sistema.

3.3. Oblak (Cloud) i kolaborativne platforme

U današnjem globalizovanom svijetu, gdje fizičke granice postaju sve manje važne, oblak tehnologije i kolaborativne platforme igraju ključnu ulogu u automatizovanom projektovanju. Ove tehnologije omogućavaju inženjerima da rade na projektima iz različitih geografskih lokacija, čime se podstiče inovacija i unapređuje efikasnost u projektovanju mehaničkih sistema. Oblak tehnologije podrazumijevaju korišćenje interneta za skladištenje, upravljanje i obradu podataka, umjesto lokalnih servera ili računarskih sistema. Ova tehnologija omogućava inženjerima i dizajnerima da pristupe potrebnim resursima i alatima u bilo kom trenutku, sa bilo kog mjesta. Kolaborativne platforme, kao što su Autodesk Fusion 360, SolidWorks 3DExperience i Microsoft Teams, integrišu funkcionalnosti oblaka s alatima za timsku saradnju, čime

se poboljšava komunikacija i razmjena informacija u timu. Oblak tehnologije i kolaborativne platforme predstavljaju značajan napredak u automatizovanom projektovanju. One omogućavaju inženjerima da sarađuju u realnom vremenu, dele podatke i optimizuju proces projektovanja. Uspostavljanjem efikasne komunikacije i smanjenjem grešaka, ove tehnologije poboljšavaju kvalitet proizvoda i ubrzavaju razvoj. Kako se tehnologija razvija, kolaborativne platforme će nastaviti da igraju ključnu ulogu u budućnosti automatizovanog projektovanja, otvarajući nove mogućnosti za inovaciju i efikasnost, [5].

3.4. Virtualna stvarnost i proširena stvarnost

Virtuelna stvarnost (VR) i proširena stvarnost (AR) su inovacije koje značajno mijenjaju način na koji inženjeri i dizajneri pristupaju projektovanju i razvoju mehaničkih sistema. Ove tehnologije ne samo da poboljšavaju vizualizaciju i analizu dizajna, već i unapređuju saradnju i komunikaciju u timu, što dovodi do efikasnijeg i inovativnijeg procesa projektovanja. Virtuelna stvarnost podrazumijeva stvaranje potpuno digitalnog okruženja u kojem korisnici mogu interaktivno učestvovati. Korišćenjem VR tehnologije, inženjeri mogu da projektuju i testiraju složene sisteme u simuliranoj sredini, što im omogućava da istraže različite dizajnerske opcije bez potrebe za fizičkim prototipima. Proširena stvarnost kombinuje digitalne elemente sa stvarnim svijetom, stvarajući interaktivno iskustvo koje omogućava inženjerima da na realne objekte projektuju digitalne informacije. Ova tehnologija predstavlja značajan napredak u procesima montaže, održavanja i obuke. U poređenju sa statičkim 3D modelima ili tradicionalnim 2D crtežima, tehnologija proširene stvarnosti nudi tro-dimenzionalnu perspektivu koja je značajno intuitivnija i informativnija, što poboljšava prostorno razmatranje.

4. ZAKLJUČAK

Računarska grafika i kompjuterski podržano projektovanje (CAD) predstavljaju temelj savremenog mašinskog inženjeringa, menjajući način na koji se mehanički elementi i sistemi zamišljaju, vizualizuju i realizuju. Ovaj rad bavi se različitim aspektima inženjerske grafike i CAD-a, pružajući razumevanje njihovog značaja, principa, tehnika i primena u mašinskom projektovanju. Tokom ovog istraživanja istaknut je razvoj inženjerske grafike od ručnog crtanja do digitalnog projektovanja, naglašavajući ključnu ulogu vizualizacije u efikasnoj komunikaciji u mašinskom projektovanju. Analizirani su osnovni koncepti ortogonalnih projekcija, geometrijskog modeliranja, parametarskog projektovanja i crtanja, otkrivajući kako ove tehnike unapređuju efikasnost, tačnost i svestranost projektovanja. Istraživanjem spektra CAD softvera i alata, prikazano je kako inženjeri koriste digitalne platforme za kreiranje složenih 3D modela, simulaciju mehaničkog ponašanja i besprekornu saradnju između timova koji se nalaze na različitim geografskim lokacijama.

Integracija parametarskog projektovanja, 3D modeliranja i principa sklopova omogućava inženjerima brzu iteraciju, optimizaciju projekata i virtuelnu validaciju performansi pre fizičke realizacije. Pored toga, važnost tehničkog crtanja i dokumentacije ne sme se potceniti, jer tačni tehnički crteži, liste materijala (BOM) i prateća dokumentacija predstavljaju osnovu komunikacije u projektovanju, obezbeđujući uspešan prenos zamisli u opipljive proizvode.

Kako mašinski inženjeri nastavljaju da prihvataju moć računarske grafike i CAD-a, stiču sposobnost da inoviraju, unapređuju procese projektovanja i realizuju nove ideje. Bilo da je reč o kreiranju složenih 3D modela, simulaciji kompleksnih mehaničkih sistema ili komunikaciji tehničkih specifikacija, ovladavanje ovim principima i tehnikama vodi mašinski inženjering u eru neviđene kreativnosti i efikasnosti.

5. LITERATURA

[1] A. Dag, A. Özdemir, "A Comparative Study for 3D Surface Modeling of Coal Deposit by Spatial Interpolation Approaches", *Resource Geology*, 63 (4), pp. 394-403, 2013.

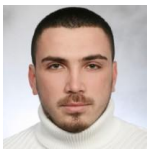
[2] Z. Jelić, B. Popkonstantinović, M. Stojičević, *"Usage of 3D Computer Modelling in Learning Engineering Graphics"*, 2016.

[3] Anonim., *"*Surface modeling*"*, İTÜ - Transport Tekniği Grubu, Istanbul, 2009.

[4] M. Fenooh, O. Alquabeh, M.M. Nisar, S. Zia, "Generative Design of a Mechanical Pedal", *International Journal of Engineering and Management Sciences*, (6), pp. 48-56, 2021.

[5] V. Gharibvand, M.K. Kolamroudi, Q. Zeeshan et al., "Cloud based manufacturing: A review of recent developments in architectures, technologies, infrastructures, platforms and associated challenges", *The International Journal of Advanced Manufacturing Technology*, (131), pp. 93-123, 2024.

Kratka biografija:



Nemanja Radić rođen je u Bijeljini 1997. god. Master rad na Fakultetu tehničkih nauka iz oblasti Mašinskog inženjerstva – Mehanizacija i konstrukciono mašinstvo odbranio je 2025. god.
kontakt: nemanja.radic997@gmail.com

Radomir Đokić,vanredni profesor na Fakultetu tehničkih nauka u Novom Sadu. Doktorirao je na Fakultetu tehničkih nauka 2016. god. Oblast interesovanja su projektovanje mobilnih mašina i konstrukcija.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.7/.9
DOI: https://doi.org/10.24867/33AM03Ivanovic

RIZICI USLED PARKIRANJA VOZILA SA POGONOM NA TEČNI NAFTNI GAS U ZATVORENIM PROSTORIMA

RISKS POSED BY PARKING LIQUEFIED PETROLEUM GAS POWERED VEHICLES IN ENCLOSED SPACES

Nina Ivanović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – MAŠINSTVO

Kratak sadržaj – *Ovaj rad analizira fizičko-hemijske karakteristike tečnog naftnog gasa i njegovu primenu kao pogonskog goriva, sa posebnim fokusom na rizike usled parkiranja vozila na TNG u zatvorenim garažama. Istraženi su fenomeni VCE i BLEVE, mehanizmi akumulacije gasa, rizici od eksplozije, kao i uticaj ventilacije i detekcije na bezbednost. Obrađeni su efekti kontakta TNG-a sa ljudskim tkivom i predložene mere zaštite, uz osvrt na važeće propise i neophodnost integracije tehničkih i organizacionih rešenja u upravljanju rizicima povezanim sa primenom TNG-a u zatvorenim prostorima.*

Ključne reči: *Tečni naftni gas, zatvorene garaže, ventilacija i detekcija, BLEVE, bezbednost i zdravlje*

Abstract – This paper analyzes the physicochemical characteristics of liquefied petroleum gas and its use as a fuel, with emphasis on the risks associated with parking LPG-powered vehicles in enclosed garages. The study explores the phenomena of VCE and BLEVE, mechanisms of gas accumulation, explosion hazards, and the impact of ventilation and detection systems on safety. It also addresses the effects of LPG contact with human tissue and proposes protective measures, with reference to relevant regulations and the necessity of integrating technical and organizational solutions in risk management related to LPG use in confined spaces.

Keywords: *Liquefied petroleum gas, enclosed garages, ventilation and detection, BLEVE, safety and health*

1. UVOD

Vozila na tečni naftni gas (TNG) poznata su po ekološkoj prihvatljivosti, ali njihovo prisustvo u zatvorenim garažama nosi rizike stvaranja eksplozivnih koncentracija gasa. Oslobađanje TNG-a može izazvati ozbiljne incidente poput fenomena BLEVE i VCE, sa potencijalno teškim posledicama po ljude i imovinu. Čak i male količine u neprovetravanom prostoru mogu izazvati eksploziju u prisustvu izvora paljenja. Zato su mere zaštite, kao što su ventilacija i sistemi detekcije gasa, kao i sigurnosni uređaji na vozilima, neophodni za sigurnost u urbanim garažama.

2. TEČNI NAFTNI GAS

TNG je mešavina propana i butana, poznata i kao propan-butan gas. Oko 60% se dobija iz prirodnog gasa, a ostatak iz nafte. Sastav varira zavisno od klimatskih uslova: u hladnijim krajevima je veći udeo propana zbog bolje isparljivosti, dok se u toplijim više koristi butan. U Srbiji je odnos približno jednak [1].

2.1. Fizičke i hemijske karakteristike

Tečni naftni gas je bezbojan, visoko zapaljiv i eksplozivan, ali bez mirisa. Za lakšu detekciju, komercijalnom TNG-u dodaje se etil-merkaptan, organsko jedinjenje sa sumporom, što omogućava otkrivanje čak i pri niskim koncentracijama. TNG ima gotovo dvostruko veću gustinu od vazduha, zbog čega se smatra zagušljivcem koji istiskuje kiseonik. Slabo je rastvorljiv u vodi, što olakšava skladištenje i transport jer se ne meša s vodom. Iako nije toksičan, pri visokim koncentracijama može delovati kao anestetik i izazvati gušenje zbog nedostatka kiseonika. Kontakt s kožom može izazvati promrzline zbog intenzivnog isparavanja, dok je TNG agresivan prema materijalima kao što su guma i plastika [1].

Granica zapaljivosti određuje koncentraciju zapaljive materije u mešavini koja može izazvati eksploziju. Za smešu propan-butana u odnosu 35:65, donja granica zapaljivosti je 2%, a gornja 9% zapreminskog udela u vazduhu. Sagorevanjem TNG oslobađa toplotu i proizvodi ugljen-dioksid i vodenu paru, s temperaturom plamena do 1900°C. TNG karakteriše i visoki napon zasićenih para. Kod butana, pritisak pare iznosi 0,005 bara na 0°C i 0,8 bara na 15°C, a kod propana – 4 bara na 0°C i 5–6 bara na 15°C. Temperatura ključanja propana je -43°C, dok je za butan -0,5°C [1].

2.2. Upotreba u vozilima

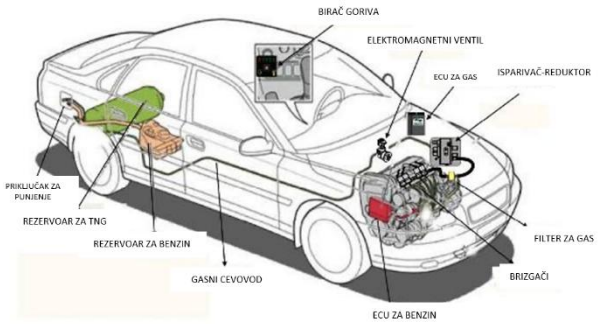
TNG predstavlja izuzetno pogodno gorivo za motore sa unutrašnjim sagorevanjem zbog brojnih prednosti koje pruža. Jedna od ključnih karakteristika TNG-a je njegova sposobnost da se brzo i ravnomerno meša sa vazduhom, čime formira homogenu smešu za sagorevanje. Tokom procesa sagorevanja, TNG ne proizvodi dim niti taloge, čak ni u uslovima promenljivog rada motora, što doprinosi njegovoj pouzdanosti i efikasnosti. Pored toga, TNG se izdvaja i po svojoj ekološkoj prihvatljivosti. Udeo vodonika u molekulima jedinjenja koja čine TNG je visok, te u produktima sagorevanja dominira vodena para, dok je emisija ugljen-dioksida smanjena. Emisija oksida azota je takođe snižena, a gasovi iz sagorevanja ne sadrže olovna i sumporna jedinjenja. Kao gas, TNG ne stvara kondenzat, čime se smanjuje rizik od razređivanja ulja i dodatno produžava vek trajanja motora. Za razliku od benzinskih rezervoara koji mogu eksplodirati na visokim temperaturama, rezervoari za TNG izrađeni su od čelika debljine 3 do 4 mm, anatomski su oblikovani i otporni na

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Nebojša Nikolić, van. prof.

deformacije pri sudarima. Uz to, ekonomska isplativost TNG-a omogućava brz povraćaj ulaganja u gasni sistem, što ga čini atraktivnim izborom za korisnike, ali postoje nedostaci kao što su manja snaga motora, dodatna masa vozila i potreba za posebnim uslovima skladištenja i rukovanja [1].

3. INSTALACIJA TNG SISTEMA U VOZILIMA

Instalacije za napajanje TNG-om obično predstavljaju alternativne sisteme goriva za motore koji rade na benzin ili dizel gorivo, koji blisko sarađuju sa originalnim sistemom napajanja gorivom [2]. Različiti tipovi gasnih instalacija obično koriste isti osnovni skup komponenti, koji je prikazan na Slici 1.



Slika 1 *Osnovne komponente instalacije za TNG [3]*

TNG se pod pritiskom u tečnom stanju od rezervoara transportuje do isparivača, koji funkcioniše i kao reduktor. Isparivač je povezan sa sistemom za hlađenje motora, omogućavajući toploj tečnosti iz motora da zagreje tečni gas koji u tom procesu isparava. Na taj način, isparivač omogućava prelazak TNG-a u gasovito stanje, njegovo zagrevanje na odgovarajuću temperaturu, i smanjuje pritisak gasa na oko 0,8 bara, što je potrebno za napajanje motora [1].

Gasni sistemi novije generacije koriste se kod vozila sa ubrizgavanjem goriva u više tačaka (*multi-point injection*). Kod ovakvih sistema, gas se brizgava u usisne kanale, što bliže usisnim ventilima, a zatim se, zajedno sa vazduhom, odvodi u radne cilindre. Na taj način, kod ove vrste instalacije nema potrebe za mešačem. Jedna od ključnih karakteristika ovog sistema je njegova visoka brzina rada [4].

Radni pritisak u rezervoaru za TNG iznosi između 6 i 10 bara. Rezervoari koji se ugrađuju u vozila moraju biti u skladu sa trenutno važećim standardom, SRPS EN 12805:2011, čime se osigurava bezbednost sistema na TNG [1].

Svi priključci rezervoara smešteni su u gasno nepropusnom kućištu povezanom sa ventilacionim kanalima. Kada dođe do curenja, gas izlazi u spoljašnu sredinu kroz ventilacioni sistem, čime se minimizira rizik od ulaska gasa u putnički prostor vozila [2].

Ventil za ograničavanje punjenja na 80% odgovoran je za prekidanje dovoda gasa kada nivo goriva dostigne 80% geometrijske zapremine rezervoara. Ova funkcija omogućava zadržavanje slobodnog prostora unutar rezervoara, čime se omogućava prilagođavanje zapremine TNG-a u zavisnosti od temperature okoline. Prepunjavanje rezervoara iznad 80% zapremine, usled visoke temperature okoline, može dovesti do porasta pritiska u rezervoaru i aktivacije sigurnosnog ventila [2].

Sigurnosni ventil za oslobađanje pritiska je sigurnosni uređaj namenjen ograničavanju maksimalnog pritiska unutar rezervoara na 27 ± 1 bar, i u slučaju prekomernog pritiska, omogućava ispuštanje TNG-a iz rezervoara u gasovitoj fazi. Gas iz ovog ventila ne sme biti usmeren ka izduvnoj cevi, u putnički prostor, u prostor za smeštaj motora, kao i u pravcu potencijalnog električnog varničenja. Pri brzom porastu temperature, delovanje sigurnosnog ventila možda neće biti dovoljno da pritisak svede na bezbedan nivo [2].

Protivpožarni ventil je sigurnosni uređaj opremljen termo-osiguračem koji se topi na unapred određenoj temperaturi od 120 °C ± 10 °C, čime se smanjuje pritisak kako bi se izbegla nekontrolisana deformacija ili curenje iz rezervoara. U slučaju požara, osigurač se topi i otvara otvor za brzo ispuštanje TNG-a iz rezervoara, čime se sprečava rizik od eksplozije [2].

4. RIZICI PRI SKLADIŠTENJU VOZILA SA POGONOM NA TNG

Rizik od isticanja TNG-a ograničava širu primenu vozila na ovaj pogon. Jedan od ciljeva istraživanja je prikaz metode kvantitativne analize rizika (QRA – *Quantitative Risk Analysis*) za upotrebu vozila na TNG u zatvorenim garažama. Metoda omogućava procenu rizika i dizajn ventilacionih i detekcionih sistema. Treba naglasiti važnost integracije CFD simulacija u QRA, čime se precizno procenjuje rizik od povreda ili smrtnog ishoda u incidentima s TNG-om [5].

4.1 Tehnika QRA

Suština kvantitativne procene rizika leži u odgovoru na tri ključna pitanja: Šta može poći po zlu? Kolika je verovatnoća za to? Koje su posledice? QRA je metod procene rizika koji se sprovodi odozgo ka dole i uključuje: definisanje neželjenih krajnjih stanja, identifikaciju početnih događaja koji mogu do njih dovesti, korišćenje dijagrama događaja i grešaka za razvoj scenarija nesreća, procenu verovatnoće svakog scenarija na osnovu dostupnih podataka i iskustava, kao i njihovo rangiranje prema učestalosti radi dobijanja pregleda rizika. Ovaj pristup omogućava strukturalno i detaljno razmatranje mogućih rizika i njihovih posledica [6].

4.2 CFD simulacije

CFD (*Computational Fluid Dynamics*) podrazumeva softversku simulaciju koja se može primeniti pri različitim uslovima, ali zahteva značajan nivo znanja i iskustva. CFD simulacije su posebno korisne u istraživanjima akcidenata, a verifikacija simulacionih rezultata se obično sprovodi eksperimentalnim putem kako bi se osigurala tačnost i pouzdanost predviđanja [7].

Primena CFD simulacija na problematiku isticanja TNG-a u neki prostor može se podeliti u tri kategorije [5]:

1. Analiza nenamernog curenja TNG-a i formiranja oblaka pare, uključujući njegovu veličinu i vreme detekcije.
2. Istraživanje razblaživanja oblaka pare putem ventilacije do koncentracije ispod donje granice zapaljivosti.

3. Procena posledica eksplozije zapaljivog oblaka, uključujući obim bljeska i nastali natpritisak.

4. REZULTATI I DISKUSIJA

Isparenja tečnog naftnog gasa mogu preći značajne udaljenosti od mesta ispuštanja do izvora zapaljenja, gde mogu izazvati požar, povratni plamen ili eksploziju. Pored toga, rezervoari u vozilima mogu oslabiti i pući ukoliko su izloženi visokim temperaturama [8].

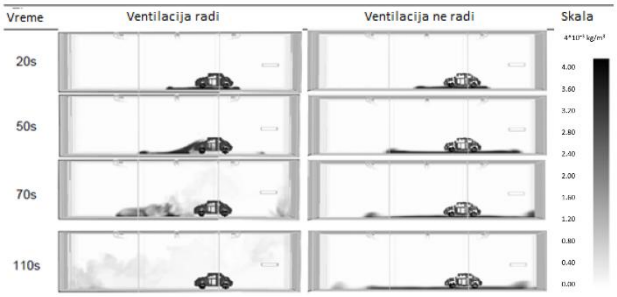
Najveća opasnost po život dolazi od letećih fragmenata koji se mogu odbijati od zidova prostorije, krećući se nepredvidivim putanjama i stvarajući rizik od povreda. Takođe, udarni talas može izazvati ozbiljna oštećenja sluha i destabilizovati strukturu zgrade, što dodatno povećava rizik od kolapsa ili drugih strukturalnih oštećenja [9].

U istraživanju [5], 7 od 26 analiziranih scenarija identifikovani su kao visokorizični usled akcidenta koji uključuje isticanje gasa ili eksploziju rezervoara za TNG. Najverovatniji među njima je požar vozila praćen mlaznim požarom usled otvaranja sigurnosnog ventila. Ako dođe do prepunjavanja rezervoara za TNG i porasta temperature, može doći do otvaranja sigurnosnog ventila i formiranja mlaza goriva koji, ukoliko se zapali, izaziva mlazni požar. U slučaju da sigurnosni ventil zakaže, pritisak u rezervoaru može dovesti do njegovog pucanja i BLEVE eksplozije, često praćene vatrenim bljeskom, vatrenom loptom ili eksplozijom oblaka pare. Spontano otvaranje ventila takođe može dovesti do mlaznog požara ili, ako se mlaz ne zapali odmah, do stvaranja oblaka pare koji predstavlja rizik od vatrenog bljeska. Ovi scenariji predstavljaju potencijalne opasnosti u slučaju neispravnog funkcionisanja sigurnosnih uređaja ili uslova koji mogu dovesti do eksplozivnih događaja ili požara.

Očigledno je da su požar ili eksplozija najgori mogući scenariji, i mogu se očekivati u dva slučaja – kada postoji izvor paljenja, ali samo u situacijama kada nema sistema za detekciju i ventilaciju. Ako oba sistema rade kako treba, može se očekivati samo ograničen požar ili rasprostiranje gasa [8]. Ovo ukazuje na važnost primene odgovarajućih sigurnosnih sistema, kao i adekvatno održavanje vozila kako bi se smanjila verovatnoća prepunjavanja i spontanog otvaranja sigurnosnih ventila [5].

U dosadašnjim istraživanjima istaknuta su dva osnovna tipa ventilacionih sistema koji se koriste u ovakvim prostorima: tradicionalni sistemi sa ventilacionim kanalima i sistemi mlaznih ventilatora. Prema [8], tradicionalni ventilacioni sistemi sa kanalima raspoređuju odvod vazduha tako da se 50 % ukupne izduvne zapremine odvodi blizu poda, dok se preostalih 50 % odvodi ispod plafona garaže. Nasuprot tome, sistem mlaznih ventilatora koristi aktivne uređaje za usmeravanje vazduha, sa ciljem razbijanja i razblaživanja oblaka gasa. Eksperimentalni rezultati pokazuju značajne razlike u performansama ova dva sistema.

Tokom kontinuiranog rada tradicionalnog kanalskog sistema, oblak gasovitog TNG-a se zadržava pri podu, a uticaj ventilacije na njegovo uklanjanje je minimalan. S druge strane, uključivanjem sistema mlaznih ventilatora, primećuje se brzo razbijanje oblaka i njegovo ravnomerno rasprostiranje kroz prostor garaže. Nakon nekoliko minuta kontinuiranog rada, gas se razblažuje i stvara se ujednačena gustina u celom prostoru (Slika 2), što značajno smanjuje opasnost od lokalizovanih visokih koncentracija gasa [8].



Slika 2 *Koncentracija TNG-a u slučaju sa i bez aktivne ventilacije (4*10⁻³ kg/m³ na skali odgovara 10 % donje granice eksplozivnosti) [8]*

Istraživanje naglašava da je u slučaju nenamernog ispuštanja TNG-a najverovatnija kratkotrajna pojava eksplozivne atmosfere, u trajanju od 10 do 30 sekundi. Takođe, koncentracija TNG-a je znatno veća na visini od 10 cm u odnosu na 30 cm, što ukazuje na važnost postavljanja gasnih detektora što bliže podu i blizu izvora emisije za pravovremenu detekciju. Utvrđeno je da je detekcija TNG-a na visini od 10 cm u proseku 40 % verovatnija nego na visini od 30 cm [8].

Zaključeno je da detektori TNG-a moraju biti postavljeni što bliže podu, a ventilacioni sistemi treba da efikasno ventiliraju donje delove garaža. Rezultati testova pokazuju da sistemi mlaznih ventilatora koji se pale po potrebi i sistemi sa vazdušnim kanalima često nisu dovoljni za ovaj zadatak, već je potreban kontinuiran rad ventilacije kako bi se obezbedila pravovremena reakcija [8].

Sprovedena je kvantitativna analiza rizika u garaži dimenzija 30×30 metara sa kapacitetom od 40 parking mesta, koristeći CFD simulacije [5]. Simulacija je omogućila proračun formiranja zapaljivog oblaka pare i njegovog razblaživanja, kao i procenu natpritisaka izazvanih eksplozijom. Pretpostavlja se da zapaljiv oblak pare, ukoliko se formira, ostaje prisutan u garaži dva minuta. To znači da, u slučaju da ventilacioni sistem ne radi neprekidno, oblak zapaljivih para mora biti detektovan putem sistema za detekciju gasa i zatim razblažen ventilacijom u roku od dva minuta. Stoga, trajanje prisustva zapaljivog oblaka para predstavlja jedan od ključnih parametara u dizajnu sistema za detekciju gasa i ventilaciju.

Vertikalno oslobađanje TNG-a brzinom od 0,55 kg/s može dovesti do stvaranja stehiometrijskih oblaka pare preko 100 m³ u roku od 50 sekundi, što približno odgovara vremenu potrebnom za pražnjenje rezervoara goriva zapremine 70 l. Oblak pare ove veličine može imati značajne posledice. Dok za isti vremenski period, ispuštanje gasa brzinom od 0,21 kg/s dovodi do stvaranja oblaka od oko samo 5 m³, koji neće imati značajne posledice čak i ako dođe do zapaljenja jer će natpritisak biti manji od 0,02 bara. Prilikom ovakvih akcidenata, natpritisak od 0,02 bara može izazvati povrede ljudi, dok natpritisak od 0,1 bar može prouzrokovati i smrtne povrede. Takođe je primećeno da vertikalno oslobađanje, koje se suočava sa preprekama poput plafona, vozila i zidova, rezultira znatno većim oblacima pare u odnosu na horizontalno

Pored toga, simulacije su ukazale da eksploziivne sile izazvane velikim oblakom gasa mogu biti veoma opasne. Eksplozija oblaka od 200 m³ može generisati natpritiske veće od 0,3 bara u garaži površine oko 900 m², što može ozbiljno ugroziti stabilnost građevine i sigurnost prisutnih osoba [5].

Tokom simulacije, veliki oblak je razblažen na koncentraciju ispod donje granice zapaljivosti unutar 60 sekundi. Detektori gasa postavljeni na visini od 15 cm iznad poda mogli su da otkriju prisustvo zapaljivog oblaka pare za manje od minut nakon oslobađanja, što opravdava pretpostavku da zapaljivi oblak može biti prisutan u garaži maksimalno 2 minuta [5].

Važan zaključak iz simulacija odnosi se na efikasnost ventilacionih sistema u razblaživanju opasnih oblaka. ventilacioni protok od 200.000 m³/h značajno smanjuje koncentraciju gasa i brzo razblažuje oblak pare unutar prvih 20 sekundi rada. Nasuprot tome, manji protok od 50.000 m³/h nije bio dovoljan da spreči širenje oblaka po celoj podnoj površini garaže, što povećava rizik od eksplozije. Prema tome, preporučuje se minimalni protok ventilacije od približno 0,06 m³/s po kvadratnom metru površine garaže kako bi se efikasno razblažili veliki oblaci nastali usled brzog ispuštanja TNG-a [5]. Smanjenje ovog protoka produžava prisustvo opasnih oblaka i rizik od nesreća.

U zaključku, istraživanja [5, 8] potvrđuju da su efikasni ventilacioni sistemi i pravilno postavljeni detektori ključni za minimizaciju rizika od eksplozije u garažama sa TNG vozilima. Kontinuirani rad ventilacije i pravovremena detekcija mogu značajno smanjiti opasnost od požara i povreda, čime se obezbeđuje veća bezbednost kako ljudi tako i objekata.

6. UTICAJ NA LJUDE

Kontakt TNG-a sa kožom ili očima može izazvati promrzline i iritacije, posebno pri curenju gasa tokom dopune goriva ili zbog neispravne konverzije vozila. Zbog visokog pritiska i niske temperature skladištenja, neophodno je koristiti zaštitne mere pri radu sa vozilima sa TNG sistemima. Edukacija korisnika o opasnostima i pravilnoj upotrebi ključna je za prevenciju povreda. Pravilna dijagnostika i pravovremena prva pomoć mogu značajno poboljšati ishod povreda izazvanih kontaktom sa TNG-om.

Klinička slika povreda uzrokovanih direktnim kontaktom sa tečnim naftnim gasom može se kretati od površinskih oštećenja do nekroze i amputacija. Vreme izloženosti TNG-u je ključni faktor koji određuje raznolikost kliničkih manifestacija [10].

Početna procena hladne opekotine na osnovu izgleda gornjih slojeva kože nije uvek pouzdana, jer se dublje, degenerativne nekroze tkiva mogu pojaviti kasnije, zato pacijente sa ovim povredama treba pažljivo nadzirati od prvog dana. Najveći problem kod ovakvih trauma je ograničeno kliničko iskustvo medicinskog osoblja u vezi sa dijagnostikom i tretmanom. Da bi se smanjila stopa morbiditeta kod promrzlina izazvanih TNG-om ili sličnim uzrocima, važno je da pacijenti blagovremeno potraže medicinsku pomoć, a da zdravstveni radnici budu svesni različitih stepena ovih povreda. [10].

U slučaju promrzline preporučuju se osnovne mere prve pomoći, koje uključuju pranje zahvaćenog mesta sapunom i vodom i ispiranje kože u vodi blizu telesne temperature, do 40°C, u trajanju od 15 do 30 minuta. Ova metoda pomaže u minimalizaciji gubitka tkiva i bilo kakve hemijske iritacije. Umotavanje povređenog mesta toplim oblogom može doprineti postepenom zagrevanju povređene oblasti i smanjenju rizika od dodatnog oštećenja. [11].

7. LITERATURA

[1] S. Rakić, "Analiza primene tečnog naftnog gasa kao pogonskog energenta motora SUS," Vojnotehnički glasnik, vol. 56, br. 1, str. 90–107, 2008.

[2] J. Chojnowski, "Safety in the use of car gas fuel installations," Combustion Engines, vol. 61, 2022.

[3] https://www.cartoq.com/understanding-and-installing-lpg-kits-for/. (pristupljeno u decembru 2024.)

[4] K. Lejda i E. Zielinska, "Gas installations requirements for cars and automobile repair shops offering LPG services," Teka Komisji Motoryzacji i Energetyki Rolnictwa, vol. 15, br. 1, str. 37–42, 2015.

[5] F. Van den Schoor, P. Middha i E. Van den Bulck, "Risk analysis of LPG (liquefied petroleum gas) vehicles in enclosed car parks," Fire Safety Journal, vol. 57, str. 58–68, 2013.

[6] G. E. Apostolakis, "How useful is quantitative risk assessment?," Risk Analysis: An International Journal, vol. 24, br. 3, str. 515–520, 2004.

[7] G. Tepić, Razvoj metodološkog koncepta za upravljanje rizikom u sistemu opasnih materija, Novi Sad: Fakultet tehničkih nauka, 2019.

[8] D. Brzezińska, M. Dziubiński i A. S. Markowski, "Analyses of LPG dispersion during its accidental release in enclosed car parks," Ecological Chemistry and Engineering S, vol. 24, br. 2, str. 249–261, 2017.

[9] J. Stawczyk, "Experimental evaluation of LPG tank explosion hazards," Journal of Hazardous Materials, vol. 96, br. 2–3, str. 189–200, 2003.

[10] E. Kapı i dr., "An unusual etiology in cold injury: liquefied petroleum gas," Ulusal Travma ve Acil Cerrahi Dergisi, vol. 23, br. 3, 2017.

[11] B. Scarr i dr., "Liquefied petroleum gas cold burn sustained while refueling a car," Emergency Medicine Australasia, vol. 22, br. 1, str. 82–84, 2010.

Kratka biografija:

Nina Ivanović rođena je u Subotici 2000. god. Osnovne studije Inženjerstva zaštite na radu završila je na Fakultetu tehničkih nauka u Novom Sadu 2023. godine, a iste godine upisuje i master studije, smer Bezbednost i zaštita na radu sa transportnim i građevinskim mašinama i motornim vozilima.
kontakt: nina.ivanovic.ace@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 662.7:662.61
DOI: https://doi.org/10.24867/33AM04Milosevic

RAZLIKE IZMEĐU DVOCEVNOG I ČETVOROCEVNOG SISTEMA GREJANJA I HLAĐENJA POMOĆU VENTILATOR-KONVEKTORA

DIFFERENCE BETWEEN A TWO-PIPE AND A FOUR-PIPE HEATING AND COOLING SYSTEMS USING A FAN-COIL UNIT

Aleksa Milošević, *Fakultet tehničkih nauka, Novi Sad*

Mašinstvo – ENERGETIKA I PROCESNA TEHNIKA

Kratak sadržaj – *U okviru ovog rada dat je kratak opis i razlika između dvocevnih i četvorocevnih sistema grejanja i hlađenja. Poseban akcenat stavljen je na ventilator-konvektore pomoću kojih se greju/hlade sistemi, izvršena je njihova podela i objašnjen princip rada.*

Ključne reči: *ventilator-konvektori, grejanje, hlađenje*

Abstract – *In this paper, a brief description and a difference between two-pipe and four-pipe heating and cooling systems is given. The main topic was placed on fan-coils, which are used to heat/cool systems, their division was made and the basic principles of their operation was explained.*

Keywords: *fan-coil units, heating, cooling*

1. UVOD

Upotreba sistema za klimatizaciju je postala stalan i neizostavan deo naše svakodnevnice, polazeći od domaćinstava, zagrevanja zimi, hlađenja leti, kao i obezbeđivanje željenog kvaliteta vazduha, pa do obezbeđivanja raznih industrijskih potreba, kao što su, na primer, prostorije sa odgovarajućom niskom ili visokom temperaturom, vlažnosti vazduha i slično. Svaki objekat zahteva sistem za klimatizaciju. Složenost sistema za klimatizaciju zavisi od složenosti samog objekta. Sistem za klimatizaciju je ozbiljan energetski potrošač, i iziskuje troškove koji nisu zanemarljivi. Iz ovih razloga potrebno je konstatno raditi na unapređivanju efikasnosti sistema za klimatizaciju i izvlačiti maksimalnu ekonomičnost i efikasnost iz njega [1]. U ovom radu opisani su dvocevni i četvorocevni sistemi grejanja i hlađenja pomoću ventilator konvektora, kao i razlika između ova dva sistema.

2. DVOCEVNI SISTEMI

Karakteristika dvocevnih sistema (*slika 1.*) jeste da tokom cele godine kroz cevnu mrežu struji hladna ili topla voda, dok se temperatura primarnog vazduha podešava prema uslovima spoljne temperature.

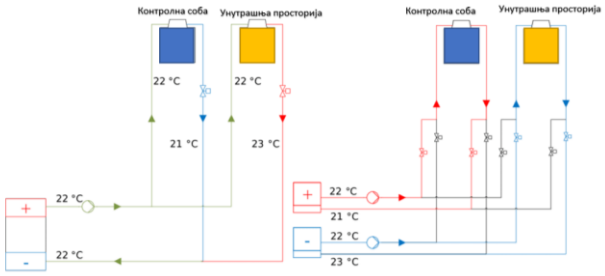
U toplijim krajevima u kojima više variraju letnja toplotna opterećenja (toplota od sunca, ljudi, uređaja...), nego zimski toplotni gubici, kao što je slučaj kod npr. savremene kancelarije sa puno opreme, uvek se koristi hladna voda u "sekundarnoj mreži". Pri najnižim spoljnim temperaturama, npr. ako zimi ipak treba zagrevanje, to se postiže grejanjem vazduha u centralnoj klima-komori (a ne vodom). U letnjem režimu i vazduh i voda oduzimaju toplotu prostoriji, jer su oba fluida na nižim temperaturama od temperature prostorije. Hladan primarni vazduh neutrališe stalne i vremenski malo promenljive dobitke toplote (transmisioni oblici), dok je kapacitet razmenjivača toplote (ispunjenog vodom) podešen prema trenutnim dobicima (sunčevo zračenje, osvetljenje, ljudi). Primarni fluid dvocevnog sistema se priprema kao u klasičnim vazdušnim sistemima, pri čemu se vodi računa da se vazduh leti hladi u hladnjaku komore sa nižom temperaturom vode, kako bi se još tu dovoljno osušio. U tom slučaju se za hlađenje sekundarnog vazduha može koristiti viša temperatura vode i tako postići hlađenje bez izdvajanja vlage. U zimskom režimu, kada je spoljna temperatura ispod temperature površine hladnjaka, hladnjak se može koristiti i za pripremu vode razmenjivača toplote indukcionog aparata. Indukcioni uređaji funkcionišu tako što se vazduh iz prostorije indukuje kroz rešetku na indukcionom uređaju, prolazi kroz izmenjivač toplote, gde se prema potrebi greje ili hladi, a potom u plenumskoj kutiji meša sa pripremljenim svežim vazduhom i zatim kroz linijske otvore koji se nalaze na dve ili na sve četiri strane uređaja ubacuje u prostoriju [2]. Takav režim rada je ekonomičan, jer je priprema vode bez potrošnje energije, a spoljni vazduh se u izvesnom stepenu zagreva, što smanjuje potrebno odavanje toplote grejača u klima komori [3].

3. ČETVOROCEVNI SISTEMI

Četvorocevni sistemi (*slika 1.*) spadaju u tzv. višecevne vazdušno vodene sisteme. Odlikuju se sa dve razvodne cevne mreže za sekundarnu vodu, pa u svakom trenutku postoji mogućnost propuštanja kroz indukcioni aparat, bilo hladne ili tople vode. Prema tome, iako nemaju formalno prebacivanje sa jednog režima na drugi, po temperaturama vazduha i vode potpuno odgovaraju dvocevnim sistemima. Indukcioni aparat četvorocevnog sistema u svakom trenutku može da greje/hladi, ili hladi i greje istovremeno. Indukcioni aparat četvorocevnog sistema ima dva priključka za dovod i odvod tople vode, i dva za odvod i dovod hladne vode. Pri tome indukcioni aparat može da poseduje jedan zajednički razmenjivač za grejanje odnosno

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Maša Bukurov Nikolić, red. prof.

hlađenje, ili, što je najčešći slučaj, dva razmenjivača – grejač i hladnjak. Ovakav sistem je svakako najpovoljniji klimatizacioni sistem sa mogućnošću trenutnog prelaska sa hlađenja na zagrevanje vazduha i obrnuto, što utiče na potrebna visoka ulaganja. Četvorocevni sistem je posebno pogodan za zgradu sa prostorijama čiji se toplotni zahtevi nesrazmerno menjaju i ne mogu da se grupišu u zone [4].



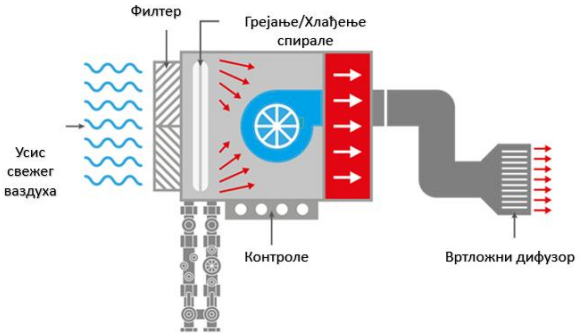
Slika 1. *Dvocevni i četvorocevni sistem grejanja i hlađenja* [5]

4. VENTILATOR-KONVEKTORI

Ventilator-konvektor („FCU") je jednostavan izmenjivač toplote koji koristi ventilator i zavojnicu za hlađenje ili zagrevanje vazduha u prostoriji bez upotrebe kanala. „FCU" je jedan od komponenta „HVAC" sistema koji se koriste u stambenim, komercijalnim i industrijskim zgradama. Ventilator-konvektor obezbeđuje brojne prednosti u odnosu na konvencionalni sistem grejanja i hlađenja uključujući: poboljšanje kvaliteta vazduha, bešuman rad, manje troškove instalacije kao i bolju energetsku efikasnost. Osigurava preciznu kontrolu temperature u prostoriji i može se koristiti za grejanje kao i za hlađenje [6].

Konstrukcija ventilator-konvektora

Struktura ventilator-konvektora (*slika 2.*) je u obliku kutije poput peći, a njegova unutrašnjost podseća na saće. Ventilator-konvektor ima niz unutrašnjih komponenti kao što su redovi spiralnih cevi, ventilator za dovod vazduha, motor, filter, kondenzat posuda za odvod, kontrolni ventil itd. Za poboljšanje kvaliteta vazduha u zatvorenom prostoru, kao i za smanjenje troškova održavanja većina ventilator-konvektora je opremljena filterom za vazduh. Rashladno sredstvo, ohlađena voda ili vruća voda cirkulišu kroz zavojnicu da ohlade ili zagreju vazduh. Neke jedinice su opremljene električnim trakama za grejanje. Ventilator izduvava klimatizovani vazduh iz sistema i nazad u unutrašnji prostor. Ventilator-konvektori dolaze u tri osnovne konfiguracije: horizontalni, vertikalni (vođeni više u zidu) i grejači jedinica nisko uzduž zida. Ventilator-konvektor nudi kompaktan opseg od prve do četvrte brzine izduvavanja ventilatora. Tipična veličina ventilator-konvektora je 500 mm do 800 mm dužine, 500 mm do 2000 mm širine i 160 mm do 400 mm dubine [7].



Slika 2. *Ventilator-konvektor [8]*

Princip rada ventilator-konvektora

Ventilator duva preko redova spiralnih cevi kroz koje prolazi rashladno sredstvo, rashlađena voda ili topla voda. U toj kombinaciji ventilator i spiralne cevi zajedno deluju kao izmenjivač toplote. Postoji prenos toplote između vazduha kome je vrela tečnost odala svoju toplotu na rashladno sredstvo ili hladnu vodu koja deluje kao hladna tečnost. Tokom hlađenja vazduha toplota se prenosi sa vazduha u rashladno sredstvo ili ohlađenu vodu. Tokom zagrevanja vazduha u prostoru topla voda koja teče unutar spiralnih cevi će delovati kao topla tečnost i prenositi toplotu na vazduh koji je u tom slučaju hladan fluid, kada recirkulacija zagreva prostor ili prostoriju. Hlađenje ili grejanje vazduha zavisi od željene temperature u sobi. Vazduh u prostoru iznova cirkuliše za postizanje željenog temperaturnog stanja prostorije. Motor instaliran u ventilator-konvektor pokreće lopatice ventilatora da se rotiraju, stvara dovoljnu zapreminu vazduha i prelazi preko zavojnice gde teče ohlađena voda, po istom principu u zimskoj sezoni topla voda teče unutar spiralnih cevi ventilator-konvektora i omogućava prenos toplote iz spiralnih cevi sa toplom vodom u vazduh koji struji preko njih pa onda u samu prostoriju, i samim tim se prostorija zagreva. U zavisnosti od zahtevane temperature prostora, da li je potrebno hlađenje ili grejanje, ohlađena ili topla voda automatski teče kroz spiralne cevi ventilator-konvektora. Termostat kontroliše temperaturu prostorije ili prostora i pokreće ventilator-konvektor za rad u režimu grejanja ili hlađenja kontrolisanjem brzine ventilatora i protoka tečnosti kroz spiralne cevi [9].

Tipovi ventilator-konvektora

Ventilator-konvektor je podeljen na osnovu instalacije i rasporeda cevovoda.

Postoje različite vrste „FCU" zasnovane na instalaciji:

1. horizontalni ventilator-konvektor;

2. Vertikalni ventilator-konvektor;

3. Podni ventilator-konvektori;

4. Ventilator-konvektori montirani na zid.

Sva četiri tipa ventilator-konvektora pružaju isti nivo performansi i isporučuju samo grejanje/hlađenje ili grejanje i hlađenje.

Na osnovu rasporeda cevovoda ventilator-konvektor može biti podeljen na dva tipa:

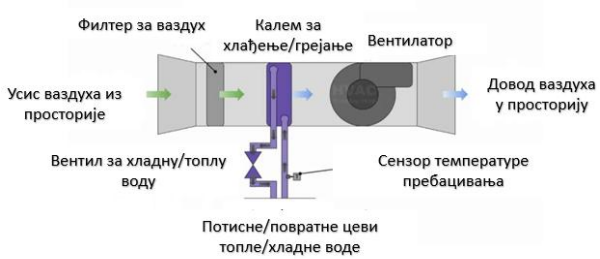
1. Dvocevni ventilator-konvektor;

2. Četvorocevni ventilator-konvektor. [7]

Dvocevni ventilator-konvektor

Dvocevni ventilator-konvektor (*slika 3.*) je jedna zatvorena petlja za napajanje i povrat sistema za distribuciju vode koji opslužuje svaku prostoriju u kojoj zgrada ima samo jednu vodenu petlju. Time je grejanje ili hlađenje dostupno u zavisnosti od sezone. Dvocevni ventilator-konvektor ima jedan kalem povezan sa dve cevi. Jedna cev je za potis vode, a druga za povrat vode. Rashlađena ili topla voda protiče kroz kalem, u zavisnosti od toga da li je potrebno hlađenje ili grejanje u zgradi. Protok kroz zavojnicu kontroliše ventil na jednoj od cevi. Ventil se otvara i zatvara u zavisnosti od potrebe za grejanjem ili hlađenjem u prostoru.

Glavna prednost upotrebe dvocevnih ventilator-konvektora u „HVAC" sistemu je što su jeftinije za instalaciju, potrebno je manje truda i materijala (cevi, fitinzi, ventili, itd.) za ugradnju dvocevnog ventilator-konvektora u poređenju sa četvorocevnim ventilator-konvektorom. [10]

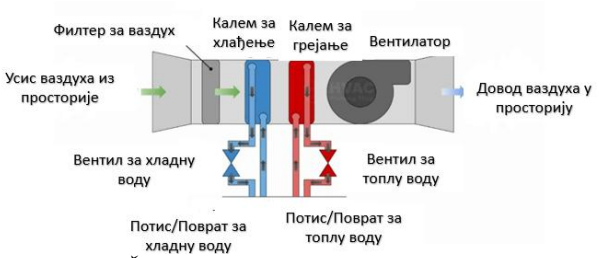


Slika 3. *Dvocevni ventilator-konvektor [10]*

Četvorocevni ventilator-konvektor

Četvorocevni ventilator-konvektor, (*slika 4.*) sastoji se od dva odvojena kalema za grejanje za hlađenje. Svaki kalem ima svoje namenske setove cevi uključujući potisne i povratne cevi i ventile. Ova vrsta ventilator-konvektora može istovremeno hladiti i grejati u zavisnosti od zahteva stanova i lokala zgrade. Ventilator-konvektor sa četiri cevi ima dva namotaja - zavojnicu za hlađenje i zavojnicu za grejanje. Svaki kalem je povezan sa dve cevi: jednom za potis vode i drugom za povrat vode. I rashladni i grejni kalemovi imaju svoje pojedinačne ventile, tako da postoje odvojeni ventili za hlađenje i grejanje. Ventil za hlađenje se otvara ako je potrebno hlađenje u prostoru. Ventil za grejanje se otvara ako je potrebno grejanje u prostoru.

Glavna prednost korišćenja četvorocevnih ventilator-konvektora u „HVAC" sistemu je to što su kalemovi za hlađenje i grejanje odvojeni jedan od drugog, što omogućava istovremeno grejanje i hlađenje različitih prostorija i lokala u celoj zgradi. [10]



Slika 4. *Četvorocevni ventilator-konvektor* [10]

5. ZAKLJUČAK

Tema ovog rada je bila sumiranje prednosti, mana i osnovnih razlika izmedju dvocevnog i četvorocevnog sistema grejanja i hladjenja pomoću ventilator konvketora. Glavna mana četvorocevnog sistema su znatno skuplji troškovi u odnosu na dvocevni, ne zbog same cene ventilator-konvektora jer ta razlika nije velika, nego zbog same količine cevi kroz koje struji medijum od toplotne pumpe pa do ventilator-konvektora. Glavna prednost četvorocevnog sistema je komfor, jer je moguće istovremeno grejanje i hlađenje različitih prostorija u objektu, dok kod dvocevnog sistema postoji samo režim hlađenja ili režim grejanja. Iz ovoga vidimo da su ova dva sistema gledano iz tehničkog apsekta potpuno ista, a da je osnovna razlika u komforu koji nudi četvorocevni sistem, i u manjoj ceni dvocevnog sistema.

6. LITERATURA

[1] Vasić, Aleksandar, Seminarski rad GREJANJE I VENTILACIJA na temu SISTEMI ZA KLIMATIZACIJU I VENTILACIJU, Kosovska Mitrovica, 2017

[2]https://www.gradjevinarstvo.rs/tekstovi/9875/820/industcioni-uredjaji-stede-energiju-i-obezbedjuju-komfor

[3] Тодоровић, Бранислав, Климатизација, Савез машинских и електротехничких инжењера Србије, Београд, 1998. [5] Catherine Rivet, Hyewong Lee, Alison Hirsch, Sharon Hamilton, Hang Lu, Microfluidics for medical diagnostics and biosensors, Elsiver Ltd., 2010.

[4] Kreider, J. A. Rabl – Hill, Heating and Cooling of Buildings, McGraw ,New York, 1994.

[5] Maccarini, A. (2017). A two-pipe system for simultaneous heating and cooling of office buildings: Transferringheat among building rooms through a room-temperature water loop. Aalborg Universitetsforlag. Ph.d.-serien forDet Ingeniør- og Naturvidenskabelige Fakultet, Aalborg Universitet [8] Gaozhe Cai, Li Xue, Huilin Zhang, Jianhan Lin, A Review on Micromixers, Micromachines, 8, 2021.

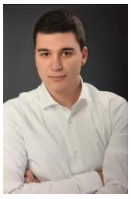
[6] Saravanan D, A Study on Fan – Coil Unit, its types and maintenance in HVAC System, Valivalam Desikar Polytechnic Collage Nagapattinam, International Research Journal of Engineering and Technology (IRJET), 2019.

[7] Price, Mike, Fan – Coil Units, Charted Institution of Building Services Engineers, 2008. [11] Clement Kleinstreuer, Microfluidics and Nanofluidics: Theory and Selected Applications, Wiley, 2013.

[8] https://www.cranefs.com/systems/fan-coil-units/

[9] Yunus, A. Cengel, Heat and mass transfer, 3rd edition Tata Mcgraw – hill publishing company limited, New Delhi.
[10] https://hvactrainingshop.com/how-a-fan-coil-unit-works/

Kratka biografija:



Aleksa Milošević rođen je u Novom Sadu 2000. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Energetike i procesne tehnike – Primer rekonstrukcije vrelovoda odbranio je 2023.god.
kontakt: acimilosevic@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 697.1
DOI: https://doi.org/10.24867/33AM05Simovic

PRAKTIČNA PRIMENA "DUAL DUCT" SISTEMA KLIMA KOMORE SA REKUPERACIJOM VAZDUHA

PRACTICAL APPLICATION OF THE „DUAL DUCT" AIR HANDLING UNIT SYSTEM WITH AIR HEAT RECOVERY

Miloš Simović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – MAŠINSTVO

Kratak sadržaj – *U radu je prikazana analiza sistema klimatizacije, grejanja i hlađenja administrativnog prostora u sklopu poslovnog objekta. Sistem je baziran na primeni toplotne pumpe i klima komore sa dvokanalnom (Dual Duct) distribucijom vazduha. Urađeni su proračuni toplotnih gubitaka i dobitaka i određene potrebne količine svežeg i recirkulisanog vazduha. Analizirani su uticaji različitih parametara (odnos svežeg i recirkulisanog vazduha, odnos vazduha iz toplog i hladnog vazdušnog kanala) na rad sistema, kao i energetska efikasnost i komfor korisnika.*

Ključne reči: *grejanje, klimatizacija, administrativni prostor, proračun.*

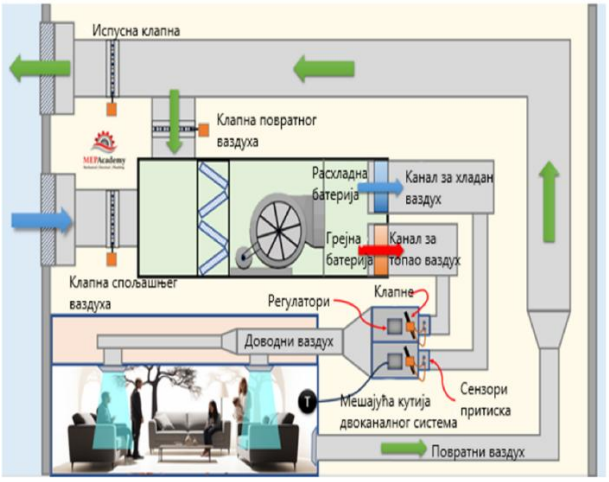
Abstract – *The paper presents an analysis of the air – conditioning, heating and cooling system of an administrative space within a comercial building. The system is based on the application of a heat pump and an air – handling unit with dual – duct air distribution. Design calculations of heat losses and gains were carried out, and the required amounts of fresh and recirculated air were determined. The impacts of various parameters (the ratio of fresh to recirculated air and the ratio of warm to cold air streams) on system operation, as well as energy efficiency and user comfort, were analyzed.*

Key words: *heating, air conditioning, administrative space, design calculation..*

1. UVOD

Savremeni sistemi klimatizacije, grejanja i hlađenja moraju istovremeno da obezbede funkionalnost, energetsku efikasnost i visok nivo komfora. Jedno od rešenja koja omogućavaju takvu ravnotežu je primena toplotne pumpe u kombinaciji sa klima komorom i Dual Duct distribucijom vazduha. Dual Duct koncept obezbeđuje veću fleksibilnost u regulisanju parametara unutrašnjeg vazduha, što je posebno značajno u prostorijama različitih namena. Cilj ovog rada je da se prikaže praktična primena takvog sistema u konkretnom objektu. Rad obuhvata analizu proračunskih parametara, kao i ocenu energetske efikasnosti i nivoa komfora koji ovaj sistem obezbeđuje.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Anđelković, vanred. prof.



Slika 1. Šema "Dual Duct" sistema [1]

2. TEHNIČKI OPIS SISTEMA

Sistem klimatizacije, grejanja i hlađenja administrativnog prostora zasnovan je na primeni 4-cevne toplotne pumpe kao primarnog izvora energije i klima komore sa dvokanalnim (Dual Duct) razvođenjem vazduha. Na strani rashladnog i grejnog medijuma, za transport toplotne energije koriste se cirkulacione pumpe, dok se distribucija u prostoru realizuje preko vrtložnih difuzora. Klima komora je projektovana kao centralni element sistema, sa modulom za mešanje svežeg i recirkulisanog vazduha, kao i sa toplim i hladnim kanalom. U letnjem režimu, komora obezbeđuje hlađenje i odvlaživanje vazduha, dok je u zimskom režimu zadužena za grejanje i ovlaživanje. Da bi se obezbedilo fino podešavanje temperature, svaka prostorija ima lokalnu kontrolu i merenja unutrašnje temperature. VAV kutija dalje otvara ili zatvara klapne na toplom ili hladnom vazduhu i tako reguliše temperaturu smeše.

Tabela 1. *Bilans instalisanih toplotnih kapaciteta*

Izvor	Grejanje(-5°C) (kW)	Hlađenje (kW)
4-cevna toplotna pumpa vazduh/voda	73,6	102,8

3. NUMERIČKA DOKUMENTACIJA

Proračunom toplotnih gubitaka dobijena je vrednost od oko 30kW, dok su toplotni dobici u letnjem periodu iznosili oko 50kW. Ove vrednosti su bile osnova za dimenzionisanje opreme, posebno toplotne pumpe i klima komore. Na osnovu toplotnih opterećenja određene su i potrebne količine svežeg i recirkulisanog vazduha. Usvojeni parametri omogućavaju stabilan rad sistema i u uslovima promenljivog opterećenja.

Tabela 2. Količine vazduha po prostorijama

Količine vazduha po prostorijama							
Prostorija		Količina vazduha					
		Свеж въздух					
			Proračun		Izabrano		
		Pripremljen vazduh	Količina vazduha	Procenat	Procenat	Količina vazduha	Povratni vazduh
		m³/h	m³/h	%	%	m³/h	m³/h
1	611 KANCELARIJA - 95	800	240	30	40	324	740
2	610 KANCELARIJA – 96	5440	1950	35	40	2192	5000
3	612 SALA ZA SASTANKE - 97	920	300	33	40	364	840
4	491 STEPENIŠTE – 98						
5	HODNIK – 99						
6	445 SVLAČIONICA – 100	540	240	44	40	220	420*
7	445e TUŠ – 101						60*
8	445e TUŠ – 102						45*
9	445e TUŠ – 103						45*
10	445a TOALET – 104						580*
11	443 SASTANCI – 105	580	270	46	40	236	580
12	613 SALA ZA SASTANKE – 106	420	270	64	40	168	420
13	615 TIM ZA PRODAJU – 107	520	270	53	40	204	470
14	613 SALA ZA SASTANKE – 108	170	60	35	40	68	150
15	615 SALA ZA SASTANKE – 109	170	60	35	40	68	150
16	616 IT SKLADIŠTE – 110						60
17	617 SKLADIŠTE KASE – 111						
18	618 SALA ZA OPUŠTANJE – 112	420	210	57	40	148	420
		9980	**3810**			**3992**	**9980**

Proračunima cevne i kanalske mreže dobili smo potrebne dimenzije cevovoda koje prenose toplotnu energije od toplotne pumpe i toplotne podstanice do klima komore i dimenzije kanala kojim se distribuira vazduh iz klima komore do prostorija.

Tabela 3. *Proračun cevne mreže od toplotne pumpe do klima komore u režimu grejanja*

PRORAČUN CEVNE MREŽE - TOPLOTNA PUMPA - KLIMA KOMORA GREJANJE S-12

Temperatura u polaznom vodu:	t_1=	55.0	[°C]
Temperatura u povratnom vodu:	t_3=	49.0	[°C]
Specifična gustina vode	cp_3=	4.181	[kJ/kgK]
Gustina vode:	r_3=	968.2	[kg/m³]
Kinematska viskoznost:	n=	3.65E-07	[m²/s]
Hrapavost cevi:	e=	0.0450	[mm]

Iz plana mreže

Deonica	Količina toplote Q	Maseni protok G	Proračunski prečnik cevi	Dužina deonice l	Prečnik deonice D	Unut. Prečnik deonice du	Stvarna brzina strujanja v	Preporučena brzina strujanja	Rejnoldsov broj Re	Jedinični pad pritiska R	l × R	Koef. lok. otpora Σξ	Lokalni otpori Z	Pad pritiska
	W	kg/h	mm	m	mm	m	m/s			Pa/m	Pa		Pa	Pa

Strujni krug najudaljenijeg grejnog tela

Cirkulacioni krug najudaljenijeg grejnog tela														
1	83000	12301.14	53.8692	13	DN65	0.0703	0.88	1.50	169639	109.219	1419.8	20	7510.87	8930.7
														8930.71
1	**Pad pritiska U VENTILIMA**													6000.0
2	**Pad pritiska na izmenjivaču KK**													30000.0
														36000.00
											Под притиска:		[Pa]	**64930.71**

Tabela 4. *Proračun vazdušnog kanala do najudaljenijeg difuzora*

PRORAČUN VAZDUŠNOG KANALA DO NAJUDALJENIJEG DIFUZORA															
Deonica		Količina vazduha		Kanal						Gubitak pritiska usled trenja		Gubitak pritiska usled mesnih otpora			RxL+Z
	L	V	V	a	b	D	Dh	F	v	R	RxL	Σξ		Z	
	m	m3/h	m3/s	mm	mm	mm	mm	m2	m/s	Pa/m	Pa	opis	broj	Pa	Pa
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
KANALI ZA UBACIVANJE VAZDUHA															
1	14.00	9980	2.77	1250	600	-	811	0.750	3.70	0.16	2.21	-		0.0	2.21
2	3.10	9640	2.68	1250	600	-	811	0.750	3.57	0.15	0.46	-		0.0	0.46
3	0.64	9120	2.53	1250	600	-	811	0.750	3.38	0.13	0.09	-		0.0	0.09
4	1.80	8120	2.26	1250	600	-	811	0.750	3.01	0.11	0.20	-		0.0	0.20
5	13.90	1460	0.41	350	300	-	323	0.105	3.86	0.51	7.2	račva ubod	0.60	5.5	12.6
6	6.00	920	0.26	250	250		250	0.063	4.09	0.78	4.7	koleno+redukcija	0.90	9.2	13.8
7	1.00	460	0.13	-	-	250	-	0.049	2.60	0.35	0.3	račva	1.20	5.0	5.3
															34.7
												Δp=1.1x1RxL+Z=			38.2
												PV ventil+rešetka			5.0
												2 PP klapne			10.0
												Σdp=			53.2

Proračunom cevne mreže smo takođe dobili i padove pritisaka u režimima grejanja i hlađenja u cevovodima između toplotne pumpe i klima komore, koji za režim grejanja sa uzetom rezervom iznose 50kPa, dok za režim hlađenja iznose 60kPa.

Tabela 5. *Proračun cirkulacione pumpe u režimu grejanja*

Instalisana snaga:	Q_{v1}=	83.000,0	[W]
Temperatura u polaznom vodu:	t_f=	55,0	[°C]
Temperatura u povratnom vodu:	t_r=	49,0	[°C]
Srednja temperatura:	t_{sr}=	52,0	[°C]
Specifična toplota vode:	cp_3=	4,177	[kJ/kgK]
Gustina vode:	r_3=	999,9	[kg/m³]

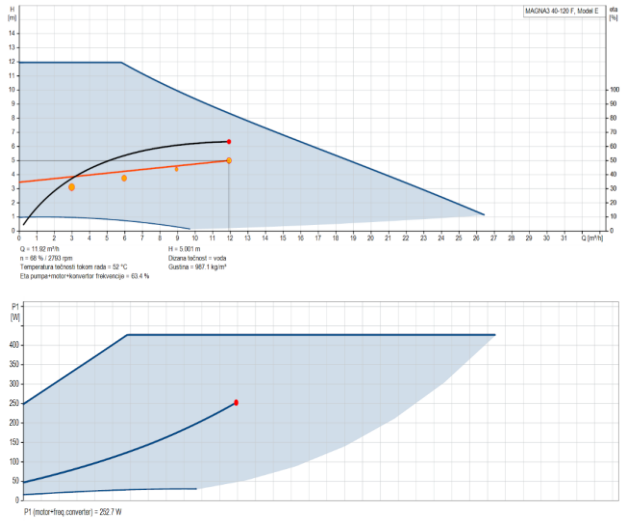
$$G_{v1} = \frac{3600 \cdot Q_{v1}}{c_{p3} \cdot 1000 \cdot \rho_3 \cdot (t_f - t_r)} = 11{,}92\ m/h^3 \qquad (1)$$

Za režim grejanja odredili smo protok vode od $G_{v1} = 11{,}92\ m/h^3$

Na osnovu protoka vode i izračunatog pada pritiska dimenzionišemo cirkulacionu pumpu za režim grejanja i usvajamo pumpu MAGNA3 40 – 120 F sa sledećim karakteristikama:

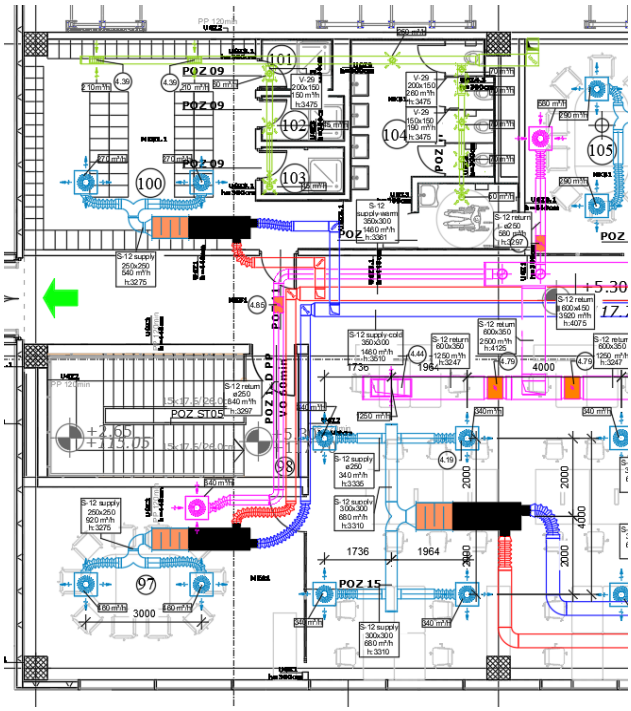
- Protok: 11,92 m³/h
- Napor: 50 kPa
- Frekvencija: 50 Hz
- Napon: 230 V

Dijagram usvojene cirkulacione pumpe [2] je dat na slici ispod.

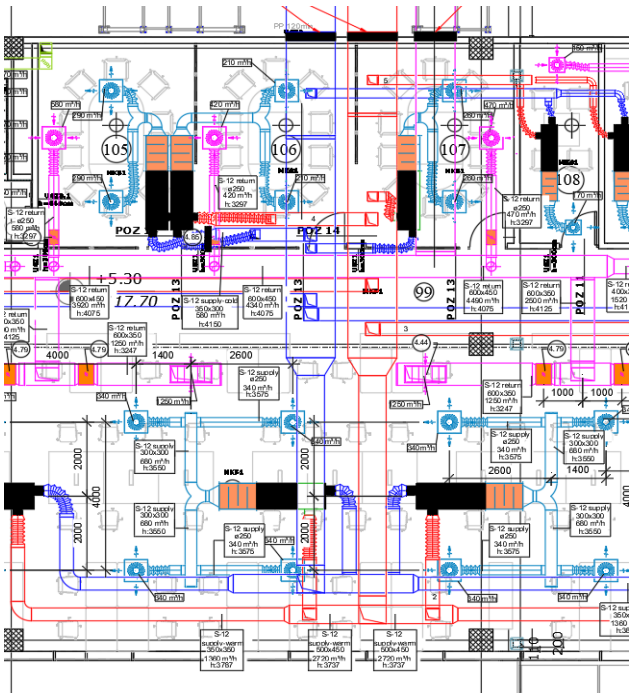


Istim postupkom smo dimenzionisali i cirkulacionu pumpu za režim hlađenja i usvojili pumpu MAGNA3 50 – 120 F.

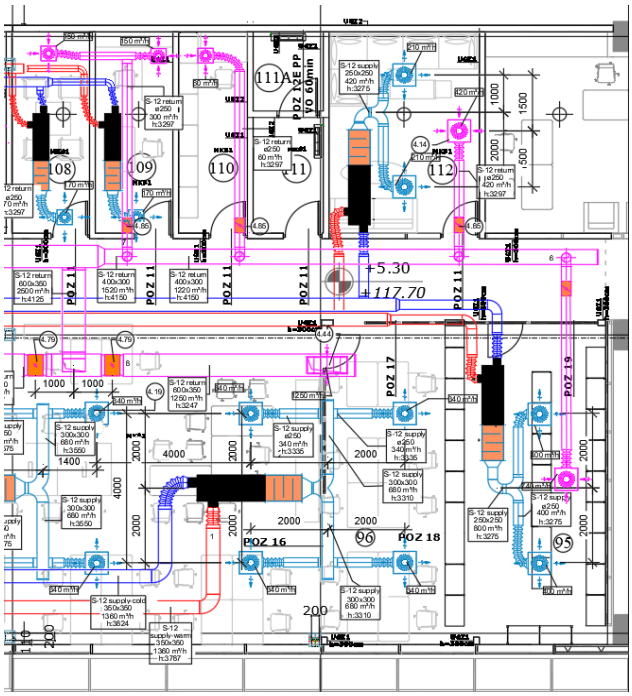
4. GRAFIČKA DOKUMENTACIJA



Slika 2. Administrativni deo sa projektovanim KGH sistemom – 1. deo



Slika 3. *Administrativni deo sa projektovanim KGH sistemom – 2. deo*



Slika 4. Administrativni deo sa projektovanim KGH sistemom – 3. deo

5. ANALIZA ODNOSA SVEŽEG I RECIRKULISANOG VAZDUHA

Analizirani su sledeći scenariji:

- Scenario A: 30% OA (spoljašnji vazduh) / 70% PA (recirkulisani vazduh)
- Scenario B: 40% OA / 60% PA (projekat)
- Scenario C: 60% OA / 40% PA.

Temperatura mešavine:

$$T_{mix} = \frac{m_{OA} \cdot T_{OA} + m_{RA} \cdot T_{RA}}{m_{OA} + m_{RA}} \quad (2)$$

- Scenario A:

$$T_{mix} = \frac{0{,}3 \cdot 34 + 0{,}7 \cdot 26}{0{,}3 + 0{,}7} = 28{,}4°C$$

- Scenario B:

$$T_{mix} = \frac{0{,}4 \cdot 34 + 0{,}6 \cdot 26}{0{,}4 + 0{,}6} = 29{,}2°C$$

- Scenario C:

$$T_{mix} = \frac{0{,}6 \cdot 34 + 0{,}4 \cdot 26}{0{,}6 + 0{,}4} = 30{,}8°C$$

Opterećenje hladnjaka:

Rashladno opterećenje proporcionalno raste sa povećanjem T_{mix}. U odnosu na scenario B (projektni), scenario A smanjuje opterećenje za oko 7.5%**,** dok scenario C povećava opterećenje za oko 15**%.**

Koncentracija CO_2**:**

$$C_{zone} = C_{out} + \frac{G}{Q_{OA}} \quad (3)$$

Pretpostavljeno: spoljašnji CO_2 = 420 ppm, broj ljudi = 50, emisija G = 18 l/h po osobi → ukupno G = 900 l/h.

- Scenario A:

$$C_{zone} = 420 + \frac{900}{2.994.000} \cdot 10^6 \approx 720\ ppm$$

- Scenario B:

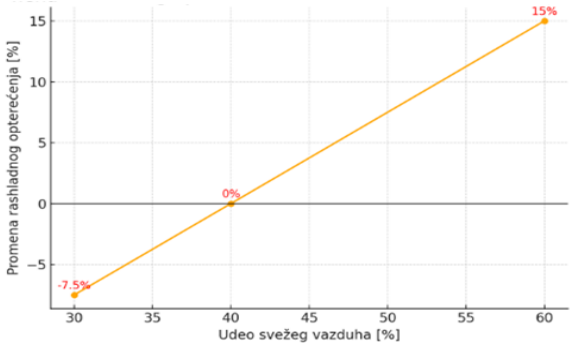
$$C_{zone} = 420 + \frac{900}{3.992.000} \cdot 10^6 \approx 645\ ppm$$

- Scenario C:

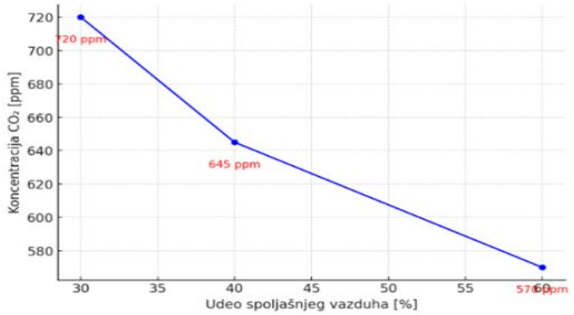
$$C_{zone} = 420 + \frac{900}{5.988.000} \cdot 10^6 \approx 570\ ppm$$

Rezultati pokazuju tipičan kompromis između energetske efikasnosti i kvaliteta unutrašnjeg vazduha. Kod odnosa 30% OA energetsko opterećenje je značajno smanjeno, ali kvalitet vazduha nije na zavidnom nivou. Kod 60% OA kvalitet vazduha je odličan (ispod 600 ppm), ali opterećenje rashladnog sistema raste za oko 15%.
Optimalan odnos za posmatrani objekat je oko 40% OA, čime se postiže balans između energetske efikasnosti i kvaliteta unutrašnjeg vazduha.

Dijagram relativne promene rashladnog opterećenja u zavisnosti od udela svežeg vazduha



Dijagram koncentracije CO_2 u zavisnosti od udela svežeg vazduha



6. ZAKLJUČAK

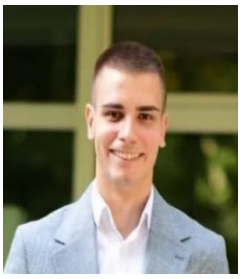
Na osnovu izvršene analize može se zaključiti da predloženo rešenje predstavlja pouzdan, energetski efikasan i komforan sistem klimatizacije, koji odgovara savremenim standardima i propisima. Rad pokazuje da je integracija toplotne pumpe sa dvokanalnom klima komorom dobro rešenje za objekte sa promenljivim opterećenjima, a dobijeni rezultati mogu poslužiti kao osnova za dalje unapređenje i primenu sličnih sistema u praksi.

7. LITERATURA

[1] MEPAcademy – **dostupno na:** https://mepacademy.com/dual-duct-system/

[2] Grundfos.com – **dostupno na:** https://www.grundfos.com/rs

Kratka biografija:



Miloš Simović, rođen je u Subotici 2001. godine. Završio je gimnaziju u Bačkoj Topoli, nakon čega 2019. god. upisuje Fakultet tehničkih nauka u oblasti Mašinstvo – Energetika i procesna tehnika. Bachelor rad odbranio je 2023. god.

Kontakt:
milossimovic2001@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

УДК: 629
DOI: https://doi.org/10.24867/33AM06Vujkovic

УТИЦАЈ СИСТЕМА ЗА КЛИМАТИЗАЦИЈУ НА ПОТРОШЊУ ГОРИВА У ПУТНИЧКОМ АУТОМОБИЛУ

THE IMPACT OF THE AIR CONDITIONING SYSTEM ON FUEL CONSUMPTION IN A PASSENGER CAR

Милош Вујковић, Драган Ружић, *Факултет техничких наука, Нови Сад*

Област – МАШИНСТВО

Кратак садржај: Потрошња енергије/горива је област која се унапређује од како су се појавила возила, било да је реч о возилима са мотором СУС, хибридним или електричним возилима. У овом раду биће приказан утицај рада система за климатизацију на потрошњу горива и поређење потрошње горива у два радна режима на возилу које је у експлоатацији дужи низ година.

Кључне речи: клима-уређај, потрошња горива, микроклима

Abstract: Energy/fuel consumption is an area that has been continuously improved since the advent of vehicles, whether they are powered by internal combustion engines, hybrids, or electric drivetrains. This paper will present the impact of air conditioning system operation on fuel consumption, as well as a comparison of fuel consumption in two operating modes on a vehicle that has been in use for an extended period of time.

Key words: Air-conditioning system, fuel consumption, microclimate

1. УВОД

Клима-уређај је део система ВГХ (вентилација, грејање, хлађење), који првенствено служи да нормализује микроклиматске услове у летњим данима, тј. да расхлади кабину возила. Од како је почео да се уграђује у возила, клима-уређај и генерално систем за вентилацију, грејање и хлађење (ВГХ) се стално унапређује у циљу побољшања микроклиматских услова у кабини возила и смањењу потрошње енергије на рад система. Познато је да је клима-уређај један од највећих екстерних потрошача енергије (горива) у аутомобилу. Процењује се да је потрошња горива у случају када је клима-уређај укључен, већа за око 20% него у случају када је искључен.

У наставку ће бити приказана стварна разлика и који радни режим је повољнији са аспекта потрошње горива.

2. ТЕОРИЈСКЕ ОСНОВЕ

Отпори кретања играју кључну улогу у уздужној динамици возила, јер возило може да се креће само ако је сила на точковима већа од укупног отпора. Постоје различити типови отпора (котрљања, успона, ваздуха, инерције, отпор прикључног возила), а који ће се узимати у обзир зависи од услова вожње. У примеру из овог рада, анализира се кретање возила по равном асфалту, константном брзином и без прикључног возила, па се не разматрају отпори успона, инерције и прикључног возила [1].

Потрошња горива зависи од снаге потребне да возило савлада отпоре кретања при одређеној брзини. Та снага се добија из енергије горива, која се у мотору претвара у механичку енергију. Потрошња зависи од укупне енергије потребне за кретање, али и од бројних конструктивних, експлоатационих фактора и додатних потрошача (попут клима-уређаја). Због свих тих утицаја, потрошњу је тешко прецизно одредити и добити исте вредности при поновљеним мерењима. [1, 5].

Микроклима у унутрашњости возила значајно утиче на удобност, радну способност и здравље путника, посебно возача. Неповољни услови могу смањити концентрацију и изазвати ефекте сличне умору или употреби алкохола. Системи за вентилацију, грејање и хлађење (ВГХ) троше значајну количину енергије, што посебно утиче на домет електричних возила, јер се енергија црпи директно из батерије. Због тога је важно стално унапређивати ове системе. [2, 3, 4].

Електронско праћање рада система на возилу: Аутодијагностички систем који се користи у ове сврхе је *OBD – On Board Diagnostics.* Од 90-их година прошлог века то је *OBD-II,* друга, напреднија верзија система за праћење рада, проверу статуса грешки и њихово отклањање. За комуникацију са електронским системом возила користе се спољне јединице (скенери, дијагностички системи). Функционалност дијагностичког уређаја зависи од прилагођености марки и типу возила, расположивости одговарајућих протокола као и од ажурираности софтвера и базе

НАПОМЕНА: Овај рад је проистекао из мастер рада чији ментор је био др Драган Ружић, редовни професор.

података. Пораст броја електричних управљачких јединица створио је потребу за мрежним системом у моторним возилима који ће имати довољно велики капацитет и брзину, способност рада у реалном времену, као и високу поузданост. *CAN – Controler Area Network* је серијски комуникациони протокол који је пројектован за примену у аутомобилској индустрији, и данас је доминантан протокол у аутомобилским мрежним системима [2, 6, 7, 8].

3. ПОСТУПАК ОДРЕЂИВАЊА И РЕЗУЛТАТИ ПОТРОШЊЕ ГОРИВА

Експеримент је вршен у околини Новог Сада, тачније на Ирмовачком путу (Футог-Руменка), на две деонице, дана 03.07.2025., са почетком у 10:21 *h*. Прва деоница је са неравним асфалтом, док је друга раван асфалт. Мерење је вршено у оба смера како би се минимизирао утицај ветра и нагиба пута уколико постоји. Возило *Peugeot 307 1.6HDi 80kW* се креће константном брзином од 70 *km/h* у 4. и 5. степену преноса мењача, са укљученим и искљученим клима-уређајем.

Опрема која се користила за мерење и бележење параметара од интереса је *Bosch KTS 590* аутодијагностички уређај који је био повезан преко *USB* кабла на лаптоп рачунар. Параметри од интереса су: број обртаја мотора, брзина кретања возила, количина убризганог горива, притисак флуида у клима-уређају, док су за одређивање потрошње горива коришћене следеће формуле:

- Часовна потрошња горива (1):

$$g\left[\frac{kg}{h}\right] = \frac{n\left[\frac{obrt}{min}\right]\cdot m_g[mg]\cdot 3600}{30\cdot 10^6} \qquad (1)$$

где је

$n\left[\frac{obrt}{min}\right]$ – број обртаја мотора,

$m_g[mg] = V_g[mm^3]\cdot 10^{-9}\cdot \rho\left[\frac{kg}{m^3}\right]\cdot 10^6$ – маса убризганог горива.

- Тренутна потрошња горива (2):

$$g\left[\frac{L}{100km}\right] = 100\cdot\frac{g\left[\frac{L}{h}\right]}{v[km/h]} \qquad (2)$$

$$g\left[\frac{L}{h}\right] = \frac{g\left[\frac{kg}{h}\right]}{\rho\left[\frac{kg}{m^3}\right]}\cdot 1000$$

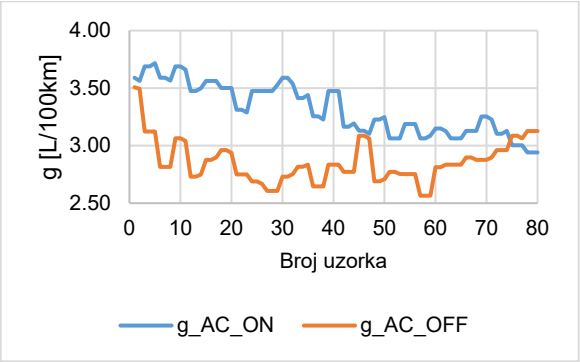
где је

$\rho = 820\ \left[\frac{kg}{m^3}\right]$ – густина дизел горива

Резултати мерења у 4. степену преноса су приказани у табели 1 и на слици 1:

Табела 1. *Потрошња горива у 4. степену преноса*

	g_AC_ON [L/100kom]	g_AC_OFF [L/100kom]	Δg [L/100kom]
MIN	2,941	2,565	0,039
MAX	3,717	3,508	0,922
AVG	3,329	3,036	0,368

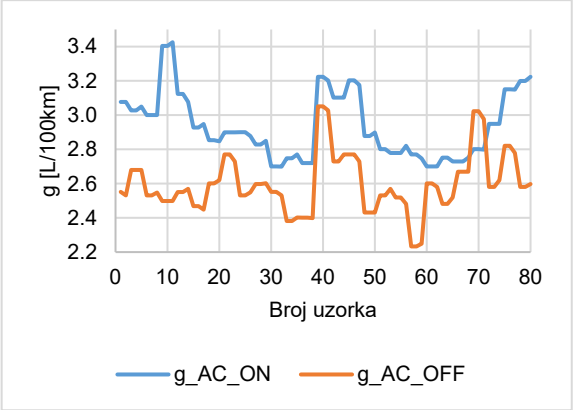


Слика 1. *Дијаграм тренутне потрошње горива у 4. степену преноса*

Потрошња горива у 5. степену преноса, са укљученим и искљученим клима-уређајем приказана је у табели 2 и на слици 2.

Табела 2. *Потрошња горива у 5. степену преноса*

	g_AC_ON [L/100km]	g_AC_OFF [L/100km]	Δg [L/100km]
MIN	2,699	2,233	0,059
MAX	3,425	3,051	0,927
AVG	3,062	2,642	0,493

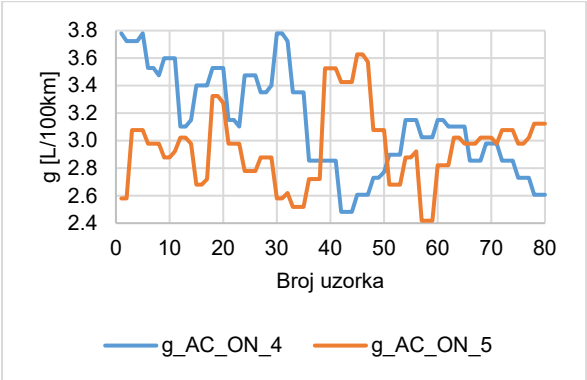


Слика 2. *Дијаграм тренутне потрошње горива у 5. степену преноса*

Следећа табела (табела 3) и слика 3 показују разлику потрошње горива између 4. и 5. степена преноса при истој брзини кретања са укљученим клима-уређајем.

Табела 3. *Потрошња горива у 4. и 5. степену преноса*

	g_4_AC_ON [L/100km]	g_5_AC_ON [L/100km]	Δg_AC_ON [L/100km]
MIN	2,941	2,699	0,063
MAX	3,717	3,425	0,890
AVG	3,329	3,062	0,476



Слика 3. *Дијаграм тренутне потрошње горива у 4. и 5. степену преноса*

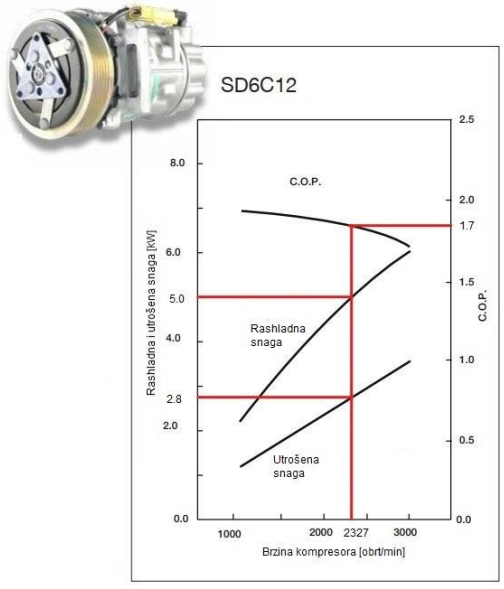
Обзиром да се преносни однос каишног преносника између коленастог вратила мотора СУС и вратила компресора клима-уређаја може одредити простим мерењем каишника, може се, на основи броја обртаја коленастог вратила и преносног односа, одредити број обртаја компресора, а самим тим и спољашња карактеристика компресора на датом режиму, слика 4 и 5.

Пречник каишника коленастог вратила је 158 *mm,* док је пречник каишника компресора 120 *mm*. На основу ових пречника, долази се до преносног односа каишног преносника (3):

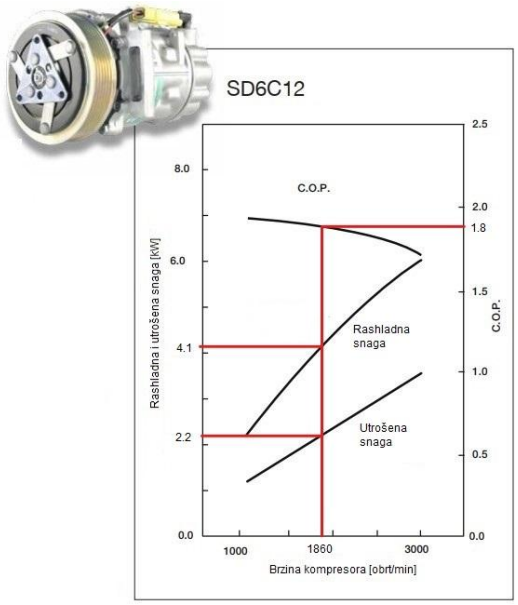
$$i = \frac{i_{kom}}{i_{kol}} = \frac{120}{158} = 0{,}759 \qquad (3)$$

Табела 4. *Радни режим компресора*

n_4 [min^{-1}]	n_AC_4 [min^{-1}]	n_5 [min^{-1}]	n_AC_5 [min^{-1}]	p_IV_AC [bar]	p_V_AC [bar]
1767	2327	1413	1860	12,58	11,64



Слика 4. *Номинална радна тачка у 4. степену преноса*



Слика 5. *Номинална радна тачка у 5. степену преноса*

3. ЗАКЉУЧАК

У случају коришћења клима-уређаја, потрошња горива се повећава у просеку око 0,5 *L* на пређених 100 *km*, без обзира на то у којем степену преноса се налази мењач. За исти степен преноса разлика, у процентима износи 8 – 18% када се гледа мерење са и без клима-уређаја. То је са аспекта економичности и домета возила је негативно, али са аспекта ергономије, тј. топлотног комфора у кабини возила је неопходно.

Стање асфалта, односно пута по којем се креће возило, значајно утиче на потрошњу горива, без обзира да ли је клима-уређај укључен или не. На неравној (грбавој) подлози, има случајева да су потрошња горива са укљученим и искљученим клима-уређајем веома блиске, чак у неколико наврата потрошња горива са искљученим клима-уређајем је већа од потрошње са укљученим клима-уређајем.

При брзини од 70 *km/h*, економичнија вожња се постиже при кретању у 5. степену преноса, тј. када мотор СУС ради на нижим бројевима обртаја (у 4. степену преноса троши се око 21% више горива у односу на 5. степен). На основу карактеристике компресора закључује се да мање енергије троши при нижим обртајима (1413 *obrt/min* у односу на 1767 *obrt/min* коленастог вратила).

4. ЛИТЕРАТУРА

[1] Стојић Б.: „Теорија кретања друмских возила“, Факултет техничких наука, Нови Сад, 2014.
[2] Дудаш А.: „Теоријске основе снимања брзинске карактеристике мотора помоћу дијагностичког уређаја“, дипломски рад, Факултет техничких наука, Нови Сад, 2014.
[3] Ружић Д.: „Ергономија моторних возила“, Факултет техничких наука, Нови Сад, 2020.
[4] Ружић Д.: „Опрема моторних возила - скрипта“, Факултет техничких наука, Нови Сад, 2023.

[5] https://x-engineer.org/real-world-fuel-consumption/ (приступљено август 2025.)
[6] https://x-engineer.org/obd-system/ (приступљено август 2025.)
[7] https://x-engineer.org/on-board-diagnostics-obd-modes-operation-diagnostic-services/ (приступљено август 2025.)
[8] https://x-engineer.org/automotive-diagnostic-standards/ (приступљено август 2025.)

Кратка биографија:

Милош Вујковић рођен у Шапцу, 2000. године. Основне академске студије, завршио 2023. године, на Факултету техничких наука, где је 2025. године одбранио мастер рад из области Моторна возила, смер Аутомобилско инжењерство.

Драган Ружић (1973) докторирао је на Факултету техничких наука из области топлотне ергономије у моторним возилима 2013. год. Запослен је на Факултету техничких наука од 2000. год., а од 2024. год. је у звању редовног професора на Катедри за моторе и возила.

Зборник радова Факултета техничких наука, Нови Сад

UDK: 621.7
DOI: https://doi.org/10.24867/33AM07Zadric

МЕТОДЕ ИНЖЕЊЕРСКЕ АНАЛИЗЕ И САВЕРЕМЕНИХ ТЕХНИКА ПРОЈЕКТОВАЊА НА ПРИМЕРУ РАЗВОЈА ШАСИЈЕ ТРКАЧКОГ БОЛИДА

METHODS OF ENGINEERING ANALYSIS AND MODERN DESIGNING TECHNIQUES IN DEVELOPMENT OF A RACE CAR CHASSIS

Марко Задрић Бардак, Драган Живанић, *Факултет техничких наука, Нови Сад*

Област - МАШИНСТВО

Кратак садржај – *Инжењерска анализа и савремене технике пројектовања представљају елементарне алате којима се инжењери модерне индустрије служе у свакодневном пројектовању. У овом раду биће описана теоријска основа поменутих метода, такође ће бити приказана практична примена метода на примеру развоја, пројектовања и анализе шасије тркачког болида.*

Кључне речи: *пројектовање, инжењерска анализа, развој, МКЕ*

Abstract – *Engineering analysis and modern designing techniques represent essential tools that are used everyday by engineers in the modern industry. This paper will describe the theorethical basis of the mentioned methods, it will also present the practical usage of the methods on the example of development, designing and analysis of a race car chassis.*

Keywords: *Design, engineering analysis, development, FEA*

1. УВОД

Инжењерска анализа у машинству представља фундаментални алат за разумевање, пројектовање и оптимизацију механичких система. Инжењерска анализа подразумева систематски приступ моделовању физичких појава помоћу коришћења савремених нумеричких метода и софтверских алата. Анализа омогућава инжењерима да предвиде понашање компоненти и система услед различитих услова оптерећења, температуре и режима рада. У овом раду обрађени су теоријски основи инжењерске анализе, њене кључне методе као што су метода коначних елемената (МКЕ), метода коначних запремина (МКЗ) и остали аналитички приступи [1]. Такође, детаљно су размотрене примене инжењерске анализе у конструкцијама, преносу топлоте, динамици система и протоку флуида, уз студије случаја из праксе.

НАПОМЕНА:
Овај рад је проистекао из мастер рада чији ментор је био др Драган Живанић, редовни професор.

2. ТЕОРИЈСКЕ ОСНОВЕ

Основни концепт методе коначних елемената се заснива на претпоставци да се сложен контуинуални систем може апроксимирати дискретним скупом коначних елемената који су међусобно повезани у чворове [2]. Сваки елемент се понаша према дефинисаним законима, на пример према Хуковом закону за линеарно еластичне материјале. Основа математичке формулације полази од диференцијалне једначине равнотеже и примењује се принцип виртуелног рада или варијациони приступ. Дискретизацијом се добија систем линеарних или нелинеарних алгебарских једначина:

$$[K] \cdot \{u\} = \{F\} \qquad (1)$$

Где су:

[K] – матрица крутости;

{u} – вектор померања;

{F} – вектор спољашњих сила.

Циљ статичке анализе је израчунавање напона идеформација унутар конструкционог елемента. Унутрашњи напони се развијају у телу на начин који обезбеђује равнотежу унутар сваког дела запремине, тако да се може применити принцип равнотеже, како би се израчунале ове величине, односно да је сума свих сила које делују на ограничену запремину тела једнака нули. Овакав принцип рада омогућује брзо решење на примеру једноставне греде – уз помоћ равнотеже сила могуће је израчунати момент савијања и силу смицања, а из тога даље израчунати нормалне и тангенцијалне напоне. Применом равнотеже на 2D облике рачун постаје тежи, а за 3D тела још сложенији.

Метода коначних елемената приступа решавању оваквом сложеном проблему тако што тело дели на већи број малих елемената који су међусобно повезани чворовима. Овакав процес се назива дискретизација, а скуп елемената и чворова се назива мрежа (*mesh*). Дискретизација је корисна јер у том случају процес рачуна из закона равнотеже мора бити задовољен само над коначним бројем дискретних елемената, уместо над целим телом. Резултат дискретизације је просторна мрежа коначних елемената у облику геометрије задатог модела. Кориснику је омогућено да произвољно изабере

густину мреже коначних елемената која директно зависи од величине коначног елемента. Број коначних елемената знатно утиче на захтеве перформанси рачунара па тако и на потребан временски период који рачунар изискује како би дошао до решења анализе. Такође, што су коначни елементи мањи и њихова количина већа, то ће вредности резултата анализе бити прецизнији.

3. ПОСТУПАК МОДЕЛИРАЊА И ПРИПРЕМЕ МОДЕЛА ЗА АНАЛИЗУ

Рачунарски подржано пројектовање (CAD) представља инжењерски процес у коме се проблеми са којима се инжењери сусрећу у реалном свету, представљају помоћу графичких и запреминских тродимензионалних модела који су добијени путем рачунара. Поменутим поступком се креира дигитални близанац производа у рачунарском окружењу на којем је могуће вршити брзе измене, анализе и прорачуне, као и креирање техничке документације. Процес геометријског моделовања шасије започет је дефинисањем координатног почетка, чија X оса представља подужну осу, док Y оса представља осу предње осовине тркачког болида. Затим се дефинисала раван која пролази кроз поменуту осу, али и кроз вертикалну Z осу чиме се добила раван која је коришћена као нулта раван модела. Нове равни су креиране помоћу команде којом се дефинише одстојање од нулте равни, где се за поменуту мерну величину увео параметар ради лакше касније промене.

Следећи корак у параметарском моделовању је било дефинисање и повезивање дефинисаних параметара са мерним величинама као што су дужине и углови. Нови уведени параметри, су затим математичком формулом изједначени, а потом су им дефинисани називи параметара. На основу креиране референтне просторне геометрије, уведени су цевни профили стандардних величина помоћу алата *Frame generator* [4] чиме се добија изглед коначног тродимензионог модела анализираног производа.

Генерисање мреже односно дискретизација тродимензионог модела шасије на коначни број елемената извршена је путем модула *NASTRAN*, помоћу којег је омогућено кориснику да произвољно изабере величину коначног елемента, која директно утиче на прецизност резултата симулације и временско трајање рачунарске анализе. Затим су дефинисани гранични услови у складу са сценаријем оптерећења. Потом су на основу анализе већ постојеће научне литературе [5, 8-13] уведена оптерећења. Вредности оптерећења добијена су путем прорачуна који користи елементарне механичке принципе једначина равнотеже.

Прорачун је обухватио случајеве оптерећења који се помињу у техничком правилнику студентског такмичења *Formula student* [6]. Анализирани су случајеви попут оптерећења шасије торзионим моментом у тренутку наиласка тркачког болида на избочину или рупу у подлози; случај оптерећења моментом савијања услед сопствене силе тежине и тежине комопонената и система на возилу, случај ударног оптерећења предњег краја болида; случај ударног оптерећења бочне стране болида.

Поменути сценарији су изабрани за критичне случајеве анализе на основу смерница дефинисаним техничким правилником такмичења. У техничком правилнику најстрожије су дефинисани безбедносни услови тркачког болида. Сигурност конструкције доказује се анализом вредности максималних напона шасије у случајевима ударног оптерећења предњег краја болида и ударног оптерећења бочне стране болида. Дати случајеви оптерећења најтачније предвиђају могућност појаве отказа конструкције приликом судара, што угрожава безбедност возача болида. Случајеви оптерећења шасије торзионим моментом и моментом савијања услед сопствене тежине и тежине компонената возила, изабрани су како би се предвиделе тркачке перформансе возила. Анализом два поменута случаја омогућава се израчунавање вредности угаоне крутости шасије. Израчуната величина директно утиче на вертикалну компоненту силе која се појављује на појединачном точку. Множењем вертикалне компоненте силе са коефицијентом трења, могуће је израчунати вредност приањања пнеуматика. Потом се израчунате величине користе у прорачуну брзинске карактеристике возила која предвиђа перформансе возила у случају скретања и кочења, без потребе да се болид тестира на стази.

4. РЕЗУЛТАТИ АНАЛИЗЕ МЕТОДОМ КОНАЧНИХ ЕЛЕМЕНАТА

За потребе прорачуна овог рада усвојен је материјал 25CrMo4/1.728, прохромовани челик који поседује следеће механичке карактеристике:

ρ=7860 kg/m^3 - густина,

σ_u=560 MPa - затезна чврстоћа;

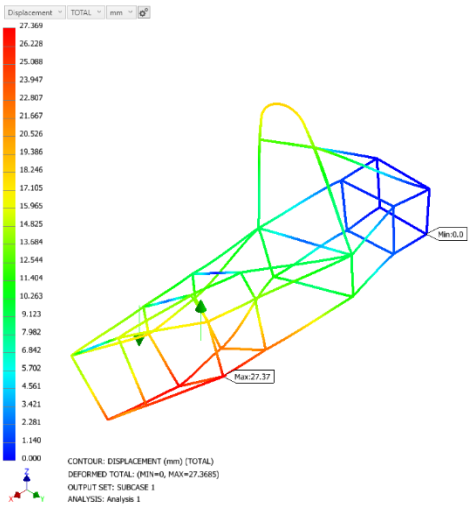
σ_y=450 MPa - напон течења (максимални дозвољени напон односно вредност напона при којој материјал подлеже пластичној деформацији).

Након извршеног рачунарског прорачуна за поменуте случајеве оптерећења, могуће је анализом резултата у виду приказивања вредности максималних напона и деформација, доћи до прелиминарних сазнања у вези локације концентрације напона у конструкцији. Да би се потврдила тачност резултата, потребно је постићи конвергенцију максималних напона. Конвергенција представља случај у коме се резултати вредности напона две узастопне итерације симулације не разликују за више од 5%. Због наведених разлога, рачунарске симулације поменутих оптерећења су поновљене са коначним елементима мање величине.

Приказани су примери случаја оптерећења моментом торзије и случаја ударног оптерећења предњег краја шасије тркачког болида. Добијена вредност напона у првој итерацији симулације је 320.8 MPa, док израчуната вредност исте у другој итерацији симулације износи 331.5 MPa. Израчунате вредности конвергирају 1.65%, чиме се доказује тачност симулације, те нема потребе за даљим умањивањем величине коначних елемената.

Табела 1: *Вредности резултата симулације оптерећења шасије торзионим моментом*

Физичка величина	Напон	Деформација
Прва итерација	320.8 MPa	27.369 mm
Друга итерација	331.5 MPa	27.369 mm
Конвергенција	3.35%	0%

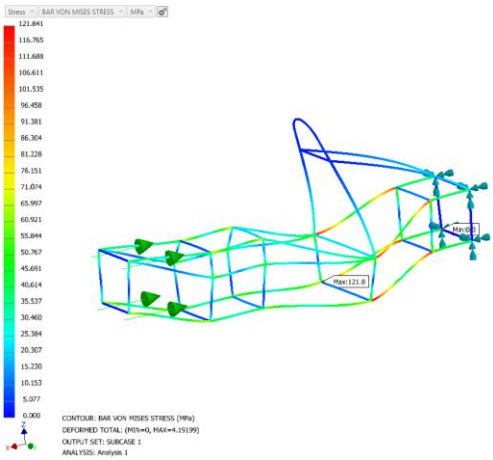


Слика 1. *Вредности деформације друге итерације оптерећења шасије торзионим моментом*

Такође се може закључити, на основу поређења вредности максималног напона који се јавља као резултат у другој итерацији симулације торзионог оптерећења, да је степен сигурности задовољен. Степен сигурности за случај торзионог оптерећења износи:

$$\text{Степен сигурности} = \frac{460\ MPa}{331.5\ MPa} = 1.38 \qquad (2)$$

Добијена вредност напона у првој итерацији симулације ударног оптерећења предњег краја болида је 121.8 MPa, док је израчуната вредност исте у другој итерацији симулације 120.3 MPa. Израчунате вредности конвергирају 1.23%, чиме се доказује тачност симулације, те нема потребе за даљим умањивањем величине коначних елемената.



Слика 2. *Вредности напона друге итерације ударног оптерећења предњег краја*

Табела 2: *Вредности резултата симулације ударног оптерећења предњег краја шасије*

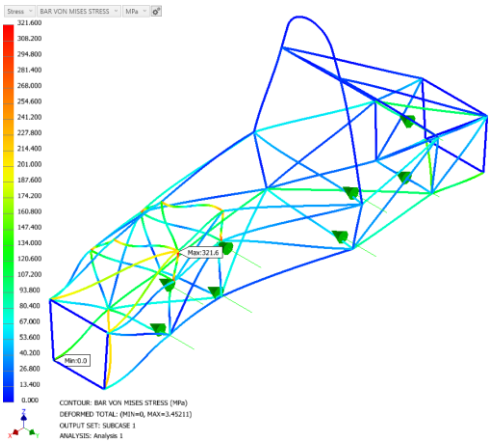
Физичка величина	Напон	Деформација
Прва итерација	121.8 MPa	4.192 mm
Друга итерација	120.3 MPa	4.192 mm
Конвергенција	1.23%	0%

Након друге итерације и потврђене конвергенције, вредност напона је убачена у формулу за израчунавање степена сигурности при ударном оптерећењу предњег краја:

$$\text{Степен сигурности} = \frac{460\ MPa}{120.3\ MPa} = 3.8 \qquad (3)$$

Након анализе наведених случајева закључено је да су најмање вредности степена сигурности, који износе 1.22 и 1.38. То је уочено код ударног оптерећења бочне стране болида и оптерећења торзионим моментом. Потом је извршена оптимизација дизајна шасије у смеру ојачавања конструкције за наведене случајеве. Додати су потпорни цевни профили мањих димензија попречног пресека, у виду дијагоналних укрућења [7] између постојећих профила у конструкцији.

Потом је рачунарска симулација поновљена са оптимизованим дизајном шасије за случај ударног оптерећења бочне стране шасије тркачког болида. Добијена вредност напона у првој итерацији симулације је 298.8 MPa, док је израчуната вредност исте у другој итерацији симулације 316.6 MPa. Израчунате вредности конвергирају 5.95%, чиме није доказана тачност симулације, те постоји потреба за даљим умањивањем величине коначних елемената.



Слика 3. *Вредности напона треће итерације ударног оптерећења бочне стране шасије*

Добијена вредност напона у другој итерацији симулације је 316.6 MPa, док је израчуната вредност исте у трећој итерацији симулације 321.6 MPa. Израчунате вредности конвергирају 1.57%, доказујући тачност симулације, те нема потребе за даљим умањивањем величине коначних елемената. Након треће итерације и потврђене конвергенције, вредност напона се може убацити у формулу за израчунавање

степена сигурности при ударном оптерећењу предњег краја тркачког болида:

$$\text{Степен сигурности} = \frac{460\ MPa}{321.6\ MPa} = 1.43 \qquad (4)$$

Табела 3: *Вредности резултата симулације ударног оптерећења бочне стране шасије*

Физичка величина	Напон	Деформација
Прва итерација	298.8 MPa	3.452 mm
Друга итерација	316.6 MPa	3.452 mm
Конвергенција	5.95%	0%
Трећа итерација	321.6 MPa	3.452 mm
Конвергенција	1.57%	0%

5. ЗАКЉУЧАК

Закључено је да добијени степен сигурности за случај ударног оптерећења бочне стране оптимизоване шасије износи 1.43, чиме је степен сигурности повећан до нивоа задовољавајуће вредности. На основу добијених резултата, може се закључити да је шасија тркачког болида оптимизована на исправан начин. Вредност максималног напона који се јавља као резултат симулације се смањила за 55.2 MPa, док се вредност степена сигурности повећала за 14.5%.

Верификацију тачности резултата могуће је потврдити путем извођења експерименталне анализе. Након пројектовања, производње, следи постављање мерних инструмената у виду мерних трака. Мерне траке представљају елементе који на основу пиезоелектричног ефекта детектују промену вредности отпорности проводника. Потом би се шасија оптеретила израчунатим силама које су добијене кроз прорачун. Вредности максималних напона који се појављују у конструкцији, могуће је израчунати на основу вредности измерених померања и познавања модула еластичности материјала. Додатан вид верификације тачности добијених резултата огледа се у понављању инжењерске анализе коришћењем различитих софтвера. Коначна провера исправности правца развоја архитектуре возила извршила би се понављањем компјутерске анализе и поступка мерења конструкције, која је произведена од другог материјала. Таквим истраживањем долази се до поређења вредности максималних напона и механичких карактеристика различитих материјала у зависности од укупне масе конструкције.

Пример развоја и оптимизације шасије тркачког болида показује примену савремених техника пројектовања. То су параметризација и инжењерска анализа. Поменуте две технике у многоме користе и доприносе у процесу развоја производа, у виду умањења потребног времена за развој и имплементацију промена у дизајну. Омогућено је предвиђање понашања конструкције приликом реалних случајева у експлоатацији, чиме се долази до вредних информација и сазнања.

6. ЛИТЕРАТУРА

[1] Joseph Edward Shigley, “Mechanical Engineering Design“, 2020, ISBN 978-0-07-339821-1.

[2] Радомир Ђокић, “Инжењерска анализа“, 2021, скрипта, Факултет Техничких Наука, Нови Сад

[3] Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. Z., “The Finite Element Method: Its Basis and Fundamentals , 2013, Butterworth-Heinemann, ISBN 0-7506-6320-0.

[4] Никола Иланковић, “Савремене технике пројектовања транспортних система“, 2023, скрипта, Факултет Техничких Наука, Нови Сад

[5] William B. Riley and Albert R. George, Cornell University, “Design, Analysis and Testing of a Formula SAE Car Chassis“, 2002, ISSN 0148-7191.

[6] Formula Society of Automotive Engineers, FSAE, “Formula Student Rulebook“, 2025, Institution of Mechanical Engineers.

[7] Carroll Smith, SAE/PT-90, Society of Automotive Engineers, “Racing Chassis and Suspension Design“, 2004, ISBN 0-7680-1120-5.

[8] Krunal Kania, Aditya Verma, Rushang Babariya, Jayendra Vasani, *IRJET*, “Design & Development of Formula Student Chassis“, 2023, Vol. 10, ISSN: 2395-0056.

[9] Mohammed Tazeem Khan, *IRJET*, “Design & Manufacturing of FSAE Chassis“, 2021, Vol. 3, e-ISSN: 2582-5208.

[10] Felix Dionisius, Imam Nur Arif, Tito Endramawan, Agus Sifa Badruzzama, “Material Selection and Analysis of Torsional Rigidity in Formula Student SAE regulation“, 2022, Vol 22., ISSN: 2549 – 9815.

[11] Sayed Zameer, Sayyed Affan Ali, Tambe Salman, Kharbe Mohammad, *IRJET,* “Static Structural Analysis of Chassis in Compliance with International Rules of a Prototype Formula Styled Vehicle“, 2022, Vol 9, e-ISSN: 2395-0056.

DOI: https://doi.org/10.24867/33AM07Zadric

DOI: https://doi.org/10.24867/33AM07Zadric

Кратка биографија:

Марко Задрић Бардак, рођен у Суботици, 2000. године. Основне академске студије завршио 2023. године, на Факултету техничких наука, где је 2025. године одбранио мастер рад на катедри за Конструкционо машинство, транспортне системе и логистику.

Драган Живанић рођен у Сремској Митровици, 1972. Године. Докторирао је на Факултету техничких наука 2012. Године. Запослен на факултету од 1997. Године, изабран у звање ванредног професора 2019, а у звање редовног професора изабран је 2024. Године.

Зборник радова факултета техничких наука, Нови Сад

UDK: 621

DOI: https://doi.org/10.24867/33AM08Burlica

КОРИШЋЕЊЕ ХТГР НУКЛЕАРНИХ РЕАКТОРА У ОКВИРУ КОГЕНЕРАТИВНИХ СИСТЕМА ЗА ПРОИЗВОДЊУ ЕНЕРГИЈЕ

UTILIZATION OF HTGR NUCLEAR REACTORS WITHIN COGENERATION ENERGY SYSTEMS

Исидора Бурлица, *Факултет техничких наука, Нови Сад*

Област – МАШИНСТВО

Кратак садржај – *Мастер рад се састоји из три велике целине: општег увода у нуклеарну енергију и нуклеарне реакторе, упознавања са ХТГР реакторима и њиховим принципом рада и примени ове технологије у когенеративним постројењима. У првој целини објашњене су нуклеарне реакције, делови реактора, а затим и његов принцип рада и подела нуклеарних електрана. Други део обухвата опис ТРИСО горива и принцип рада ХТГР реактора, са посебним нагласком на безбедност, како је то једна од његових најбитнијих карактеристика. Последња целина рада тиче се интеграције ХТГР-а у когенеративне енергетске системе, уз кратак осврт на економске аспекте ове технологије.*

Кључне речи: нуклеарна енергија, ХТГР реактор, когенеративна постројења, ТРИСО гориво.

Abstract – *The Master's thesis is structured into three main sections: a general introduction to nuclear energy and nuclear reactors, an overview of HTGR reactors and their operating principles, and the application of this technology in cogeneration plants. The first section explains nuclear reactions, the components of a reactor, as well as its operating principle and the classification of nuclear power plants. The second section covers the description of TRISO fuel and the operating principle of HTGR reactors, with particular emphasis on safety, as it represents one of their most important characteristics. The final section of the thesis concerns the integration of HTGRs into cogeneration energy systems, including a brief consideration of the economic aspects of this technology.*

Keywords: *nuclear energy, HTGR reactor, cogenerative plants, TRISO fuel*

1. УВОД

Гасови стаклене баште све више угрожавају нашу планету и здравље. У циљу ограничења пораста глобалне температуре на 1,5°C до краја века, одржан је Париски споразум, међународни уговор о климатским променама. Верује се да нуклеарна енергија може играти велику улогу у постизању овог циља, обзиром на њену поузданост и распрострањеност горива. Посебну пажњу је задобио ХТГР реактор, који захваљујући високим температурама радног медијума на крају процеса, може когенеративно да се искористи за још неки индустријски процес, поред примарне производње електричне енергије.

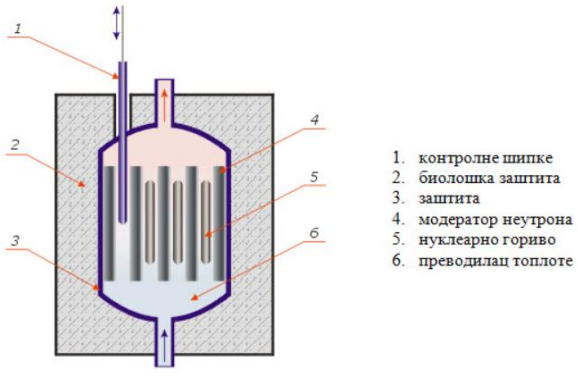
НАПОМЕНА: Овај рад проистекао је из мастер рада чији је ментор био др Ђорђије Додер, ванр. проф.

2. НУКЛЕАРНА ФИЗИКА

Основне нуклеарне реакције у природи су фузија и фисија. Фузијом се лакша атомска језгра спајају у једно теже при чему ослобађају енергију, док је фисија процес цепања нестабилног језгра на два, чиме се ослобађа енергија која се користи у данашњим комерцијалним реакторима.

2.1. Нуклеарни реактор

Нуклеарни реактор је посуда под притиском у којој се одвија контролисана ланчана реакција фисије [3]. Његови делови су гориво, модератор, управљачке шипке, систем за хлађење, рефлектор и заштитни систем, а приказани су на *слици 1:*



Слика 1. *Делови нуклеарног реактора [1]*

Гориво може да нађе у облику горивих шипки и као шљунковито гориво. Контролне шипке омогућавају регулисање ланчане реакције у виду повећања или смањења стопе фисије. Модераторе користимо како би успорили брзе неутроне. Рефлектор служи да врати неутроне назад у средиште реактора и тако да искористимо што већи број неутрона. Rashladno средство, тј. радни флуид је медијум који преноси

топлоту из језгра реактора до турбине, а такође самим тим и хлади реактор. Све ове компоненте су смештене у реакторској посуди под притиском.

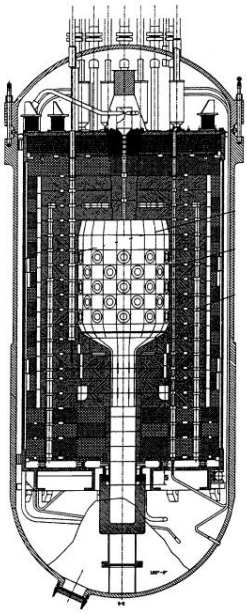
Принцип рада нуклеарне електране је следећи: у језгру се одвија процес фисије. Кроз реактор циркулише медијум који преноси топлоту насталу цепањем језгра атома. Расхладно средство у измењивачу топлоте индиректно загрева воду и претвара је у пару, због чега се оно хлади и наставља да циркулише кроз систем. Настала водена пара се доводи до турбине коју окреће, чиме се покреће генератор који производи електричну енергију.

Постоје четири генерације нуклеарних реактора, од којих је свака напреднија од своје претходне, а ХТГР спада у четврту. Реактори се такође могу класификовати према енергији неутрона који изазивају фисију, rashladnom средству, употреби и фази горива.

Призматични дизајн и дизајн са шљунковитим лежиштем, се виде на *слици 2:*

Слика 2. *Призматични дизајн и дизајн са шљунковитим лежиштем [5]*

3. ХТГР РЕАКТОРИ



Слика 3. *ХТР-10* [2]

Високо температурни гасом хлађени реактор (ХТГР) је технологија нуклеарног фисионог реактора који користи хелијум за хлађење и графит за модерацију [2]. Гориво ових реактора је смештено у ТРИСО честице које се састоје од језгра уранијума, угљеника и кисеоника, а облога од три слоја материјала на бази керамике спречава ослобађање радиоактивних производа фисије ван ових граница [2]. Опис ХТГР технологије ће се усвојити од постојећег ХТР-10 кинеског реактора (слика 3).

Језгро се састоји од 27.000 горивих елемената и окружено је графитним рефлекторима (горе, доле и бочно) [2].

Канали за хладан хелијум су пројектовани унутар бочног рефлектора и кроз њих гас путује нагоре након што уђе у посуду под притиском, а затим му се ток обрће на врху језгра реактора како би скренуо ка лежишту од куглица (гориву). Након што се загреје и дође до доњег дела реактора, ту улази у цев за врућ гас и њом путује до измењивача топлоте [2].

2.2 Безбедност ХТГР-а

Нуклеарне несреће углавном настају због заостале/остатне топлоте распадања што доводи до топљења језгра. У конвенционалним реакторима, одвођење топлоте се одвија активним системима за хлађење као што су пумпе, док је расхладни флуид вода, који не може да врши своју улогу уколико дође до промене њеног агрегатног стања. У ХТГР реактору, остатна топлота се одводи путем гаса–најчешће хелијума, где чак и при отказу компресора, хлађење реактора ће се наставити захваљујући природној конвекцији. Хелијум не може да реагује са другим хемијским елементима, није запаљив и не може постати радиоактиван.

Топљење језгра није могуће [3].

Највиша температура која може да се постигне у језгру реактора (1600°C у екстремним условима) је испод било које температуре при којој би се гориво оштетило, захваљујући ТРИСО дизајну горива [3].

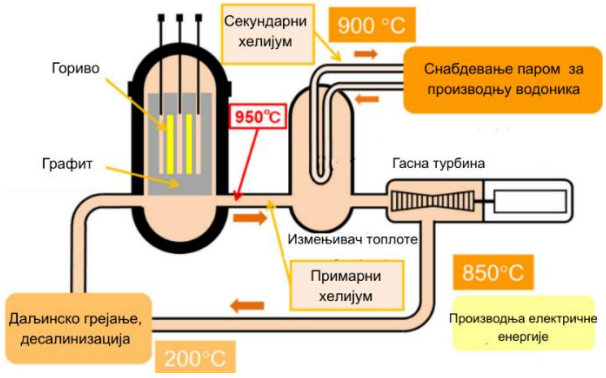
Чак и ако контролне шипке, апсорбујуће куглице или хелијум откажу, реактор ће природно зауставити фисију и охладиће се сам од себе због негативног температурног коефицијента реактивности [3].

Велики је однос површине суда у односу на запремину, што значи да је топлота која се изгуби преко површине језгра већа од оне која се ствара распадом горива у језгру [3].

Упркос свим контроверзијама у свету, доказано је да је нуклеарна енергија безбеднија од енергије из фосилних горива. Истраживања су показала да је угаљ усмртио 100 пута више људи од нуклеарне енергије, углавном као последица загађења ваздуха, а откад свет користи енергију фисије, израчунато је да је сачувано 1,84 милиона живота између година 1971. и 2009. Чак се процењује да су енергија биомасе, соларна, хидро и ветро енергија однеле више живота од нуклеарне [3].

4. ИНТЕГРАЦИЈА ХТГР-А У КОГЕНЕРАТИВНЕ СИСТЕМЕ

Нуклеарна когенерација представља истовремену производњу електричне енергије и топлоте или производа добијених из топлоте у току рада нуклеарне електране. ХТГР технологија је идеална за ову сврху, обзиром на то да су температуре хелијума на крају процеса високе и као такве углавном се само пуштају у атмосферу и губе свој потенцијал да се преусмере као извор топлоте на још неки процес. Општи процес когенерације из нуклеарне енергије је приказан на *слици 4:*



Слика 3. *Шема когенеративног ХТГР постројења [5]*

Секундарна топлота из ХТГР-а се може искористити у процесима за даљинско грејање, дестилацију, процесну топлоту за индустријске системе, експлоатацију нафте и катранских пескова, прераду сирове нафте, утечњавање и гасификацију угља, производњу амонијака, метанола, тешких метала и водоника.

4.1 Систем даљинског грејања

Примена система даљинског грејања зависи од климе и могућности да се у тој жељеној регији обезбеди ниво комфора у грејању. Главни извори енергије који се данас користе за производњу топлоте за домаћинства су угаљ и гас, док је проценат нуклеарне топлоте у ове сврхе занемарљив.

Општи принцип рада когенерације топлоте за даљинско грејање из ХТГР-а почиње са делом паре која излази из измењивача топлоте и која се усмерава ка посебној турбини ниског притиска. Тако експандирана пара се затим кондензује у измењивачу топлоте, предајући топлоту у цевовод за даљинско грејање. Модификације електране ради увођења овог система су прилично умерене и могу се реализовати без ометања уобичајеног рада постројења [4].

4.2 Десалинизација

Како се све већи број места суочава са несташицом воде, десалинизација ће постати кључна технологија за решавање овог проблема. Ово иде у прилог нуклеарној енергији, за коју се очекује да ће постати одржива опција за велике системе десалинизације широм света [4].

Десалинизацију воде можемо да вршимо помоћу више техника: РО (обрнуте осмозе), МСФ (вишестепена флеш дестилација) или МЕД (вишеефектна дестилација) техника. Иако се МСФ показала као најједноставнији и најпоузданији од главних процеса, њен значајан недостатак је већа потрошња енергије у односу на МЕД и РО. Међутим, решење може да лежи управо у ХТГР реакторима, који би ефикасно понудили своју отпадну топлоту за овај процес.

Примену овог случаја можемо наћи на јапанском ХТГР реактору ГХРГР3000.

4.3 Процесна топлота за индустријске системе

У зависности од температурног опсега, потребе за топлотом у оквиру одређеног индустријског процеса могу се задовољити прилагођавањем конфигурације когенеративног система одговарајућем типу нуклеарног реактора [4].

Поред производње електричне енергије, висока излазна температура флуида може се користити за низ индустријских процеса који захтевају топлоту у распону од 650°C до 950°C. Неке од примера су:

- експлоатација нафте и катранских пескова
- прерада сирове нафте
- утечњавање и гасификација угља
- производња амонијака
- производња метанола
- производња тешких метала

4.1 Производња водоника

Неки сектори су компликованији у постизању декарбонизације, као на пример авијација и камионски транспорт. Један од начина њиховог декарбонизовања може бити употреба водоника као горива, али да би цео систем био CO2 неутралан, тај водоник морамо да добијемо из неког чистог извора. Тада наступају ХТГР реактори, као идеално решење, чија је отпадна топлота довољна да произведе извор енергије потребан за покретање реакције електролизе.

Електролиза је процес у којем се помоћу електричне енергије вода разлаже на водоник и кисеоник, док се све то одвија у уређају званом електролизер.

Водоник се може когенерисати уз помоћ нуклеарне топлоте електролизом воде или паре, хемијским реформингом фосилних горива и биомасе и термохемијским процесима разградње паре.

Принцип рада је следећи: примарни хелијум, загрејан до високе температуре у реактору, предаје своју топлоту секундарном хелијуму у интермедијарном измењивачу топлоте унутар зграде реактора. Секундарни хелијум, који је потпуно чист и без радиоактивности, излази из заштитног система и преко посебног цевовода преноси топлоту до постројења за производњу водоника (нпр. реформера или термохемијског циклуса). У том процесу се, уз помоћ доведене топлоте, одвија конверзија сировине у водоник. Након што је предао топлоту и охладио се, секундарни хелијум се помоћу циркулационе пумпе враћа у интермедијарни измењивач топлоте, чиме се циклус затвара.

5. ЕКОНОМСКИ АСПЕКТИ

Трошкови изградње нуклеарних електрана драматично су порасли у последњих неколико година: са 2.500 €/kW двадесетих година на више од 6.000 €/kW, што је један од већих проблема за њихов развој. Нуклеарне електране су велики пројекти чија изградња захтева неколико хиљада радника и читаву деценију грађевинских радова. Ситуација је још неповољнија за ХТГР реакторе, како је њихова технологија још комплекснија и мање комерцијална.

У цену ХТГР-а убрајају се трошкови изградње, горива и трошкови рада и одржавања.

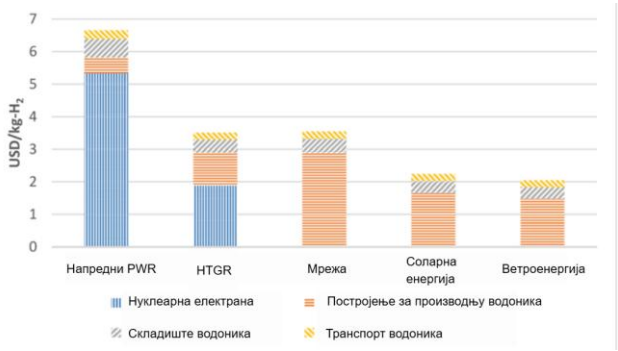
Када је у питању производња водоника, обновљиви извори нуде најнижи трошак, посебно ветроелектране, са ценом од 2,05 $/kg водоника, што је нешто ниже у односу на соларне електране чија цена износи 2,24 $/kg [6]. При поређењу напредних реактора са водом под притиском и ХТГР реактора, свакако да ХТГР остварује нижи трошак производње водоника [6].

Значајну улогу у смањењу трошка водоника игра повећање обима постројења (снаге и модула). Многе студије случаја показују да нуклеарно даљинско грејање може бити ценовно конкурентно чак и у односу на природни гас, што је још лакше остваривo уз примену пореза на угљеник [4].

Производња амонијака уз коришћење нуклеарне енергије има најниже оперативне трошкове, али највише капиталне.

Најнижи трошак производње водоника постигнут је коришћењем електричне енергије из ветра, а одмах затим и из соларних фотонапонских система. Ове вредности су за око 1,3 $/kg ниже од оних из ХТГР [6].

Резултати су приказани на *слици 6:*



Слика 4. *Поређење трошкова производње водоника из различитих извора енергије [6]*

Међутим,ове анализе често не узимају у обзир недоступнст обновљивих извора у зависности од доба дана иременске прогнозе, што нам не даје загарантоану производњу када нам је потребна. То би подразумевало додатни трошак у виду складишта енергије и додатне емисије при производњи исте. Такође се не узима у обзир чињеница да нуклеарна електрана заузима знатно мање простора од соларних електрана и ветропаркова, што отвара могућност ка искоришћењу тог земљишта за још нешто што би могло бити профитабилно. Другим речима, сама цена произведене сировине нема увек главну улогу у одабиру начина из којег ће водоник да се произведе.

Уколико би се ова технологија комерцијализовала и почела да се производи масовније, њена цена би свакако значајно опала. Нажалост, овде више улогу не играју само цифре, већ и људска осећања која су често негативна када је у питању нуклеарна технологија.

6. ЗАКЉУЧАК

ХТГР реактори представљају зрелу технологију која може да обезбеди стабилно снабдевање топлотном и електричном енергијом. Посебно се одликују високим нивоом безбедности који омогућавају напредни сигурносни системи.

Топлота произведена уз помоћ ХТГР-а може да се искористи, поред производње електричне енергије, и за широк спектар додатних производа који могу да буду део даљинског грејања и хлађења, процесне топлоте, десалинизације, производње водоника и друго.

Обзиром на неповерење које су људи изградили према нуклеарној енергији, ХТГР реактори су можда последња шанса да се то поверење врати, ако би се стручна лица ангажовала да изнесу све чињенице и појашњења о високој безбедности и ефикасности ове технологије.

7. ЛИТЕРАТУРА

[1] Nuklearni reaktori. Fizis.rs – Физика–атомског језгра. https://fizis.rs/гимназија/iv-разред/физика-атомског-језгра/nuklearni-reaktori/ [Приступ 10.06.2025].

[2] Advanced Nuclear Materials in HTGR. In: High Temperature Reactor Technology Workshop ICTP; 19 - 30 2008; Trieste, Italy.

[3] PBMR's Safety Features. U.S. Nuclear Regulatory Commission. https://www.nrc.gov/docs/ML0310/ML031000208.pdf [Приступ 20.06.2025].

[4] Guidance on Nuclear Energy Cogeneration. IAEA Nuclear Energy Series No. NP-T-1.17. Vienna: International Atomic Energy Agency; 2019.

[5] Tachibana Y. "Current Status of JAEA's Research and Development on HTGR". Presentation at Nuclear Innovation Workshop; 2022 Mar 1; Tokyo, Japan. https://nicp.ne.titech.ac.jp/jp/nicc/2022/220301/JAEA_HTGR.pdf [Приступ 09.07.2025]

[6]Alabbadi AA, Smith RL, Brown P, et al. "A comparative economic study of hydrogen production using several nuclear reactors integrated with electrolysis hydrogen production methods". International Journal of Hydrogen Energy. 2024; 49(38):12688-12704.

doi:10.1016/j.ijhydene.2023.04.2787.

Исидора Бурлица је рођена у Сремској Митровици 1999. године. Мастер рад на Факултету техничких наука из области Машинства одбранила је 2025. године. Контакт:

isidoraburlica@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE01Aleksic

SISTEM ZA PRIKUPLJANJE I PRETRAŽIVANJE PODATAKA O IGRAČIMA EVROLIGE

SYSTEM FOR COLLECTING AND SEARCHING DATA ABOUT EUROLEAGUE BASKETBALL PLAYERS

Nikola Aleksić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljen je sistem u obliku veb aplikacije koja na jednom mestu objedinuje statističke podatke o igračima Evrolige, omogućavajući njihovo poređenje, sortiranje i intuitivnu pretragu. Cilj aplikacije je rešavanje problema fragmentacije informacija o košarkašima, nudeći korisnicima centralizovan pristup statističkim podacima, biografijama, vestima i zanimljivostima. Sistem je implementiran kroz tri funkcionalne celine: korisnički interfejs razvijen uz pomoć TypeScript-a i React-a, serverski deo baziran na Javi i Spring Boot-u, te webcrawler komponente izrađene u Python-u koristeći biblioteke Selenium, BeautifulSoup i newspaper4k. Rad obuhvata proces prikupljanja i agregacije podataka sa različitih izvora, kao i njihovu prezentaciju kroz intuitivni grafički korisnički interfejs. Implementirana aplikacija omogućava efikasnu pretragu i analizu podataka, značajno unapređujući iskustvo korisniuka pri pristupu informacijama iz sveta košarke.*

Ključne reči: *Web, SpringBoot, React, Webcrawler, AWS, Evroliga*

Abstract – *This paper presents a system in the form of a web application that consolidates statistical data on EuroLeague players, enabling their comparison, sorting, and intuitive search. The goal of the application is to address the issue of fragmented basketball information, providing users with centralized access to statistical data, biographies, news, and highlights. The system is implemented through three functional components: a user interface developed using TypeScript and React, a server-side application based on Java and Spring Boot, and a web crawler component built in Python utilizing libraries such as Selenium, BeautifulSoup, and newspaper4k. The paper covers the process of data collection and aggregation from various sources, as well as their presentation through an intuitive graphical user interface. The implemented application facilitates efficient data search and analysis, significantly enhancing the user experience in accessing basketball-related information.*

Keywords: *Web, SpringBoot, React, Webcrawler, AWS, Euroleague*

NAPOMENA: Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Markoski, red. prof.

1. UVOD

Porast popularnosti košarkaških takmičenja, posebno na evropskom tlu, praćena značajnim tehnološkim razvojem u oblasti digitalnih tehnologija, dovela je do naglog povećanja količine i dostupnosti informacija iz sveta košarke. Kako za prosečne ljubitelje sporta, tako i za košarkaške stručnjake, pristup informacijama o igračima, timovima, statistici, izveštajima sa utakmica i vestima postao je omogućen kroz širok spektar digitalnih medija, uključujući i specijalizovane veb sajtove, aplikacije i društvene mreže.

Međutim, fragmentacija ovih informacija predstavlja veliki izazov pri njihovom pretraživanju. Korisnici, u potrazi za detaljnim podacima o košarkašima, često su primorani da posete veliki broj različitih specijalizovanih sajtova. Tokom čega nailaze na probleme poput prenatrpanosti nerelevantnim informacijama, reklamama i iskačućim prozorima, koji značajno otežavaju i usporavaju proces prikupljanja potrebnih informacija. Ovakav način pretrage, iako daje rezultate, često je veoma vremenski zahtevan i neefikasan.

Cilj ovog rada je da ponudi rešenje navedenog problema kroz razvoj veb aplikacije koja na jednom mestu objedinuje statističke podatke o igračima Evrolige, omogućava njihovo poređenje, sortiranje, kao i intuitivnu pretragu. Pored statistike, sistem takođe omogućava korisnicima pristup relevantnim člancima o igračima, njihovim biografijama, karijerama i zanimljivostima, olakšavajući korisniku pretragu isticanjem ključnih delova relevantnih za njihov upit u okviru grafičkog korisničkog interejsa.

Aplikativno rešenje se sastoji iz 3 funkcionalne celine. Korisničkog interfejsa implementiranog uz pomoć programskog jezika *Javascript* [1] odnosno njegove tipizirane varijacije *Typscript* i radnog okvira *React* [2]. Centralnog serverskog dela aplikacije zaduženog za agregaciju i perzistenciju podataka, kao i za akviziciju inicijalnog seta statističkih podataka i orkestraciju celokoupnog sistema, implementiranog u programskom jeziku *Java* [3] i radnom okviru *Spring Boot* [4]. Webcrawler celine implementirane u programskom jeziku *Python* [5] uz pomoć biblioteka *Selenium*, *BeautifulSoap* i *newspaper4k* i radnog okvira *Flask* [6], koja na osnovu orkestriranih zahteva prikuplja relevantne informacije sa svetske internet mreže.

2. PREGLED SLIČNIH SISTEMA

U ovom poglavlju navedeni su neki od najrelevenatnijih sličnih sistema aktuelnih u trenutku pisanja rada.

Euroleaguebasketball.net [7] je veb aplikacija posvećena evropskoj košarkaškoj ligi, koja pruža sveobuhvatne informacije o timovima, igračima i utakmicama. Korisnicima omogućava pregled statističkih podataka o igračima za poslednje 24 sezone, uključujući biografske podatke, značajna dostignuća i fotografije. Pored toga, aplikacija nudi istorijski pregled članaka, video sadržaja i rasporeda timova, kao i mogućnost kupovine karata za utakmice. Iako funkcionalna i bogata sadržajem, aplikacija ima složen korisnički interfejs i ograničenu tekstualnu pretragu, što otežava brzo pronalaženje željenih informacija.

Eurohoops.net [8] je veb aplikacija specijalizovana za praćenje aktuelnosti u evropskoj košarci. Fokusirana je na vesti, analize utakmica, transfera igrača i tabele takmičenja, ali pruža i istorijski pregled ključnih trenutaka u evropskoj košarci. Aplikacija sadrži intervjue, video sadržaje i kolumne košarkaških analitičara. Njena najveća prednost je intuitivan dizajn koji omogućava jednostavnu navigaciju i pretragu, dok je najveći nedostatak nepouzdana tekstualna pretraga, koja često ne daje relevantne rezultate i otežava lociranje specifičnih informacija.

Proballers.com [9] je aplikacija koja pokriva statističke podatke iz brojnih profesionalnih košarkaških liga, uključujući i manje popularne lige i mlađe kategorije. Omogućava pregled statistike igrača i timova po sezonama, sa širokim opsegom liga koje obuhvata, uključujući čak i manje poznate takmičarske nivoe kao što su druge lige Portugala, Švajcarske i Norveške. Međutim, njen glavni nedostatak je izostanak slobodne tekstualne pretrage – korisnici mogu birati samo unapred definisane opcije, što graničava fleksibilnost pretrage.

3. KORIŠĆENE SOFTVERSKE TEHNOLOGIJE

U ovom poglavlju navedene su tehnologije korišćene pri implementaciji ovog rada, a to su:

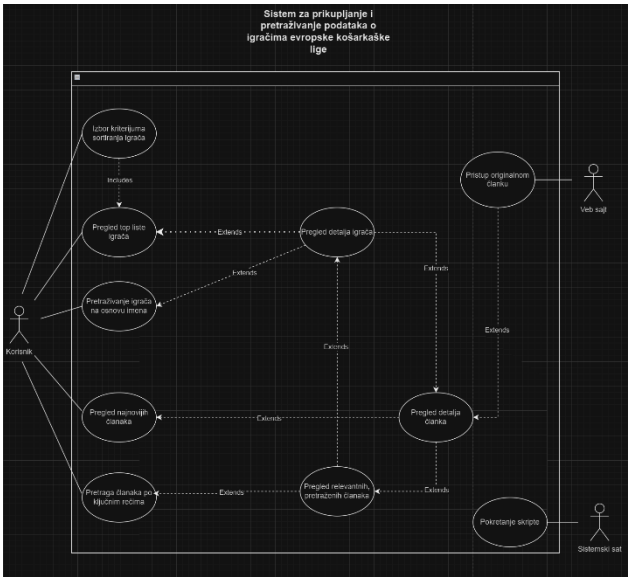
- Java – objektno orijentisani jezik višeg nivoa pogodan za implementaciju serverskih delova
- Spring – radni okvir za razvoj u programskom jeziku *Java*, nudi bogat opus modula pogodnih za brz i robustan razvoj. Neki od korišćenih modula u ovom rešenju: *Spring Boot*, *Spring Data JPA*, *Spring Data ElasticSearch*...
- Docker – alat koji omogućava automatizaciju razvoja korišćenjem tehnologije kontejnera
- LocalStack – softverski alat koji omogućava emulaciju *AWS Cloud*-a (specifično u ovom rešenju *AWS SQS*-a)
- ElasticSearch – distribuirani pretraživač i analitički sistem optimizovan za brzinu i relevantnost pri tekstualnoj pretrazi
- PostgreSQL – besplatan sistem otvorenog koda za upravljanje relacionim bazama podataka
- Flyway – alat otvorenog koda za upravljanje migracijama baze podataka
- Python – interpretirani, objektno orijentisani jezik višeg nivoa pogodan za implementaciju serverskog dela sistema
- Flask – radni okvir za razvoj u programskom jeziku *Python*
- Selenium – alat za automatizovanje veb pretraživača, pogodan pri *e2e* testiranju i u ovom rešenju korišćenom *webscraping*-u
- BeautifulSoap – *Python* biblioteka za parsiranje *HTML* i *XML* dokumenata
- Newspaper4k – Python biblioteka namenjena prikupljanju i ekstrakciji sadržaja onlajn članaka
- JavaScript – programski jezik visokog nivoa pogodan za implementaciju klijentskih i serverskih delova sistema
- React – biblioteka otvorenog koda za razvoj u programskom jeziku *JavaScript*, pogodna za implementaciju klijentskog dela sistema
- Tanstack Query – alat za rad u *React* aplikacijama koji pojednostavljuje komunikaciju sa serverskom stranom sistema
- MaterialUI – biblioteka gotovih komponenti otvorenog koda za *React*
- TailwindCSS – radni okvir otvorenog koda koji omogućava brzo i efikasno stilizovanje korisničkih interfejsa

4. SPECIFIKACIJA

4.1. Specifikacija zahteva

Zadatak obuhvata izradu sistema u obliku veb aplikacije koja na jednom mestu obuhvata statističke podatke o igračima Evrolige, omogućavajući njihovo poređenje, sortiranje i sažetu intuitivnu pretragu. Cilj aplikacije je rešavanje problema fragmentacije informacija o košarkašima, nudeći korisnicima centralizovan pristup statističkim podacima, biografijama, vestima i zanimljivostima. U ovom poglavlju dat je opis funckionalnih i nefunkcionalnih zahteva koje sistem podržava.

Na slici 4 prikazan je *UML* (*Unified Modeling Language*) dijagram slučajeva korišćenja koje sistem podržava



Slika 1. *Dijagram slučajeva korišćenja*

Pregled top liste igrača (evropske košarkaške lige) korisnička je akcija koja zauzima početni deo korisničkog interfejsa i predstavlja inicijalnu tačku interakcije sa korisnikom. Sistem prikazuje top 10 igrača sortiranih po kriterijumu efikasnosti u okviru *Carousel* komponente koja je responzivna i omogućava korisniku klikom na strelice kretanja kroz listu.

Izbor kriterijuma sortiranja igrača korisnička je akcija koja zauzima drugi deo početnog ekrana. Sistem prikazuje dodatnu listu u okviru koje korisnik ima mogućnost biranja kriterijuma sortiranja. Dostupni kriterijumi su formirani na osnovu proseka parametara igrača po utakmici i dostupni parametri su: broj poena, broj minuta, broj asistencija, broj ukradenih lopti, broj izgubljenih lopti, broj skokova.

Pretraživanje igrača na osnovu imena (punog imena) korisnička je akcija pri kojoj sistem u okviru zagljavlja aplikacije prikazuje tekstualno polje za unošenje upita u formi imena odnosno dela punog imena igrača. Pri unosu upita sa određenim kašnjenjem od *300ms* sistem izlistava u okviru padajućeg menija igrače koji zadovoljavaju upit i žutom bojom naznačava deo imena igrača koji se poklapa sa upitom.

Pregled najnovijih članaka predstavlja korisničku akciju koja se nalazi u okviru završnog dela inicijalnog ekrana korisničkog interfejsa. Sistem prikazuje korisniku poslednjih 10 prikupljenih članaka (najsvežijih). Prikazane informacije čine uvod u članak (početnih 250 karaktera) kao i slika igrača o kom je članak napisan. Klikom na članak korisnik biva preusmeren na stranicu detaljnog prikaza članka.

Tekstualna pretraga članka po ključnim rečima korisnička je akcija pri kojoj sistem u završnom delu početnog ekrana prikazuje tekstualno polje za unošenje upita za pretragu članka. Pri pritisku dugmeta *Enter*, sistem prikazuje korisniku listu članaka koji zadovoljavaju upit, žutom bojom označavajući preklapajuće delove članka sa upitom kao i sumarizaciju konteksta oko preklapajućeg dela (250 karaktera).

Pregled detalja igrača korisnička je akcija koju korisnik inicira klikom na sliku igrača iz liste (pretrage ili top liste) odnosno klikom na sliku igrača u okviru detaljnog pregleda članka. U okviru detaljnog pregleda igrača sistem prikazuje naredne informacije o igraču: ime, fotografiju, statistiku, listu članaka o njemu.

Pregled detalja članka akcija je koju inicira korisnik klikom na bilo koji njemu prikazan članak iz liste članaka. Sistem prikazuje korisniku detaljan prikaz sadržaja članka i njegovih meta podataka. Prikazane informacije čine: fotografiju igrača o kom je članak, ime igrača o kom je članak, celokupan tekstualni sadržaj članka, link ka originalnoj lokaciji članka.

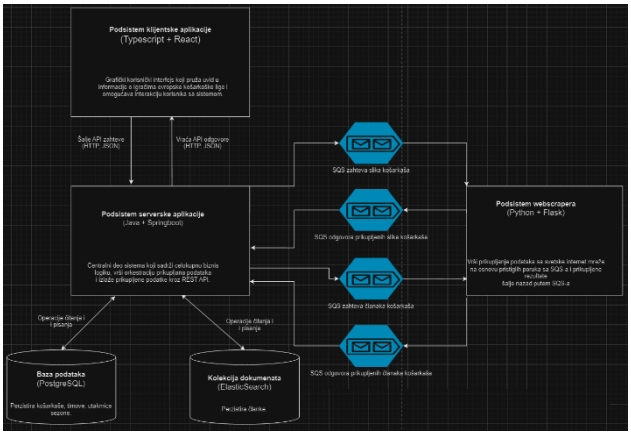
Pristup originalnom članku je akcija koju inicira korisnik klikom na link ka originalnom sadržaju u sklopu detaljnog prikaza članka. Po kliku korisnik biva preusmeren na lokaciju u okviru veb aplikacije sa koje je preuzet originalni članak.

Budući da se radi o veb aplikaciji koja manipuliše velikom količinom podataka, a korisnik očekuje brz odziv bez dugotrajnog čekanja na poslati zahtev, neophodno je obezbediti performantan i robustan sistem, Jedno od rešenja navedenog problema ogleda se u keširanju odgovora serverskog dela sistema (bekenda) u trajanju od 10 minuta, što omogućava sama priroda domena sistema odnosno njegove sprorije promene.

Kritična tačka sistema predstavlja *webscraper*, koji svakom promenom sadržaja odnosno veb stranica koje obrađuje, može dovesti do neželjenih rezultata i potencijalnog usporavanja ili čak rušenja sistema. Navedeni problem rešen je na arhitekturalnom nivou izdvajanjem *webscraper*-a u zaseban servis i neblokirajućom komunikacijom sa njim putem *SQS*-a pri čemu centralni deo serverskog dela aplikacije ne biva usporavan niti u bilo kakvoj rizičnoj zavisnosti od rušenja u slučaju problema ili nedostupnosti *webscraper*-a. Takođe *webscraper* servis ne briše *SQS* poruku tokom čijeg izvršavanja je došlo do greške nego je ponovo uzima u obradu nakon isteka *visibility-timeout*-a, odnosno prekonfigurisanog vremena nakon kog pročitana poruka može ponovo biti preuzeta sa *SQS*-a. Time pružajući pretraživanoj stranici vreme da se oporavi i da se potencijalno izvrši željena pretraga iz ponovnih pokušaja.

4.2. Specifikacija sistema

U ovom poglavlju dat je detaljan uvid u specifikaciju sistema, njegovu arhitekturu i komponente koje ga čine. Na slici 2. prikazan je *C4 UML* dijagram arhitekture sistema visoke apstrakcije u okviru koje je svaki servis modelovan zasebnom komponentom.



Slika 2. *Dijagram arhitekture sistema*

Celokupan sistem za prikupljanje i pretraživanje podataka o igračima evropske košarkaške lige sastoji se iz tri logičke celine, odnosno iz 3 podsistema. Podsistem klijenteske aplikacije, podsistem serverske aplikacije i podsistem *webscrapera*.

Podsistem *webscrapera* predstavlja servis zadužen za prikupljanje podataka sa svetske internet mreže. Putem drajvera interaguje sa veb pretraživačem i pretražuje i prikuplja podatke na osnovu upita pristiglih iz poruka sa *SQS*-a,a zatim prikupljene podatke zapisuje i šalje ih nazad takođe putem poruka preko *SQS*-a.

Podsistem serverske aplikacije čini *REST API* koji izlaže podatke spoljnim sistemima putem *Http*-a, odnosno omogućuje slanje odgovora i zahteva na relaciji klijent-server. Takođe navedeni sistem vrši orkestraciju prikupljanja podataka, izvršavajući jedan deo akvizicije samostalno komunicirajući sa evroliginim javno dostpunim *API*-em. Zatim za svakog pronađenog igrača šalje asinhron zahtev putem *SQS*-a *webscraper*

podsistemu, nakon toga čitajući prikupljene podatke i objedinjeno sa već prikupljenim podacima sa evroliginog *API*-a perzistira ih u *PostgreSQL* i *ElasticSearch* bazama podataka.

Podsistem klijentske aplikacije čini prednji deo sistema (*frontend*) namenjen za direktnu kominikaciju sa korisnikom. Omogućuje korisniku interakciju sa sistemom putem grafičkog korisničkog interfejsa, odnosno omogučuje slanje zahteva serveru i vizualizaciju pristiglih odgovora.

5. ZAKLJUČAK

Rešenje rada predstavlja aplikaciju koja uprošćava proces onlajn pretrage podataka o igračima Evrolige, njihovih statistika, biografija i novinskih članaka. Pri čemu korisnik često biva primoran da posećuje velik broj sajtova različitog sadržaja ne bi li na svakom od njih našao neke od željenih informacija o igraču. Navedni proces pretrage i prelaska sa veb sajta na veb sajt neretko biva ispraćen mnoštvom iskačućih obaveštenja (kolačića, pretplata, akcija..) dodatno produžavajući sam proces pretrage i odvlačeći pažnju korisnika.

Razlika navedenog rešenja u odnosu na prethodno pomenute slične sisteme obrađene u 2. poglavlju ogleda se prvenstveno u obimu informacija koje prikazuje. Ne koncentrišući se na striktno statističke ili striktno biografske podatke, već objedinjujući ih i grupišući ih na nivou igrača time značajno ubrzavajući korisnikovu pretragu i dalje istraživanje. Takođe bitan faktor rešenja, koji ga izdvaja od ostalih sistema, čini tekstualna pretraga sa sumarizacijom. Pri čemu osim same brzine i količine podataka koja je dostupna korisniku, sistem žutom bojom u okviru korisničkog interfejsa označava delove sadržaja koji se preklapaju sa upitom i takođe prikazuje sadržaj koji se nalazi u kontekstu preklapajućeg dela. Čineći sistem veoma intuitivnim i samu pretragu izuzetno brzom i olakšanom.

Potencijalna unapređena proizvoda i samog sistema mogu da se ogledaju u 2 pravca, košarkaško i komercijalno.

Košarkaško unapređenje rešenja moguće je kroz akviziciju dodatnih statističkih podataka i njihovu vizualizaciju. Jedno od potencijalnih unapređenja predstavlja prikaz 2D mape terena u okviru koje bi se videlo sa kojih pozicija igrač promašuje odnosno pogađa šuteve. Mape šuta igrača bi se perzistirale u okviru *ElasticSearch*-ove već podržane geolokacione pretrage te bi top liste igrača osim po dosadašnjim kriterijumima omogućili i po izboru dela terena. Korisnik bi u okviru 2D mape mogao da označi deo terena a zatim šalje upite sistemu koji bi mu prikazali listu igrača koja imaju najviše pogođenih šuteva sa tog dela terena, zatim najviše promašenih šuteva, najbolji odnosno najgori procenat i slično. Takođe sam domen koji je trenutno fokusiran na igrače mogao bi da se prebaci i na timove te bi sistem mogao da omogući pretragu timova, njihova poređenja, sortiranja kao i prikupljanje biografskih podataka i relevantnih članaka o njima.

Komercijalno rešenje bi se ogledalo u proširenju *webscraper* sistema da podrži akviziciju komercijalnijih tipova informacija, odnosno nekoj vrsti *press-clippinga*. Te bi korisnik osim striktno biografskih i košarkaških članaka imao mogućnost pretrage i svih ostalih tipova članaka kao što su članci žute štampe.

Takođe potencijalno unapređenje postojećeg sistema moglo bi da se ogleda i u refaktorisanju njegove arhitekture, odnosno korišćenju i ostalih *AWS* servisa poput *SNS* (*Simple Notification Service*) i *S3* (*Simple Storage Service*). Pri čemu bi poruka poslata od strane serverskog dela sistema ka skrejperskom delu sistema mogla da se centralizuje u vidu *SNS* teme koja bi odašiljala ime igrača te bi iz perspektive serverskog dela bio poslat samo jedan događaj dok bi *SNS* dalje prosleđivao događaj na već postojeće *SQS* servise korišćenjem *fan-out* arhitekturalnog obrasca pri čemu bi se zadržale pozitivne strane *SQS*-a (robusnost, ponovni pokušaji neuspelih akcija, *DLQ*), a smanjila spregnutost sistema sa serverske strane. Takođe mana trenutnog sistema je korišćenje *URL*-ova ka originalnom sajtu koji potencijalno mogu da se promene ili ukinu pri čemu bi korisnik bio pogrešno naveden na stranicu bez željenog sadržaja. Rešenje navedene mane ogledalo bi se u perzistenciji *HTML*-a originalne stranice, pri njenoj akviziciji, u okviru *S3* baketa. Pri čemu u okviru baze podataka ne bi perzistirali *URL* do originalnog sajta već lokaciju originalnog *HTML*-a u okviru *S3* baketa koji bi prikazivali u okviru grafičkog korisničkog interfejsa aplikacije bez nepotrebnog redirektovanja korisnika.

6. LITERATURA

[1] Minnick, C. (2023). JavaScript All-in-One For Dummies. For Dummies.

[2] Kumar, T. (2024). Fluent React: Build Fast, Performant, and Intuitive Web Applications. O'Reilly Media.

[3] Schildt, H. (2024). Java: The Complete Reference, Thirteenth Edition. McGraw Hill.

[4] Miguel Puig, F. (2024). Spring Boot 3.0 Cookbook: Proven recipes for building modern and robust Java web applications with Spring Boot. Packt Publishing.

[5] Matthes, E. (2024). Python Crash Course, 3rd Edition: A Hands-On, Project-Based Introduction to Programming. No Starch Press

[6] Sherwin C. Tragura, J. (2024). Mastering Flask Web and API Development: Build and deploy production-ready Flask apps seamlessly across web, APIs, and mobile platforms. Packt Publishing.

[7] https://www.euroleaguebasketball.net/about/

[8] https://www.eurohoops.net/en/

[9] https://www.proballers.com/about

Kratka biografija:

Aleksić Nikola, rođen je 22.02.2000. godine u Novom Sadu. Školske 2018/19 upisuje se na Fakultet tehničkih nauka, na studijski program Računarstvo i automatika u okviru kojeg završava osnovne akademske studije školske 2021/2022 godine. . Iste godine upisuje master studije na Fakultetu tehničkih nauka, na smer Elektronsko poslovanje.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.41
DOI: https://doi.org/10.24867/33BE02Artukov

SISTEM ZA PRETRAGU SLIKA ZASNOVAN NA CLIP MODELU I VEKTORSKIM BAZAMA

IMAGE SEARCH SYSTEM BASED ON THE CLIP MODEL AND VECTOR DATABASES

Katarina Artukov, *Fakultet tehničkih nauka, Novi Sad*

Oblast – PRIMENJENE RAČUNARSKE NAUKE I INFORMATIKA

Kratak sadržaj – *Ovim radom predstavljen je sistem za pretragu slika zasnovan na CLIP modelu i vektorskim bazama podataka, koji omogućava efikasnu pretragu vizuelnog sadržaja na osnovu tekstualnih ili slikovnih upita.*

Ključne reči: *Pretraga slika, vektorske baze, multimodalna ugnježđenja, mašinsko učenje, CLIP model.*

Abstract – *This thesis presents an image search system based on the CLIP model and vector databases, enabling efficient search of visual content using textual or image-based queries.*

Keywords: *Image search, vector databases, multimodal embeddings, machine learning, CLIP model.*

1. UVOD

U doba kada količina digitalnih slika raste neviđenom brzinom proizilazi potreba za preciznom i efikasnom pretragom vizuelnog sadržaja. Uspešan sistem za pretragu mora biti u stanju da protumači korisnikove upite, zaključi njihove namere, a zatim primeni strategiju pretrage koja će verovatno vratiti relevantne rezultate. Tradicionalni sistemi za pretragu često zavise od tekstualnih oznaka (tagova) ili metapodataka koji se ručno dodaju, što nije održivo rešenje za velike baze podataka i često nije dovoljno tačno u reprezentaciji sadržaja slika.
Razvoj pretraživača slika uz korišćenje savremenih tehnika mašinskog učenja i obrade podataka omogućava nove načine pristupa informacijama i sadržaju, olakšavajući pronalaženje vizuelno sličnih ili konceptualno povezanih slika.

U ovom radu, razvijen je sistem za pretragu slika koji se oslanja na *CLIP* (*Contrastive Language–Image Pretraining*) model za generisanje multimodalnih ugnježđenja [1], koja kombinuju tekstualne i vizuelne karakteristike u jedinstveni prostor vektora. Ugnježđenja se zatim indeksiraju u *Pinecone* vektorsku bazu podataka.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Ivanović, red. prof.

Integracija sa *Pinecone* vektorskom bazom [2] pruža skalabilno i brzo rešenje za pretragu i indeksiranje velikog broja slika. S obzirom na mogućnost indeksiranja multimodalnih ugnježđenja i brzog pretraživanja, ovaj sistem postaje izuzetno koristan u poljima kao što su istraživanje i pronalaženje sličnog vizuelnog sadržaja, marketing, analiza društvenih medija i kreativne industrije. Osim što unapređuje pristup vizuelnim podacima, ovaj sistem takođe predstavlja praktičan primer primene savremenih tehnika mašinskog učenja u realnim okruženjima, čime doprinosi razvoju naprednih rešenja u oblasti obrade i pretrage slika.
U zavisnosti od potreba korisnika, sistem podržava dva modela pretrage. Prvi model omogućava pretragu na osnovu slike, gde se kao ulazni parametar koristi fotografija, za koju će se u rezultatu vratiti deset najsličnijih iz baze. Drugi model omogućava pretragu na osnovu tekstualnog upita, tako što se kao rezultat vraćaju fotografije koje vizuelno reprezentuju sadržaj zadanog teksta.
Sistem je realizovan kao veb aplikacija, pri čemu je pozadinski deo (*backend*) implementiran u *Python FastApi* radnom okviru, dok je korisnički interfejs implementiran u *React*-u. Ovakva arhitektura omogućava upotrebu savremene platforme za pretragu slika na osnovu opisa ili sličnih vizuelnih karakteristika, čime se značajno unapređuje korisničko iskustvo.
Aplikacija je osmišljena da bude intuitivna kako profesionalcima koji se bave analizom vizuelnog sadržaja, tako i krajnjim korisnicima koji žele da lakše i brže dođu do relevantnih vizuelnih informacija.

1.1. Skup podataka

U ovom radu korišćen je *Flickr30k Entities* [3] skup podataka, proširena verzija popularnog *Flickr30k* skupa, standardnog benchmarka za opisivanje slika. Skup obuhvata 31.800 fotografija i 158.915 rečeničnih opisa, organizovanih u fajlu *results.csv* sa kolonama: *image_name*, *comment_number* i *comment*. Nakon eksplorativne analize podataka zaključene su karakteristike skupa podataka. Glavni nalazi uključuju ravnomernu distribuciju komentara po slikama, umeren sentiment i kraće opise u većini komentara. Skup podataka je sam po sebi dobro pripremljen, tako da faza pretprocesiranja ima svega nekoliko dodatnih koraka.

1.2. Radni tok vektorske baze podataka i CLIP modela mašinskog učenja

CLIP model mašinskog učenja predstavlja savremeni multimodalni pristup zasnovan na *Transformer* arhitekturi, sposoban za istovremenu obradu tekstualnih i vizuelnih podataka. Obuka *CLIP* modela obuhvata velike skupove tekstualno-vizuelnih parova, gde model uči zajednički reprezentativni prostor. Ovaj prostor omogućava da slični tekstualni i vizuelni sadržaji budu bliski u vektorskom prostoru, što čini *CLIP* izuzetno korisnim za zadatke kao što su pretraga, klasifikacija i semantičko povezivanje.

Kada se *CLIP* model integriše sa vektorskom bazom podataka, dobija se optimalan radni tok koji kombinuje snage oba sistema. Na početku, *CLIP* model se prethodno obučava kako bi naučio vektorske reprezentacije koje mogu da povežu tekst i slike na semantičkom nivou. Ove vektorske reprezentacije zatim se ugrađuju u vektorsku bazu podataka, koja omogućava efikasno skladištenje i brzo pretraživanje.

Kada korisnik unese tekstualni upit ili dostavi sliku, *CLIP* generiše vektorsku reprezentaciju upita. Vektorska baza zatim koristi napredne algoritme za približnu pretragu najbližih suseda, kao što je *HNSW*, kako bi pronašla slične vektore iz skupa podataka. Rezultati pretrage se vraćaju korisniku, pri čemu *CLIP* osigurava da ovi rezultati odgovaraju semantičkom kontekstu korisničkog upita.

Integracija *CLIP* modela i vektorske baze podataka omogućava značajno poboljšanje u odnosu na upotrebu samo jedne od ovih komponenti. Ako se koristi samo *CLIP*, rezultati mogu biti manje tačni u radu sa velikim skupovima podataka. Sa druge strane, ako se koristi samo vektorska baza podataka, rezultati mogu biti tehnički precizni, ali ne i u potpunosti semantički relevantni za korisnika. Kombinacija ove dve tehnologije omogućava postizanje i preciznosti i prilagođenosti rezultata korisničkim očekivanjima, čime se postiže optimalan balans između efikasnosti i korisničkog iskustva u multimodalnim aplikacijama.

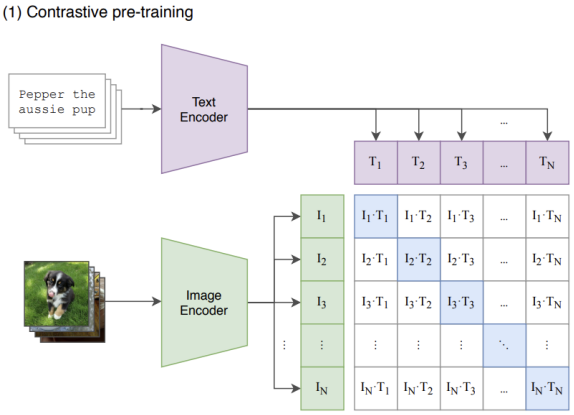
1.3. Kontrastivno pretreniranje

Kontrastivno učenje je metod obuke *AI* modela koji ga uči da razlikuje slične od različitih stvari, koristeći kontrastivni gubitak. Konkretnije, na osnovu skupa od *N* parova (slika, tekst), *CLIP* zajednički trenira enkoder za slike i enkoder za tekst kako bi predvideo koji od *N*×*N* mogućih (slika, tekst) parova u okviru skupa zapravo postoje [4].

Model prima *batch* podataka u obliku parova (slika, tekst). Za svaku sliku u *batch*-u, vizuelni enkoder generiše vektor slike. Prva slika odgovara vektoru *I1*, druga *I2*, itd. Svaki vektor je veličine *de*, gde je *de* dimenzija latentnog prostora, koja predstavlja broj komponenti (dimenzija) vektora koji opisuju sliku ili tekst u zajedničkom embeding prostoru. Veličina *de* je ključna jer određuje kapacitet modela da uči i reprezentuje kompleksne odnose između tekstualnih i slikovnih podataka. Veće dimenzije omogućavaju bogatiju reprezentaciju, ali istovremeno zahtevaju više resursa za obradu i memoriju. Izlaz ovog koraka je matrica *N*×*de*. Slično, tekstualni opisi se konvertuju u tekstualne embedinge *{T1, T2, T3… Tn}*, čime se dobija matrica *N*×*de*. Nakon toga, ove matrice se međusobno množe, izračunavajući kosinusne sličnosti između svake slike i teksta, što rezultira *N*×*N* matricom.

CLIP uči multimodalni ugnježđeni prostor tako što zajednički trenira enkoder za slike i enkoder za tekst kako bi maksimizirao kosinusnu sličnost ugnježđenja slike i teksta za *N* stvarnih parova u skupu, istovremeno minimizujući kosinusnu sličnost ugnježđenja za $N \times N - N$ pogrešnih parova.

Na slici 1.3.1, to znači da želimo da maksimiziramo dijagonalu u matrici, dok minimiziramo sve ostale elemente.



Slika 1. *Kontrastivno pretreniranje*

Za svako ulazno slikovno ugnježđenje vrši se klasifikacioni zadatak u pravcu *x* ose, koji nam govori koji od tekstualnih ugnježđenja *{T1, T2, T3… Tn}* najviše odgovara toj slici. Takođe, za svako tekstualno ugnježđenje vrši se klasifikacioni zadatak u pravcu *y* ose koji govori koje od slikovnih ugnježđenja *{I1, I2, I3...In}* ga najbolje reprezentuje. Na primer, slika *I1* se opisuje tekstom *T1*, a ne tekstovima *T2, T3* itd.

Gubitak koji se ovde koristi je *cross entropy* gubitak, izračunat između logita i labela u *x* pravcu i *y* pravcu, a zatim se računa njihova prosečna vrednost. To znači da ova vrsta gubitka minimizuje greške u oba pravca – od slike ka tekstu i od teksta ka slici. Suštinski, sa kontrastivnom tehnikom, *CLIP* je treniran da razume da slična predstavljanja treba da budu blizu u latentnom prostoru, dok različita treba da budu udaljena.

1.4. *Pinecone*

Pinecone [2] je specijalizovana baza podataka za upravljanje vektorima, dizajnirana da olakša rad sa velikim skupovima podataka i složenim zadacima pretrage u domenu veštačke inteligencije i mašinskog učenja. Zasnovana je na konceptu vektorske pretrage i omogućava korisnicima da efikasno organizuju, indeksiraju i pretražuju podatke visoke dimenzionalnosti.

Jedna od ključnih prednosti *Pinecone*-a je njegova sposobnost da precizno i brzo vrši pretragu sličnosti u velikim skupovima podataka. Ovo se postiže korišćenjem sofisticiranih algoritama za vektorsku pretragu, koji omogućavaju brzo pronalaženje najrelevantnijih rezultata na osnovu sličnosti ugrađenih vektora.

1.5. Hijerarhijski navigabilan mali svet (*HNSW*)

Hijerarhijski navigabilan mali svet [5] je savremena tehnika za pronalaženje približnih najbližih suseda datog vektora u velikom skupu vektora. Ona funkcioniše tako što gradi graf koji povezuje vektore na osnovu njihove

sličnosti ili udaljenosti, a zatim koristi strategiju pretrage zasnovanu na pohlepnom pristupu za kretanje kroz graf i pronalaženje najsličnijih vektora.
Algoritam takođe gradi hijerarhijsku strukturu grafa, gde se svakoj tački dodeljuje različit sloj sa određenom verovatnoćom. Viši slojevi sadrže manji broj tačaka i duže ivice, dok niži slojevi sadrže veći broj tačaka i kraće ivice. Najviši sloj sadrži samo jednu tačku, koja predstavlja ulaznu tačku za pretragu.
Hijerarhijska struktura omogućava efikasnu i preciznu pretragu, počevši od najvišeg sloja i postepeno prelazeći na niže slojeve, pri čemu se u svakom koraku bira najbliži čvor kao sledeća tačka pretrage.
Hijerarhijska struktura koju koristi *HNSW* omogućava brzu i preciznu pretragu, jer se pretraga započinje od najvišeg sloja i postepeno se spušta ka nižim slojevima, pri čemu se u svakom koraku bira najbliži čvor kao sledeća tačka pretrage.
Performanse HNSW algoritma zavise od nekoliko faktora, kao što su dimenzionalnost vektora, broj slojeva, broj suseda po čvoru i broj koraka pretrage. Ovi faktori utiču na balans između tačnosti i efikasnosti, kao i na složenost i skalabilnost algoritma.

2. SPECIFIKACIJA SISTEMA ZA PRETRAGU SLIKA

Sistem za pretragu slika koristi arhitekturu koja objedinjuje više komponenti i tehnologija za omogućavanje efikasnog indeksiranja i pretrage. Kao što je prikazano na dijagramu sa slike 2, ova arhitektura je zasnovana na integraciji klijentske aplikacije, *CLIP* modela i vektorske baze podataka sa sistemom za pretragu.

Slika 2. *Arhitektura sistema*

Klijentska aplikacija: Razvijena u React-u, ova komponenta pruža korisnički interfejs koji omogućava interaktivnu pretragu i prikaz rezultata. Korisnik može pretraživati slike na osnovu teksta ili druge slike, a aplikacija šalje zahteve sistemu za pretragu.
Sistem za pretragu: Glavni deo sistema, implementiran u *Python*-u, vrši osnovne funkcije za pretragu i obradu podataka. On komunicira sa *CLIP* modelom i vektorskom bazom kako bi kreirao i upravljao embedinzima za slike i tekst.
CLIP model: Ovaj model, razvijen od strane *OpenAI*-a, služi za kreiranje multimodalnih embedinga koji predstavljaju semantički sadržaj slika i tekstova. *CLIP* model generiše embedinge koji se kasnije upoređuju sa embedinzima u vektorskoj bazi kako bi se našle slike koje su najsličnije zadanom upitu.
Vektorska baza (Pinecone): *Pinecone* služi kao baza ugnježđenja, gde se embedinzi slika i tekstova čuvaju u vektorskom formatu radi efikasnog pretraživanja. Kada korisnik postavi upit, sistem upoređuje embedinge upita sa embedinzima u *Pinecone* bazi kako bi vratio najsličnije srezultate.
Ovakva arhitektura omogućava fleksibilno pretraživanje slika, bilo na osnovu tekstualnih upita ili sličnih slika, sa velikom tačnošću i brzinom.

3. EKSPERIMENT

Ovim poglavljem definisaćemo izabran način za evaluaciju implementiranog sistema i objasniti razlog za izbor istog.
U sistemima za pretragu informacija, kao što su pretrage teksta, slika ili multimodalnih podataka, metrike poput top-N tačnosti igraju ključnu ulogu u proceni kvaliteta sistema. Top-N tačnost predstavlja procentualni udeo upita gde je relevantni rezultat (tj. dokument, slika ili drugi objekat) pronađen među prvih N rezultata rangiranih od strane sistema.
U informacionim sistemima, korisnici očekuju da relevantni rezultati budu rangirani među prvima. Visoka top-1 tačnost pokazuje sposobnost sistema da konzistentno vraća najrelevantniji rezultat kao prvi, dok visoke vrednosti top-3, top-5 ili top-10 tačnosti ukazuju na to da sistem efikasno identifikuje većinu relevantnih rezultata, iako oni možda nisu uvek rangirani na samom vrhu.
Zbog velikih količina podataka u našem sistemu (oko 31.800 slika), korisnička evaluacija bila je nepraktična, jer bi ispitanici morali da procenjuju rezultate bez uvida u ceo skup slika. Zbog toga smo se odlučili za automatsko testiranje metrikom top-N tačnosti, koja omogućava objektivnu procenu na reprezentativnom uzorku od 500 nasumično izabranih slika.

4. REZULTATI I TUMAČENJA

Dobijeni rezultati prikazuju procente tačnosti sličnosti slika i tekstova. Vrednosti po sve četiri metrike date su narednom tabelom

Tabela 1. *Rezultati tačnosti CLIP modela u predviđanju sličnosti slika i tekstova*

Metrika tačnosti	Procenat tačnosti
Top-1 tačnost	61%
Top-3 tačnost	75%
Top-5 tačnost	85%
Top-10 tačnost	94%

Ovi rezultati ukazuju da model često greši u identifikaciji najboljeg podudaranja, ali sa većim N postiže znatno bolju tačnost. Niska top-1 tačnost od 61% ukazuje na ograničenja modela u direktnoj tekstualnoj pretrazi. Ipak, visoka tačnost u top-10 (94%) čini ga veoma pogodnim za inicijalnu fazu pretrage, gde se potencijalno relevantni rezultati mogu rangirati za dalju obradu.
CLIP model pokazuje veliki potencijal kao osnova za sistem pretrage, uz mogućnost poboljšanja kroz dodatno podešavanje parametara, izborom drugih arhitektura enkodera ili integracijom sa specifičnim algoritmima za fino rangiranje.

5. ZAKLJUČAK

U ovom radu predstavljen je sistem za pretragu slika zasnovan na CLIP modelu i vektorskim bazama podataka, koji omogućava efikasnu pretragu vizuelnog sadržaja na osnovu tekstualnih ili slikovnih upita. Razmatrajući izazove savremenih sistema za pretragu, kao što su ograničenja tekstualnih oznaka i potreba za skalabilnošću, implementiran je pristup koji kombinuje multimodalne embedinge i vektorske baze kako bi se obezbedila tačnost i brzina u obradi velikog broja podataka.

Sistem je omogućio dve vrste pretraga – na osnovu teksta i na osnovu slike – pri čemu je postignuta visoka tačnost u identifikaciji relevantnih rezultata, naročito u top-10 metrici (94%). Iako top-1 tačnost od 61% ukazuje na potrebu za dodatnim poboljšanjima, rezultati su u skladu sa očekivanjima za primenu modela u složenim i velikim skupovima podataka.

Naučni doprinos rada ogleda se u uspešnoj primeni *CLIP* modela za rešavanje stvarnih problema pretrage vizuelnog sadržaja. Evaluacija sistema pruža vredne uvide u mogućnosti i ograničenja trenutnog pristupa, dok sposobnost modela da generalizuje preko različitih zadataka bez dodatnog finog podešavanja potvrđuje njegovu robusnost i svestranost.

5.1. Dalji razvoj sistema

Buduća istraživanja mogu se usmeriti na unapređenje tačnosti kroz primenu naprednih tehnika, uključujući različite arhitekture enkodera i pažljiv odabir parametara modela u skladu sa specifičnim zahtevima domena primene. Proširenje sistema detaljnijim pretprocesiranjem slika takođe bi moglo doprineti poboljšanju performansi.

Trenutno implementirani sistem čuva slike sa metapodacima u indeks vektorske baze, pri čemu metapodaci obuhvataju isključivo nazive slika. Unapređenje sistema dodavanjem opisa slike kao dela metapodataka značajno bi poboljšalo korisnost i funkcionalnost. Ovo proširenje bi, pored bolje pretrage, omogućilo i primenu naprednih tehnika semantičkog i hibridnog pretraživanja koje podržava *Pinecone*, čime bi se dodatno unapredile mogućnosti povezivanja i identifikacije relevantnih sadržaja.

Dodatno, integracija *reranker*-a u postojeći sistem mogla bi značajno poboljšati redosled prikazanih rezultata, naročito u slučajevima gde je potrebno preciznije rangiranje u okviru top rezultata. *Reranker* bi omogućio finu obradu i prilagođavanje rezultata specifičnim kontekstima pretrage, što bi ujedno podiglo nivo tačnosti sistema i korisničko iskustvo.

Zaključno, sistem razvijen u ovom radu predstavlja značajan korak napred u oblasti pretrage vizuelnog sadržaja, objedinjujući savremene tehnike mašinskog učenja i vektorske baze podataka. Njegova primena i rezultati pokazuju potencijal za široku upotrebu u različitim domenima, dok istovremeno postavljaju osnovu za budući razvoj i istraživanje u ovoj oblasti.

6. LITERATURA

[1] openai.com, „https://openai.com/index/clip/ “ [Na mreži].

[2] pinecone.io, „https://www.pinecone.io/“ [Na mreži].

[3] J. R, „kaggle.com,“ [Na mreži]. Available: https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset/data

[4] Radford, Alec, et al. "Learning transferable visual models from natural language supervision." International conference on machine learning. PMLR, 2021.

[5] Y. A. Malkov and D. A. Yashunin, “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs,” IEEE transactions on pattern analysis and machine intelligence, vol. 42, no. 4, p. 824–836, 2018.

Kratka biografija:

Katarina Artukov rođena je 12.06..2000. godine u Sremskoj Mitrovici. Školske 2019/20 godine upisuje se na Fakultet tehničkih nauka na studijski program Računarstvo i automatika, a 2022. godine upisuje usmerenje Primenjene računarske nauke i informatika. Osnovne akademske studije završila je u septembru 2023. godine, nakon čega je u istoj školskoj godini upisala master akademske studije.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38
DOI: https://doi.org/10.24867/33BE03Stamenkovic

FORMALNA VERIFIKACIJA DVOJEZGARNOG JEDNO-CIKLUSNOG RISC-V PROCESORA I DELA MEMORIJSKOG PODSISTEMA

FORMAL VERIFICATION OF A DUAL CORE SINGLE-CYCLE RISC-V CORE WITH PART OF MEMORY SUBSYSTEM

Petar Stamenković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu je prikazan način verifikacije dvojezgarnog jedno-ciklusnog RISC-V procesora i dela memorijskog podsistema pomoću formalnih metoda i JasperGold alata kreiranog od strane kompanije Cadence. Dat je osnovni opis dizajna koji se verifikuje, osnovni operatori SystemVerilog jezika koji se koristi kao i verifikacione tehnike koje su upotrebljene za efikasniju verifikaciju sistema.*

Ključne reči: *Formalna verifikacija, RISC-V, Memorijski podsistem, Pokrivenost dizajna, Keš memorija*

Abstract – *This paper presents the way that dual core single-cycle RISC-V processor with part of memory subsystem is verified with formal methods by using a formal tool JasperGold developed by Cadence. Basic description of design is given, alongside with basic SystemVerilog operators and verification tecniques that were used for efficent verification of the system.*

Keywords: *Formal verification, RISC-V, Memory subsystem, Design coverage, Cache memory*

1. UVOD

Formalna verifikacija podrazumeva upotrebu alata koji matematički analiziraju dostižna stanja u dizajnu umesto da proračunavaju vrednosti za konkretno zadate ulazne vektore. Danas predstavlja veoma efikasan i temeljit način verifikacije jer teoretski može da potvrdi totalno odsustvo greški, uz idealno napisane osobine. Alat pruža ogroman broj mogućnosti i veliki broj aplikacija koje olakšavaju proveru određenih aspekata sistema kao što su provera pokrivesnoti (*Coverage app*), provera validnosti napisanih osobina (*FPV*), provera ekvivalentsnoti sistema (*SEQ*) i tako dalje. Formalna verifikacija sa sobom nosi neke veoma važne prednosti u odnosu na verifikaciju baziranu na simulaciji i neke od njih su:

1. Za jednostavne *RTL* modele nema potrebe za pisanjem test okruženja.
2. Lako ispitivanje nama nepoznatog sistema.
3. Efikasnije i uglavnom kraće vreme proveravanja.
4. Omogućena je potpuna pokrivenost
5. Dostupnost kontra-primera
6. Dostupnost analize beskonačnih putanja

U radu je najviše korišćena *FPV* aplikacija koja radi sa osobinama. Osobina (*property*) je jezička konstrukcija koja formalno opisuje neki aspekt digitalnog sistema i predstavlja uopštenje već poznate *assert* naredbe. Definišu se pomoću *HDL* jezika od kojih je u ovom radu korišćen *SystemVerilog*. Na njih se primenjuju verifikacione akcije *assert, assume* ili *cover*.

2. KRATAK OPIS PROJEKTOVANOG DIZAJNA

Kao što je pomenuto, projektovani dizajn je RISC-V sistem i on se sastoji od sledećih komponenti :

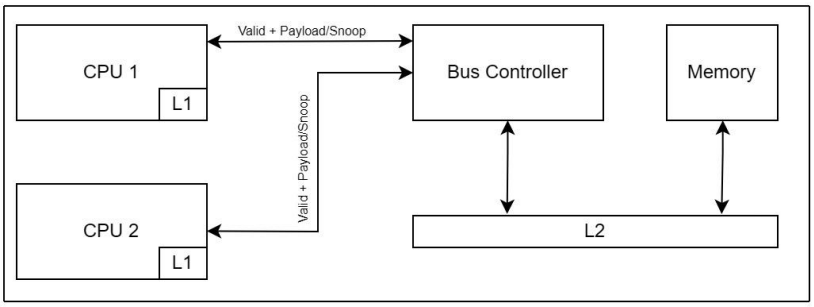
- Dva RISC-V jedno-ciklusna procesora sa sopstvenim L1 keš memorijama.
- Globalna magistrala sa kontrolerom.
- Deljena L2 keš memorija sa keš kontrolerom.
- Globalna memorija za podatke.

Blok šema sistema je data na slici 1 na sledećoj stranici. Projektovanje sistema je rađeno u 3 faze, gde je prva faza podrazumevala procesor koji sadrži i keš memoriju i *data memory* komponentu za podatke. U nastavku je ustanovljeno da to nije optimalno, pa je u fazi 2 *data memory* komponenta izbačena van procesora. Konačno faza 3 sadrži ceo sistem i njegovo povezivanje. Svaku fazu je pratila jednostavna verifikacija i pisanje kratkih test scenarija za simulaciju u okviru alata *Vivado*.

2.1. RISC-V JEDNO-CIKLUSNI PROCESOR

Srce ovog sistema predstavlja upravo pomenuti RISC-V procesor. Inicijalno, procesor je sadržao osnovne komponente RISC-V arhitekture a zatim je po fazama dodato ostalo. Procesor podržava instrukcije tipova R, I ,L, S, U, B i J. Sadrži sopstvenu L1 memoju koja je direktno mapirana i ima 256 lokacija i uz nju je i keš kontroler za istu.

NAPOMENA: Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vuk Vranjković, vanr. prof.



Slika 1 **:** *Blok šema projektovanog sistema*

Protkol koherenntnosti koji je odabran za ovaj rad je MESI i on je takođe implementiran u okviru L1 keš memorije. Suštinski, ako se desi pogodak (*cache hit*) u L1 keš memoriji procesora koji traži podatak, on se jednostavno upisuje u registar. Ako se desi promašaj (*cache miss*), prvo proveravamo dostupnost podatka u drugom procesoru, zatim u L2 keš memoriji i na samom kraju u globalnoj memoriji za podatke koja predstavlja poslednji nivo memorijske hijerarhije. Procesor se sastoji od sledećih komponenti: aritmetičko-logička jedinica (*ALU*), instrukciona memorija, sabirač (*sa 4 i sa poljem imm*), modul za grananje (*branch condition*), kontroler samog procesora (*kontorlni signali*), modul za generisanje konstante (*immediate generator*), programski brojač(*PC*), multiplekseri, registarski fajl, modul za prosleđivanje podatka koji se upisuje u registarski fajl (*write back*) i L1 keš memorija sa svojim kontrolerom.

Bitno je napomenuti da se magistrali pristupa samo u slučaju L ili S tipa instrukcija i da u slučaju istovremenog pristupa oba procesora postoji implementriana arbitraža koja odlučuje koji će imati prednost u tom momentu. Inicijalno, prednost ima prvi procesor. Upis u sve memorije kao i registarski fajl se izvršava na opadajuću ivicu taktnog signala dok je čitanje iz svih asinhrono.

2.2. Ostatak sistema

Procesor je instanciran dva puta i oba su povezana na zajedničku magistalu koja za cilj ima efikasno rutiranje i prosleđivanje podataka, bilo to od strane drugog procesora ili L2 memorije. Magistrala dobija informacije kao što su adresa, operacija, tag kao i sam podatak i pomoću istih odlučuje gde se šta šalje. Takođe, poseduje indikatore o tome gde se podatak može naći, što delom predstavlja *snoop* protokol. Ukoliko jezgro 1 nema podatak, magistrala će proveriti da li ga ima drugo jezgro i tek u slučaju da ga nema će se osvrnuti na ostatak sistema.
Deljena L2 keš memorija je set-asocijativna sa dva kanala i ima 1024 lokacije koje su podeljene u 512 setova (*4 puta veći kapacitet od L1*). Algoritam zamene podataka unutar kanala koji se koristi je LRU i implementiran je pomoću LRU bita koji je sastavni deo svake linije u L2 memoriji. Od trenutka kada su oba podatka validna unutar jednog seta u memoriji, LRU biti tih kanala moraju uvek biti suprotni.
Globalna memorija za podatke predstavlja poslednji nivo memorijske hijerarhije i nakon odluke u fazi 2, izbačena je van procesora. Njoj pristupamo u slučaju da podatak nije dostupan niti u jednom od procesora niti u L2 keš memoriji. Tokom rada, njen kapacitet je bio 1024 kako bi se alatu olakšala provera osobina, međutim nakon iste, moguće je povećati njen kapacitet.

3. PROCES VERIFIKACIJE PROJEKTOVANOG SISTEMA

Slično procesu projektovanja, i proces verifikacije je rađen u 3 faze. Verifikacija procesora koji je imao i keš memoriju i *data memory* komponentu predstavalja verifikaciju faze 1 dok je verifikacija finalne verzije procesora bez *data memory* komponente predstavljena sa verifikacijom faze 2. Verifikacija celokupno povezanog sistema je verifikacija faze 3. Kao što je pomenuto, pre same formalne verifikacije pisani su jednostavni testovi za test okruženje u okviru alata *Vivado* gde je proverena osnovna funkcionalnost sistema za par smišljenih ulaznih kombinacija instrukcija.

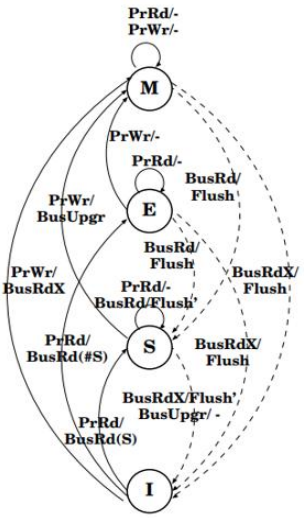
3.1. Verifikacioni plan

Verifikacioni plan predstavlja prvi korak u svakoj verifikaciji i predstavlja bitan dokument jer se upravo na njega ceo verifikacioni tim oslanja. Sastavlja se na osnovu funkcionalne specifikacije sistema i preporuka je pisati ga „hladne glave“ kako bi rasterećeni obuhvatili sve bitne tačke za proveru sistema.
Faza 1 i faza 2 su pokrivene istim verifikacionim planom obzirom da obe obuhvataju proveru procesora uz modifikaciju pomenute *data memory* komponente. Radi se provera svih komponenti unutar procesora kao i proveri tranzicija mašina stanja i MESI protkola koherentnosti.
Faza 3 je pokrivena drugim verifikacionim planom i on podrazumeva proveru interakcija između svih komponenti sistema kao što su dešavanje nakon *flush* operacije, provera oba kanala u jednom setu L2 memorije, LRU biti i slično. Takođe se proverava i upis i čitanje kako iz L2 keš memorije, tako i iz globalne memorije za podatke.

3.2. Verifikaciono okruženje

Za razvijanje verifikacionog okruženja je korišćena metoda referentog modela, to jest, napisana je *checker* komponenta. Za povezivanje ove komponente i samog dizajna se koristila *bind* komanda i napisana skripta. Slično kao i za verifikacioni plan, i u ovom slučaju su napisana dva referentna modela, jedan za verifikaciju procesora i jedan za verifikaciju celokupnog sistema. U ovim fajlovima su

pisane osobine koje za ulogu imaju proveru funkcionalnosti sistema. Oba referentna modela imaju sličnu strukturu i ona se sastoji od sledećih delova :



Slika 2 **:** *MESI protkol*[2]

- Deklaracija portova dizajna procesora odnosno sistema
- Sekcija za definisanje pomoćnih signala, parametara i struktura.
- Sekcija za definisanje restrikcija i ograničenja (*assumes*)
- Sekcija za definisanje tzv. *grey box* signala koji idu kroz hijerarhiju i uzimaju se direktno iz dizajna.
- Pomoćni kod (*Auxillary code*)
- Sekcija za definisanje tvrdnji i pomoćnih tačaka pokrivenosti (*asserts, covers*)

Restrikcije su veoma bitan deo referentnog modela jer ograničavaju vrednosti ulaza koje alat može da kreira. Osobina koju alat dokaže, uz restrikcije koje nisu dobro napisane, ne može se smatrati validnom jer smo time možda maskirali grešku. Na primer, RISC-V arhitektura ne dozvoljava da u registru x0 bude bilo koja vrednost sem 0. Ukoliko ne napišemo restrikciju koja osigurava da alat ne kreira takvu instrukciju koja će da radi upis u taj registar, naša osobina za upis u registarski fajl neće biti korektna. Još jedan primer su vrednosti maske za S ili L tip instrukcije. Maska u ovim slučajevima određuje koji deo reči od 32 bita će biti upisan ili učitan, byte (*8 bita*), half word (*16 bita*) ili word (*32 bita*). Po RISC-V standardu, polje za masku je široko 3 bita pa je alatu dostupno 8 vrednosti za istu, iako za S postoje samo 3 vrednosti koje se koriste i za L samo 5 vrednosti. Bez ograničenja, alat bi odmah dao kontra-primer sa jednom od vrednosti koje se ne koriste, što ne bi bilo korektno. Bitno je napomenuti da, obzirom da sistem radi i na rastuću i na opadajuću ivicu takta, svaka restrikcija mora imati svoju kopiju koja se evaluira i na opadajuću ivicu.

3.3. Tehnike formalne verifikacije

Kako bi se alatu olakšao rad i dokaz napisanih osobina, korišćene su razne tehnike. One su pomogle i pri tome da se kod organizuje i time inžinjeru olakša da razume povratnu informaciju alata.

Metoda crne kutije (*black box*) – Unutar instrukcione memorije su učitane instrukcije koje sistem treba da izvrši. Kako bi verifikacija bila preciznija, na ovu komponentu je primenjena *black box* tehnika. Alat sada ne posmatra unutrašnjost ove komponente i njenu funkcionalnost, već ima slobodu da na njene ulaze dovodi sve moguće vrednosti koje poštuju zadate restrikcije. Obzirom da je ova komponenta jednostavna i ne izvršava neki napredan kod, nije postojalo nikakvih mera opreza za primenu ove tehnike. Nakon primene iste, trebalo je samo napisati propratne restrikcije koje će da osiguraju korektan rad sistema. Preciznije, donjih 7 bita izlazne instrukcije je moralo da ima jednu od 7 vrednosti za svaki od podržanih tipova instrukcije.

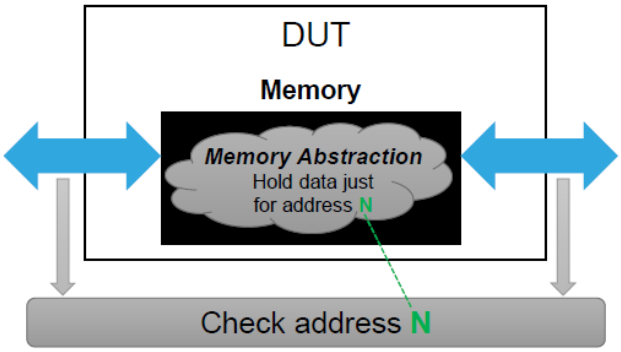
Pisanje pomoćnog AUX koda – Kod kompleksnijih dizajnova kao što je ovaj, osobine mogu biti takođe veoma kompleksne i dugačke, što nikako ne odgovara alatu. Pomoćni kod omogućava da se kreiraju neki pomoćni signali koje kasnije možemo da upotrebimo za poređenje u okviru nekog dokaza. Na primer, za proveru programskog brojača bi na prvi pogled imali 7 provera za svaki tip instrukcije. Međutim ukoliko samo napišemo jednostavni pomoćni kod i kreiramo signal *pc_ref* koji prati promenu vrednosti brojača u referentnom modelu, kao što je urađeno u ovom projektu, dovoljna je samo jedna osobina koja dokazuje validan rad programskog brojača. U nekom drugom slučaju je bolje jednu osobinu rastaviti na više manjih osobina. Recimo da želimo da proverimo skladištenje podataka u L1 keš memoriju. Postoje 3 vrednosti maske za S tip instrukcije tj. možemo skladištiti 1, 2 ili 4 bajta. Teoretski je moguće napisati jednu osobinu koja će proveriti sve maske odjednom, ali obzirom da je skladištenje podataka kompleksna operacija za alat (*jer radi sa velikim brojem stanja*), mnogo je pametnije i efikasnije to uraditi sa 3 odvojene osobine.

Upotreba nedeterminističke konstante (*location based coupling*) – Nedeterministička konstanta ili još poznato kao i slobodna varijabla je varijabla koju alat zadaje tokom provere. Ova tehnika podrazumeva proveru jedne nedeterminističke lokacije i često se koristi upravo za adrese i tagove kod keš memorija. Na ovaj način se apstrahuju svi elementi osim lokacije koja se proverava, čime se drastično smanjuje prostor koji alat mora da ispita. Ovo se često koristi u slučajevima:

- Kada sistem ima dosta simetričnih elemenata kao što su tabele, memorije ili nizovi.
- Kada sistem ima veliki adresni prostor, kao što je DMA kontroler ili memorija.

Uz ovu tehniku je potrebno napisati restrikciju koja drži

ovu varijablu stabilnom, jer ona ne sme da se menja tokom jedne iteracije rada alata. Takođe, ako je potrebno, moguće je ograničiti i vrednost ove varijable. Tehnika se uglavnom primenjuje u osobinama tako što se evaluacija iste radi pri podudaranju adrese koju je doveo alat i pomenute varijable. Grafički prikaz ove tehnike je dat na slici 3.



Slika 3 *: Location based coupling*[3]

Za većinu osobina je korišćeno pisanje pomoćnog AUX koda i za sve kompleksnije osobine (*rad sa memorijama*) je korišćena nedetermisnitčka konstanta. Međutim, postoje i osobine za koje ovo nije bilo potrebno. Ovo su uglavnom jednostavne osobine za koje je dovoljno koristiti dostupne portove i *grey-box* signale kako bi se dokaz napisao. Primer jedne takve osobine je dat ispod ovog pasusa.

```
property checking_transition_from_IDLE_to_DMEM_WRITE;
        top.cache_L2.state == 2'b00 &&
        top.cache_L2.cache_hit_out == 2'b01 |=>
        top.cache_L2.state == 2'b01;
endproperty
```

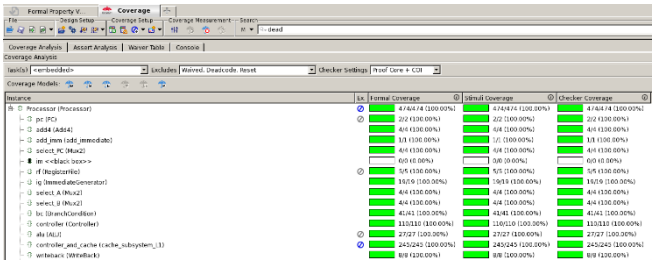
Napisana osobina jednostavno proverava da li se prelazi u stanje DMEM_WRITE u slučaju promašaja u memoriji.
Sve osobine koje su napisane su dokazane i sve pronađene greške u svim fazama su ispravljene i dokumentovane u samom radu.

4. ANALIZA REZULTATA

Nakon što su sve osobine napisane i dokazane, poslednji korak je upotreba *Coverage* aplikacije u okviru alata. Njena uloga je da nam pruži metriku i povratnu informaciju o tome koliko smo dizajna pokrili sa našim osobinama, da li su sva granjanja proverena i slično.
Formalni alat nudi 3 tipa pokrivenosti :

1. *Stimuli coverage* – Koji kod ili funkcionalnost je dostižna pomoćnu formalnog alata? Koje osobine alat može da pokrije?
2. *Checker coverage* – Ova pokrivenost nam daje informaciju o tome da li je verifikacija završena i koliko dizajna je pokriveno pomoću napisanih tvrdnji.
3. *Formal coverage* – Kombinacija rezultata prethodne dve metrike. Tačka u okviru ove pokrivenosti se smatra „pokrivenom“ ako je ista „pokrivena“ i u *Stimuli* i u *Checker* metrici. Ova metrika nam daje informaciju o ukupnoj pokrivenosti sistema.

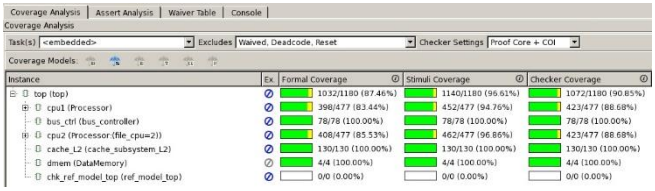
Aplikacija je pokrenuta nakon pisanja oba referetna modela, na kraju faze 2 i na kraju faze 3.
Nakon verifikacije procesora, aplikacija je pokazala maksimalnu pokrivenost (*slika 4*) nakon određenog vremena. Ovime smo dobili zeleno svetlo da nastavimo sa verifikacijom ostatka sistema.



Slika 4 : *Pokrivenost nakon verifikacije procesora*

Nakon uspešne verifikacije celokupnog sistema, aplikacija je pokrenuta i nakon faze 3.
Usled manjka memorijskih resursa na virtuelnoj mašini na kojoj je bilo rađeno, pri kraju rada aplikacije alat je izdao upozorenje zbog kojeg je provera morala biti obustavljena pred kraj. Ovime pokrivenost nije teoretski maksimalna, međutim na slici 5 možemo primetiti sledeće stvari:

- Novo dodate komponente su maksimalno pokrivene, dok je alat ostavio samo već proverene procesore za kraj.
- Većina funkcionalnosti kod procesora je dokazana, zbog čega se pretpostavlja da bi i ostatak bio dokazan sa više dostupnih resursa.
- Nema oblasti u koje alat ne može uopšte da uđe (*crvene oblasti*), već su preostale samo žute oblasti koje su „još uvek nedokazane“ (*undetermined*).



Slika 5 : *Pokrivenost nakon verifikacije sistema*

Uz sve ove tačke kao i osobine koje su dokazane, određeno je da se na ovom mestu verufikacija završi, a time i ovaj projekat.

5. ZAKLJUČAK

Nakon analize rezultata je ustanovljeno da sistem sadrži sve osobine koje su predviđene po specifikaciji. Procesori mogu da izvrše sve instrukcije koje podržavaju sa korektnim rezultatima. Za verifikaciju su uspešno iskorišćene formalne metode i tehnike za smanjenje kompleknosti. Sve pronađene greške su dokumentovane i ispravljene i analiza rezultata pomoću *Coverage* aplikacije je dala dobar rezultat čime zaključujemo da je verifikacija uspešno odrađena.

6. LITERATURA

[1] E. Seligman, T. Schubert, M. V. A. K. Kumar, *An essential toolkit for modern VLSI Design*
[2] T. Suh, *Integration and evaluation of cache coherence protocols for multiprocessor socs,* 2006
[3]"Jasper expert course", Cadence [Na mreži]. Available: https://www.cadence.com/en_US/home/training/all-courses.html

Kratka biografija

Petar Stamenković rođen je u Novom Sadu 2000. godine. Diplomirao je na Fakultetu tehničkih nauka, na smeru Energetika, elektronika i telekomunikacije 2023. godine. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Energetika, elektronika i telekomunikacije odbranio je 2025. godine. Kontakt: petarstamenkovic35@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.42
DOI: https://doi.org/10.24867/33BE06Todic

OBRADA VELIKIH SKUPOVA PODATAKA KORIŠĆENJEM CLOUD TEHNOLOGIJA

PROCESSING LARGE DATA SETS USING CLOUD TECHNOLOGIES

Bosiljka Todić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu analizirani su dnevni valutni kursevi korišćenjem Google BigQuery platforme, koja omogućava obradu velikih skupova podataka i predikciju budućih trendova. Analiza podataka uključuje identifikaciju dugoročnih trendova i volatilnosti valuta, uz korišćenje grafičkih prikaza i prediktivnih modela. Fokus je na upotrebi BigQuery-a kao efikasnog cloud alata, koji nudi skalabilnost i brzinu obrade kompleksnih upita. Cilj rada je pokazati potencijale cloud tehnologija i Big Data alata u analizi i donošenju strateških odluka.*

Ključne reči: *Cloud computing, Big Data, Google Cloud BigQuery, Kaggle, Dnevni valutni kursevi*

Abstract – *This paper analyzes daily currency exchange rates using the Google BigQuery platform, which enables the processing of large datasets and the prediction of future trends. The data analysis focuses on identifying long-term trends and currency volatility, complemented by graphical representations and predictive models. The emphasis is on BigQuery as a robust cloud tool that offers scalability and speed for executing complex queries. The goal is to demonstrate the potential of cloud technologies and Big Data tools in analysis and strategic decision-making.*

Keywords: *Cloud computing, Big Data, Google Cloud BigQuery, Kaggle, Daily exchange rates*

1. UVOD

Razvoj savremenih tehnologija i sve veća količina dostupnih podataka stvorili su potrebu za efikasnim alatima i platformama za obradu velikih količina informacija. Ova potreba dovela je do širenja koncepta Big Data i cloud računarskih platformi. Danas se mnoge industrije, poput finansija, maloprodaje, zdravstva, trgovine valutama, oslanjaju na brze i precizne informacije kako bi donele strateške odluke. Mogućnost analize velikih skupova podataka u realnom vremenu postala je ključna za uspeh u ovim sektorima.

U ovom radu fokus će biti na obradi i analizi velikih skupova podataka korišćenjem cloud servisa. Konkretno, obrađivaće se skup podataka sa platforme Kaggle, koja služi za deljenje i analizu podataka.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Vukmirović, red. prof.

Skup podataka sadrži dnevne promene valutnih kurseva u odnosu na baznu valutu, što pruža priliku za analizu dugoročnih trendova i promenjivosti valuta. Podaci će biti obrađeni, analizirani i grafički prikazani. Koristiće se Google-ova platforma BigQuery koja je serverless skladište podataka sa ugrađenim mogućnostima mašinskog učenja. Platforma ima mnoštvo funkcija koje pomažu u analitici različitih veličina i tipova podataka. Takođe, platforma nudi nekoliko načina za vizualizaciju, kao i kreiranje predikcija pomoću BigQuery ML ugrađenih funkcija. Cilj ovog rada je da prikaže kako cloud tehnologije i Big Data alati mogu biti korisni u analizi velikih skupova podataka, kroz konkretan primer valutnih kurseva.

2. OSNOVNI POJMOVI

U ovom poglavlju će biti detaljno objašnjeni ključni pojmovi koji čine osnovu za dalju razradu teme i analizu podataka u ovom radu. Fokus će biti na definisanju i razumevanju osnovnih principa cloud computinga i big data tehnologija, kao i na njihovoj međusobnoj povezanosti i značaju u savremenim IT sistemima.

2.1. Cloud computing

Cloud computing predstavlja isporuku IT resursa na zahtev putem interneta uz plaćanje prema korišćenju. Umesto kupovine, posedovanja i održavanja fizičkih data centara i servera, moguće je pristupiti tehnološkim uslugama, kao što su računarska snaga, skladištenje i baze podataka, po potrebi, od pružalaca cloud usluga [1].

Tipovi modela implementacije cloud computing-a:

- Javni cloud – u vlasništvu i pod upravom trećih strana, poput pružalaca cloud usluga kao što je Google Cloud. Ova vrsta clouda omogućava kompanijama pristup računarskim, skladišnim i mrežnim resursima putem interneta. Resursi su dostupni na zahtev, što firmama pruža fleksibilnost da ih koriste prema svojim specifičnim potrebama i poslovnim ciljevima, bez potrebe za ulaganjima u sopstvenu IT infrastrukturu.
- Privatni cloud – infrastruktura koju gradi, poseduje i upravlja jedna organizacija, pri čemu se resursi hostuju unutar njenih sopstvenih data centara. Ovaj pristup, poznat i kao „on-premises" rešenje, pruža kompanijama veću kontrolu, sigurnost i upravljanje podacima.

- Hibridni cloud – kombinuju modele javnog i privatnog cloud-a, omogućavajući kompanijama da koriste usluge javnog cloud-a, a istovremeno zadrže mogućnosti koje su uobičajene u arhitekturama privatnog cloud-a.

Postoje tri glavna tipa modela cloud computing usluga koje se mogu izabrati u zavisnosti od nivoa kontrole, fleksibilnosti i upravljanja koje korisnik zahteva:

- Infrastructure as a service (IaaS) – obezbeđuje osnovne resurse, kao što su serveri, skladištenje i mogućnosti umrežavanja, upotrebom virtuelnih mašina, kao što su Amazon EC2, Google Compute Engine i Microsoft Azure.
- Platform as a service (PaaS) – obezbeđuje platformu za razvoj, testiranje i implementaciju softverskih aplikacija, kao što su AWS Elastic Beanstalk, Google App Engine i Heroku.
- Software as a service (SaaS) – model usluge u cloud-u koji omogućava korisnicima pristup softverskim aplikacijama putem interneta, bez potrebe za instalacijom i održavanjem tih aplikacija na svojim lokalnim uređajima ili serverima, kao što su Amazon WorkSpaces, Gmail, Salesforce, Dropbox, Google Apps for Work i Microsoft Office 365.

2.2. Big Data

Ogromne količine podataka prikupljenih tokom vremena koje je teško analizirati i obrađivati korišćenjem uobičajenih alata za upravljanje bazama podataka. Ovi podaci se analiziraju radi identifikacije tržišnih trendova u poslovanju, kao i u oblastima proizvodnje, medicine i nauke. Tipovi podataka uključuju poslovne transakcije, e-poruke, fotografije, snimke nadzornih kamera, evidenciju aktivnosti i nestrukturirani tekstovi sa blogova i društvenih mreža, kao i ogromne količine podataka koje mogu prikupljati senzori različitih vrsta [2].

Big Data se definiše kroz pet ključnih karakteristika poznatih kao 5V. Prva karakteristika je obim (Volume), koja se odnosi na količinu podataka koji se generišu iz različitih izvora i u različitim formatima, često toliko velika da tradicionalne metode skladištenja i obrade nisu dovoljno efikasne. Zatim, tu je vrednost (Value), koja označava korisne uvide koje se mogu izvući iz podataka, a koji dodaju vrednost poslovnim procesima. Brzina (Velocity) se odnosi na brzinu generisanja, prikupljanja i analize podataka, što omogućava donošenje odluka u realnom vremenu. Raznovrsnost (Variety) označava različite tipove podataka, uključujući strukturirane, polu-strukturirane i nestrukturirane podatke, prikupljene iz brojnih izvora. Na kraju, tu je validnost/verodostojnost (Validity/Veracity), koja se odnosi na kvalitet i pouzdanost podataka, koji su ključni za preciznu analizu i donošenje tačnih odluka.

2.3. Povezanost Cloud computing i Big Data koncepta

Cloud computing i Big Data su međusobno povezane tehnologije koje su promenile način na koji organizacije upravljaju i analiziraju podatke. Big Data, generisana iz raznovrsnih izvora, često dolazi u različitim formatima. Pružaoci cloud usluga nude alate poput veštačke inteligencije za standardizaciju i analizu ovih podataka.

Cloud platforme, poput Amazon S3 ili Google Cloud Storage, omogućavaju centralizovano i skalabilno skladištenje velikih količina podataka. U kombinaciji sa alatima poput Hadoop-a i Spark-a, podaci se mogu brzo obrađivati i transformisati u korisne uvide. Dodatno, alati za analitiku i vizualizaciju, kao što su Power BI i Tableau, pomažu organizacijama da identifikuju trendove i donesu bolje odluke.

Jedna od najvećih prednosti cloud-a je skalabilnost, zbog toga šro organizacije mogu povećavati ili smanjivati resurse prema potrebama, plaćajući samo za ono što koriste. Ova fleksibilnost omogućava preduzećima svih veličina da analiziraju podatke na efikasan i isplativ način. Kombinacija cloud tehnologija i Big Data tako otvara vrata ka inovacijama, smanjenju troškova i optimizaciji poslovanja.

3. CLOUD SERVISI ZA OBRADU BIG DATA

Serijska obrada (batch processing) i strim obrada (stream processing) su dva osnovna načina obrade velikih skupova podataka. Ove metode prilagođene su specifičnim potrebama i slučajevima upotrebe. Serijska obrada omogućava analizu podataka iz prošlosti u unapred definisanim vremenskim intervalima, dok je strim obrada optimalna za aplikacije kojima su potrebni trenutni rezultati, kao što su detekcija prevara u realnom vremenu ili praćenje finansijskih transakcija.

Servisi za obradu velikih skupova podataka u cloud-u pružaju skalabilnost, pouzdanost i visoke performanse, omogućavajući kompanijama efikasno upravljanje i analizu podataka. Ključne cloud platforme i njihovi servisi uključuju:

- Amazon Web Services (AWS) – alati poput Amazon S3 za skladištenje, EMR za obradu (Hadoop/Spark), Redshift za analitiku i Kinesis za strimovanje podataka u realnom vremenu.
- Google Cloud Platform (GCP) – BigQuery za analitiku, Dataflow za paralelnu obradu, Dataproc za Hadoop/Spark i Pub/Sub za razmenu poruka.
- Microsoft Azure – HDInsight za Hadoop/Spark, Synapse Analytics za skladištenje i obradu, Data Lake Storage za velike skupove podataka i Stream Analytics za real-time analitiku.
- IBM Cloud – Watson Studio za AI, Cloud Object Storage za skladištenje i Streams za analitiku u realnom vremenu.
- Oracle Cloud – Autonomous Data Warehouse za AI analitiku i Big Data Service za Hadoop bazirane procese.
- Alibaba Cloud – MaxCompute za analizu podataka, DataWorks za ETL i EMR za Hadoop/Spark klastere.
- Cloudera – Platforma za hibridna rešenja koja kombinuje upravljanje podacima, analitiku i AI.
- Apache Hadoop: Osnova za mnoge Big Data servise sa HDFS za skladištenje i MapReduce za obradu.

4. OBRADA KAGGLE DATASETA POMOĆU GCP BIGQUERY SERVISA

Google Cloud BigQuery je platforma za podatke koja pomaže pri upravljanju i analiziranju podataka sa ugrađenim funkcijama kao što su mašinsko učenje, pretraga, analiza i poslovna inteligencija. Serverless arhitektura BigQuery-ja omogućava korišćenje programskog jezika poput SQL-a i Pythona za odgovaranje na najveća pitanja, bez potrebe za upravljanjem infrastrukturom [3]. Podaci su automatski replicirani na više lokacija radi visoke dostupnosti i otpornosti.

4.1. Opis skupa podataka sa Kaggle platforme

Skup podataka koji je korišćen pri obradi je dostupan na sledećem linku na Kaggle platformi:

https://www.kaggle.com/datasets/asaniczka/forex-exchange-rate-since-2004-updated-daily/data

Skup podataka o dnevnim valutnim kursevima pruža istorijske i aktuelne kurseve za preko 160 valuta. Ovaj skup podataka se ažurira svakodnevno i sadrži podatke o dnevnim valutnim kursevima od 2004. godine do danas. Uključuje pet kolona: valuta (currency) je kod strane valute, npr. USD za američki dolar, EUR za evro. Tip podataka je string. Sledeća kolona je osnovna valuta (base_currency) koja predstvalja kod osnovne valute, u odnosu na koju se kurs izračunava, u ovom slučaju EUR. Tip podataka je string. Sledi kolona za puno ime valute (currency_name), kao što su „Američki dolar" ili „Evro". Tip podataka je string. Četvrta kolona je vrednost valutnog kursa (exchange_rate), tj. odnos između strane valute i osnovne valute. Ovo je broj koji pokazuje koliko osnovne valute vredi jedna jedinica strane valute. Tip podataka je float. Poslednja kolona je datum (date) kada je valutni kurs zabeležen. Tip podataka je datetime.

Skup podataka sadrži skoro 400000 redova podataka o dnevnim valutnim kursevima i trenutna verzija zauzima 17.01 MB, što ga čini dovoljno obimnim za detaljne analize kretanja valutnih kurseva i tržišnih trendova na globalnom nivou.

4.2. Koraci za uvoz skupa podataka i SQL analiza

Koraci:

1. Priprema podataka – preuzeti dataset u CSV format i pripremiti za uvoz.
2. Kreiranje projekta – prijava na Google Cloud Console i kreiranje novog projekta.
3. Uvoz podataka – kliknuti na Create Table i izaberite CSV fajl sa lokalnog diska ili Google Cloud Storage-a. Definisati šemu kolona.
4. Opcije uvoza – Append to table ili Replace table
5. Analiza pomoću SQL upita za filtriranje i analizu podataka, kao npr. u listingu 1.

```
SELECT *
FROM `masterrad-438309.dailyForexRates.DailyForexRates`
WHERE date >= '2023-01-01' AND currency = 'USD'
ORDER BY exchange_rate ASC;
```

Listing 1. *SQL upit za dobavljanje liste svih kurseva za USD od početka 2023. godine*

5. ANALIZA SQL UPITA, KREIRANJE MODELA I PREDIKCIJA POMOĆU BIGQUERY ML

5.1. Analiza SQL upita i prikaz rezultata

Prvi SQL upit kroz koji su podaci analizirani se odnosi na broj jedinstvenih valuta. SQL upit je korišćen za prebrojavanje različitih valuta u tabeli. Identifikovane su sve jedinstvene vrednosti u koloni valute, što pomaže u analizi pokrivenosti skupa podataka.

Drugi upit se odnosi na prosečni mesečni kursevi između RSD i EUR. Grupisanjem po godini i mesecu, dobijeni su prosečni kursevi, zaokruženi na dve decimale, sa ciljem praćenja trendova na mesečnom nivou. BigQuery nudi nekoliko različitih mogućnosti prikaza reultata, među kojima je i tabelarni prikazan na slici 1.

Trećim upitom izvučeni su podaci o dnevnim kursevima RSD, hronološki poredani. Rezultati omogućavaju vremensku analizu promenjivosti kursa i vizualizaciju trendova.

Row	year	month	avg_exchange_rate
111	2024	1	117.21
112	2024	2	117.21
113	2024	3	117.2
114	2024	4	117.13
115	2024	5	117.14
116	2024	6	117.07
117	2024	7	117.05
118	2024	8	117.03
119	2024	9	117.06
120	2024	10	117.04

Slika 1. *Tabelarni prikaz rezultata SQL upita*

5.2. Kreiranje prediktivnog modela pomoću BigQuery ML

BigQuery ML je alat koji omogućava kreiranje i treniranje modela mašinskog učenja direktno u BigQuery-u koristeći SQL upite. Ovo pojednostavljuje proces primene mašinskog učenja, jer korisnici mogu raditi sa SQL-om, koji je već poznat mnogim analitičarima i programerima [4]. Prednost BigQuery ML-a leži u mogućnosti direktne primene na postojeće podatke u BigQuery-u, čime se izbegava potreba za prenosom podataka u drugo okruženje. Kreiran je model predikcije za kurs američkog dolara (USD) koristeći vremenske serije ARIMA modela. Model predviđa kurs USD za narednih 15 dana i evaluira se na osnovu podataka u BigQuery bazi podataka.

ARIMA model (Autoregressive Integrated Moving Average) koristi se za analizu i predikciju vremenskih serija, posebno onih koje nisu stacionarne [5]. Cilj ARIMA modela je da pronađe što precizniji matematički opis podataka, što omogućava bolje razumevanje serije i tačnije predviđanje budućih vrednosti.

U ovom procesu je kreiran ARIMA model za predikciju kursa američkog dolara na osnovu podataka o dnevnim kursevima od 1. januara 2016. do 1. septembra 2024. Prvo je postavljen model u BigQuery koristeći podatke iz tabele koja sadrži dnevne kurseve. Parametri modela su automatski podešeni kako bi optimizovali predikciju.

Zatim je upotrebljena funkcija ML.FORECAST za generisanje predikcija kursa na sledećih 15 dana sa 80% nivoa poverenja.

Nakon što su predikcije izvršene, model je evaluiran koristeći stvarne podatke od 1. do 15. septembra 2024. kako bi se proverila tačnost predikcija. Rezultati evaluacije su pokazali različite metrike greške, kao što su MAE, MSE, RMSE, MAPE i SMAPE, što je omogućilo analizu preciznosti modela. Iako su vrednosti MAE i RMSE bile relativno niske, visoki procenti greške (MAPE i SMAPE) ukazali su na to da model može imati poteškoće sa sezonskim promenama ili neredovnim podacima.

5.3. Identifikacija vezanih valuta

U sljedećem delu analize, identifikovane su valute vezane za evro iz posmatranog skupa podataka. Valutni kursevi mogu biti fiksni ili promenljivi. Za identifikaciju vezanih valuta posmatrani su podaci sa istorijom valutnih kurseva pre 2015. godine i izračunat je koeficijent varijacije (CV), koji pokazuje stabilnost kursa, pomoću formule (1):

$$CV = \frac{\sigma}{\mu} \qquad (1)$$

Početna analiza se bazira na filtriranju valuta koje su ušle u skup podataka pre 2015. godine, jer se pretpostavlja da su ove valute stabilnije, imaju dužu istoriju i mogu biti pogodnije za dalju analizu. SQL upit je korišćen da bi se identifikovale valute koje su prvi put prikazane u skupu podataka pre 1. januara 2015. Na osnovu ovog upita, dobijeno je 134 valuta koje zadovoljavaju ovaj kriterijum.

Za svaku od identifikovanih valuta, izračunata je prosečna vrednost valutnog kursa, kao i standardna devijacija koja pokazuje promenjivost kursa tokom vremena. Ovo je ključno za razumevanje koliko se kurs jedne valute menja u odnosu na evro. Prosečna vrednost kursa daje osnovnu vrednost za analizu, dok standardna devijacija pomaže u oceni stabilnosti kursa.

Nakon što su izračunati srednji kurs i standardna devijacija, koristi se koeficijent varijacije (CV) kao kriterijum za identifikaciju vezanih valuta Valute sa niskim CV (manje od 0.03) smatraju se stabilnim, što znači da njihovi kursevi imaju malu varijaciju oko prosečne vrednosti, što može ukazivati na to da su vezane za evro. Na osnovu ovog kriterijuma, identifikovane su valute sa malim CV, koje su potencijalno vezane za evro.

Nakon identifikacije vezanih valuta pomoću SQL upita i analize CV, rezultati su upoređeni sa informacijama sa Wikipedije o međunarodnom statusu i upotrebi evra, koje su dostupne na sljedećem linku:

https://en.wikipedia.org/wiki/International_status_and_usage_of_the_euro#Pegged_currencies

Valute koje su tačno identifikovane kao vezane za evro su BAM, BGN, DKK, KMF, MAD, MKD i XAF. Takođe su identifikovane valute koje nisu bile prisutne u analizi, kao što su CVE, STN, XOF i XPF, pri čemu su valute poput STN-a bile izostavljene zbog nedostatka podataka, dok su ostale dodane u poslednjem ažuriranju podataka krajem 2023. godine. Postoje i dodatne valute koje se nalaze u listi identifikovanih vezanih valuta, ali nisu prisutne na Wikipediji: RSD, HRK i XDR.

6. ZAKLJUČAK

Ovaj rad je uspešno pokazao povezanost Cloud computing i Big Data koncepta, kao i primenu cloud tehnologija za analizu i predikciju tržišta valuta. Korišćenjem Google Cloud BigQuery platforme, analizirani su dnevni kursevi valuta, identifikovani dugoročni trendovi i razvijen model za predviđanje vrednosti kurseva. SQL upiti su omogućili analize, dok je ARIMA model delimično uspešno predvideo vrednosti kurseva u određenim periodima što ukazuje na potencijal, ali i na prostor za unapređenje ovog pristupa.

Google BigQuery se ističe u izvođenju kompleksnih analiza, dok za jednostavnije operacije nije najučinkovitiji. Rad otvara mogućnosti za dalja istraživanja, kao što su korišćenje složenijih mašinskih modela i uvođenje ekonomskih faktora za bolje predikcije. Takođe, širenje analize na veće vremenske intervale i druge valute može doprineti boljem razumevanju globalnih finansijskih tokova. Ovaj rad pruža čvrstu osnovu za buduća istraživanja u oblasti cloud tehnologija i analize podataka.

7. LITERATURA

[1] https://aws.amazon.com/what-is-cloud-computing/ (pristupljeno u septembru 2024.)

[2] https://www.pcmag.com/encyclopedia/term/big-data (pristupljeno u septembru 2024.)

[3] https://cloud.google.com/bigquery/docs/introduction (pristupljeno u septembru 2024.)

[4] https://cloud.google.com/bigquery/docs/create-machine-learning-model#sql (pristupljeno u oktobru 2024.)

[5] https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average (pristupljeno u oktobru 2024.)

Kratka biografija:

Bosiljka Todić je rođena 2000. godine u Bijeljini, Bosna i Hercegovina. Završila je gimnaziju „Vaso Pelagić" u Brčkom, 2019. godine. Fakultet Tehničkih Nauka u Novom Sadu je upisala 2019. godine. Diplomirala je u septembru 2023. godine na smeru Primenjeno softversko inženjerstvo. Uspešno je ispunila sve akademske obaveze i položila sve ispite predviđene master studijskim programom Primenjeno softversko inženjerstvo.

kontakt: bosiljkatodic00@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.41
DOI: https://doi.org/10.24867/33BE07Nedic

PLATFORMA ZA VIZUALIZACIJU DISTRIBUIRANIH ALGORITAMA NA PRIMERU KLASE ALGORITAMA ZA IZBOR LIDERA

PLATFORM FOR VISUALIZING DISTRIBUTED ALGORITHMS ON THE EXAMPLE OF A CLASS OF ALGORITHMS FOR LEADER ELECTION

Aleksandra Nedić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je predstavljena proširiva platforma za vizualizaciju distribuiranih algoritama za izbor lidera u sinhronim mrežama. Platforma omogućava korisniku da dodaje nove algoritme pored predefinisanih, kao i da manipuliše sistemom kroz dinamičko dodavanje i uklanjanje čvorova koji učestvuju u izvršavanju algoritama. Detaljno se razmatraju implementirani algoritmi za izbor lidera, različite topologije i tehnološke osnove platforme, uz prikaz njenih ključnih elemenata i načina funkcionisanja. Rad takođe opisuje specifikaciju i arhitekturu sistema, kao i implementaciju platforme.*

Ključne reči: *distribuirani algoritmi, biranje lidera, topologije, Chang-Roberts, Gallager-Humblet-Spira, Hirschberg-Sinclair, Bully, Hypercube*

Abstract – *The paper presents an extensible platform for visualizing distributed algorithms for leader election in synchronous networks. The platform allows the user to add new algorithms in addition to predefined ones, as well as to manipulate the system through dynamic addition and removal of nodes participating in the execution of algorithms. The implemented algorithms for the selection of leaders, different topologies and technological foundations of the platform are discussed in detail, with a presentation of its key elements and ways of functioning. The paper also describes the system specification and system architecture, as well as the platform implementation.*

Keywords: *distributed algorithms, leader election, topologies,fklas Chang-Roberts, Gallager-Humblet-Spira, Hirschberg-Sinclair, Bully, Hypercube*

1. UVOD

U kontekstu izgradnje kompleksnih sistema, postoje dva glavna pristupa: monolitni i distribuirani. Monolitne arhitekture integrišu sve komponente u jedan proces, dok distribuirani sistemi funkcionišu kroz više povezanih čvorova bez centralnog autoriteta. To čini praćenje komunikacije, stanja čvorova i toka izvršavanja algoritama složenim. Zbog toga je razvijena edukativna platforma koja vizuelno prikazuje rad distribuiranih algoritama, uključujući topologiju sistema, razmenu poruka i tabelu rutiranja. Platforma inicijalno podržava algoritme za izbor lidera (engl. *Leader Election*) [1] u sinhronim mrežama. Omogućava korisnicima interaktivno razumevanje ponašanja distribuiranih sistema i algoritama nakon izbora lidera. Platforma je proširiva i omogućava jednostavno dodavanje novih algoritama putem Python skripti.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji je mentor bio dr Milan Stojkov, docent

2. TOPOLOGIJE

Topologija unutar distribuiranih sistema predstavlja način na koji čvorovi komuniciraju i sarađuju. Ona definiše organizaciju čvorova i putanje preko kojih se podaci prenose. Topologije definisane u okviru platforme jesu prsten [2], hiperkocka [3], meš [4] i mreža.

2.1. Prsten

Prsten topologija je struktura u distribuiranim sistemima u kojoj su čvorovi povezani u krug. Svaki čvor je direktno povezan sa svoja susedna dva čvora formirajući zatvoreni krug. Podaci se kreću kroz prsten od jednog do drugog čvora dok ne stignu do željenog odredišta.

2.2. Hiperkocka

Hiperkocka je složena struktura u distribuiranim sistemima koja povezuje čvorove tako da formira graf koji predstavlja n-dimenzionalnu kocku. Za n dimenzija, broj čvorova u hiperkocki mora biti 2^n. Svakom čvoru je dodeljen binarni broj od n bitova, a čvorovi su povezani ako se njihovi binarni kodovi razlikuju u tačno jednom bitu.

2.3. Meš

Meš topologija predstavlja strukturu u kojoj postoje tri vrste čvorova u zavisnosti od broja suseda. Čvorovi u uglovima imaju dva suseda, čvorovi na ivicama imaju tri, dok unutrašnji čvorovi imaju četiri suseda. Ukupan broj veza m u meš topologiji veličine izračunava se formulom $m = 2ab - a - b$, što predstavlja horizontalne i vertikalne veze između čvorova.

2.4. Mreža

Mreža je univerzalan naziv za topologiju u kojoj su čvorovi povezani na određeni način. Ukoliko je unutar mreže svaki čvor povezan sa ostalim čvorovima, u pitanju je kompletna mreža, dok je parcijalna mreža ona mreža u kojoj nisu svi čvorovi direktno povezani jedan sa drugim. Kod kompletnih mreža prenos podataka je veoma brz i ovakva

mreža je korisna u sistemima u kojima je minimalno kašnjenje ključno.

3. ALGORITMI ZA BIRANJE LIDERA

Algoritmi za biranje lidera igraju veoma važnu ulogu u distribuiranim sistemima jer omogućavaju izbor jednog čvora koji će preuzeti ulogu koordinatora. Ovaj proces je ključan za osiguranje efikasnosti, stabilnosti i organizacije sistema. Koordinator deluje kao centralna tačka koja upravlja zadacima poput sinhronizacije čvorova i upravljanja resursima, što omogućava usklađenu komunikaciju i sprečava nastanak konflikata.

3.1. Chang-Roberts algoritam

Chang-Roberts algoritam [5] bira lidera u sistemima sa unidirekcionom prstenastom topologijom gde svaki čvor ima jedinstveni identifikator (ID). Bilo koji čvor može pokrenuti algoritam slanjem poruke sa svojim ID-jem susedu u smeru kazaljke na satu i time postaje učesnik. Prilikom primanja poruke, čvor:

1. prosleđuje poruku ako je ID u njoj veći od njegovog,
2. šalje svoju poruku ako je ID u njoj manji i nije učesnik,
3. ignoriše poruku ako je ID manji i već je učesnik,
4. proglašava se liderom ako je ID jednak njegovom.

3.2. Bully algoritam

Bully algoritam [6] je algoritam za biranje lidera u sistemima u kojima svaki čvor ima svoj ID i u kojima je mreža potpuno povezana, te svaki čvor može direktno komunicirati jedan sa drugim. Za lidera se bira onaj čvor koji ima najveći ID. Proces započinje čvor koji otkrije da trenutni lider više nije aktivan, ukoliko, na primer, ne odgovara na poruke. Čvor pokreće proces za izbor tako što šalje poruku svim čvorovima koji imaju veći ID od njega. Ukoliko ne dobije odgovor od čvorova sa većim ID-jem, taj čvor proglašava sebe liderom. Ako čvor sa većim ID-jem odgovori na poruku inicijatoru, on preuzima proces izbora lidera i šalje poruku čvorovima koji imaju veći ID od njega.

3.3. Hypercube algoritam

Hypercube algoritam [7] koristi se u distribuiranim sistemima sa hiperkockastom topologijom. Za hiperkocku dimenzije k postoji ukupno 2^k čvorova, pri čemu svaki čvor ima binarni ID dužine k bita. Algoritam pokreće bilo koji čvor slanjem poruke sa svojim ID-jem susedima u k koraka. U svakom koraku poruka se šalje susedu koji se razlikuje u jednom bitu — počevši od najmanje značajnog (desnog) ka najznačajnijem (levom) bitu. Po prijemu poruke, čvor:

1. ukoliko je ID iz poruke veći od njegovog ID-ja, čvor označava sebe da nije lider
2. ukoliko je ID iz poruke manji od njegovog ID-ja, a čvor do sada nije bio označen kao ne-lider, čvor označava sebe kao lidera

Čvor zatim šalje svoj ID ostalim susedima. Po završetku k koraka, čvor koji je označen kao lider postaje lider, dok ostali završavaju rad.

3.4. Gallager-Humblet-Spira algoritam

Gallager-Humblet-Spira algoritam [8] je algoritam za pronalaženje minimalnog razapinjujućeg stabla (engl. Minimum spanning tree) u povezanim grafovima.
Koristi se u distribuiranim sistemima kod kojih veze između čvorova imaju određenu težinu. Algoritam se zasniva na ideji fragmentacije grafa, gde čvorovi i grane formiraju fragmente koji se iterativno spajaju dok se ne formira jedno stablo. Na početku algoritma, svaki čvor predstavlja fragment za sebe i ima sopstveni ID. Jedan čvor započinje proces za izbor lidera tako što šalje poruku susedu sa kojim ima vezu najmanje težine. Kada čvor primi poruku, on upoređuje svoj ID sa ID-jem iz poruke, ukoliko je njegov ID manji, ova dva čvora se spajaju u jedan fragment, a za ID fragmenta se uzima manji od ova dva. Ovaj proces se nastavlja iterativno dok svi čvorovi ne postanu deo jednog fragmenta, čiji ID ima najmanju vrednost. Na ovaj način je formirano minimalno razapinjujuće stablo, a lider postaje čvor čiji ID predstavlja ID čitavog fragmenta.

3.5. Hirschberg-Sinclair algoritam

Hirschberg-Sinclair algoritam [9] koristi se za izbor lidera u distribuiranim sistemima sa bidirekcionom prstenastom topologijom, gde svaki čvor ima jedinstven ID. Izvršava se u više iteracija, pri čemu poruke putuju na sve većim udaljenostima, udvostručujući domet u svakoj iteraciji, tako da važi $k \leqslant 2^i$, gde je k udaljenost, a i broj iteracije. Algoritam počinje slanjem izborne poruke (engl. election message) u oba smera. Prilikom prijema izborne poruke, čvor upoređuje svoj ID sa ID-jem iz poruke:

1. prosleđuje poruku ako je veći ID i domet nije dostignut,
2. šalje povratnu poruku ako je domet dostignut,
3. ignoriše poruku ako je ID manji,
4. proglašava se liderom ako je ID isti.

U slučaju povratne poruke (eng. reply message), ukoliko ID-jevi čvora i poruke nisu isti, čvor vraća poruku svom susedu, dok ukoliko su ID-jevi isti, i povratna poruka je stigla iz oba smera, čvor je lokalni lider, čime je jedna iteracija završena i čvor započinje sledeću fazu elekcije.

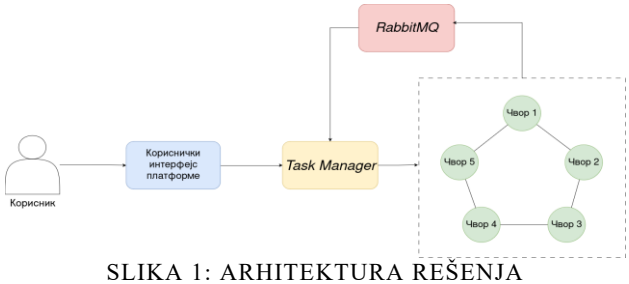
4. MODEL SISTEMA

4.1. Arhitektura sistema

U arhitekturi platforme centralnu ulogu ima task manager, čvor zadužen za upravljanje i orkestraciju celokupnog sistema. On poseduje kompletnu konfiguraciju sistema, uključujući definicije čvorova, algoritama i njihove komunikacione topologije. Ova konfiguracija se čuva u JSON formatu na fajl sistemu i povezuje sa Docker kontejnerom putem bind mount mehanizma, što omogućava lako ažuriranje bez potrebe za restartovanjem sistema. Task manager je jedini čvor izložen spoljašnjoj mreži i služi kao komunikaciona spona između korisničke aplikacije i ostatka sistema. Koristi HTTP protokol za prijem korisničkih zahteva, poput dodavanja čvorova ili pokretanja algoritama, dok se WebSocket koristi za dvosmernu komunikaciju u realnom vremenu, omogućavajući korisniku da prati status algoritama i trenutno stanje čvorova. Osnovna funkcija task manager-a je koordinacija - inicira i uklanja čvorove, pokreće

algoritme, upravlja njihovim izvršavanjem i prati njihov status. Informacije o izvršavanju prosleđuje korisniku kako bi on imao uvid u rad sistema u realnom vremenu. Pored task manager-a, sistem se sastoji od više čvorova koji međusobno komuniciraju sinhrono putem HTTP-a kako bi izvršavali distribuirane algoritme. Svaki čvor, takođe, asinhrono šalje informacije o svom stanju task manager-u koristeći RabbitMQ message broker, čime se omogućava ažuriranje prikaza sistema u realnom vremenu.

Arhitektura sistema prikazana je na slici 1.



SLIKA 1: ARHITEKTURA REŠENJA

4.2 Funkcionalnosti sistema

Platforma je namenjena jednoj vrsti korisnika i ne zahteva autentifikaciju. Korisnik može pregledati listu dostupnih algoritama, izabrati neki i pratiti njegovo izvršavanje na grafičkom prikazu.

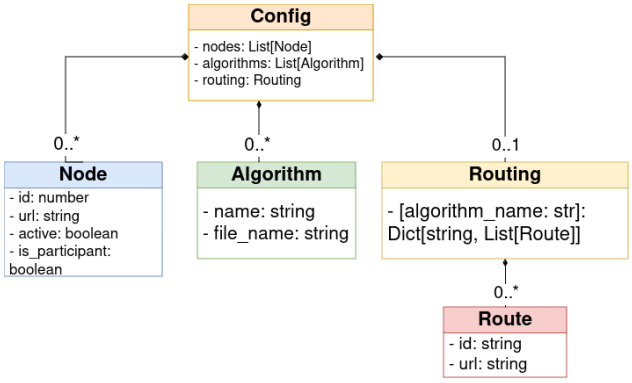
Postoji i posebna stranica za pregled celog sistema, uključujući čvorove sa parametrima i tabele rutiranja koje prikazuju učesnike i njihove veze. Na toj stranici korisnik može dodavati nove algoritme, menjati čvorove i njihove karakteristike, kao i dodavati nove čvorove unosom podataka. Takođe, može menjati tabele rutiranja postojećih algoritama i time prilagođavati topologiju sistema.

4.3. Model podataka

Model podataka platforme je jednostavan i odnosi se na konfiguraciju koja definiše algoritme koji su dostupni u sistemu, čvorove zajedno sa njihovim karakteristikama i tabele rutiranja koje definišu topologiju algoritama.

Konfiguracija sistema predstavljena je klasom *Config*, koja sadrži atribute *nodes*, *algorithms*, *routing*. Atribut *nodes* predstavlja listu čvorova koji su prisutni u sistemu i prikazani su klasom *Node*, atribut *algorithms* označava algoritme unutar sistema i oni su predstavljeni klasom *Algorithm*, dok atribut *routing* predstavlja tabelu rutiranja za svaki algoritam i unutar platforme je modelovan klasom *Routing*. Klasa *Node* sadrži atribute *id*, *url*, *active*, is*_participant* koji definišu osnovne karakteristike čvora u sistemu. Klasa *Algorithm* sadrži dva atributa, *name* koji predstavlja ime algoritma, i *file_name* koji označava naziv Python skripte u kojoj je algoritam implementiran. Tabela rutiranja, predstavljena klasom *Routing*, modeluje topologiju komunikacije između čvorova za svaki algoritam. Klasa *Routing* sadrži dinamičke parametre, gde svaki ključ predstavlja ime algoritma, a njegova vrednost je rečnik u kome su ključevi ID-jevi čvorova, dok su vrednosti liste objekata koji opisuju rute. Svaka ruta je predstavljena klasom *Route* koji sadrži atribute *id* (ID odredišnog čvora) i *url* (adresa odredišnog čvora).

Klasni dijagram prikazan je na slici 2.



SLIKA 2: KLASNI DIJAGRAM

5. IMPLEMENTACIJA

Serverska strana platforme sastoji se iz više servisa koji igraju različite uloge. Svaki servis predstavlja čvor i implementiran je u Python programskom jeziku uz pomoć FastAPI radnog okvira za lakšu komunikaciju između servisa i klijentske strane.

Svaki servis je kontejnerizovan radi lakšeg skaliranja čvorova. Za kontejnerizaciju se koristi Docker alat, a u Docker Compose datoteci, pored servisa koji predstavljaju čvorove, definisan je i RabbitMQ za asinhronu komunikaciju, kao i volumen koji je deljen između task manager-a i ostalih čvorova, a predstavlja direktorijum u kojem su smeštene sve Python skripte koje predstavljaju implementaciju algoritama. Na ovaj način, bez prekidanja rada platforme korisnici mogu da definišu nove algoritme koji će biti odmah dostupni čvorovima na izvršavanje.

Kompletna konfiguracija platforme koju čine dostupni čvorovi i algoritmi, kao i tabela rutiranja za svaki od algoritama smeštena je u JSON datoteci i njoj može da pristupi centralni čvor radi orkestracije zadataka.

Konfiguracija se učitava u sistem i zadržava u radnoj memoriji prilikom pokretanja centralnog čvora. Konfiguracija se nalazi u globalnom objektu tipa Config i pri svakom klijentovom zahtevu za pokretanje algoritama koriste se podaci iz konfiguracije, a u slučaju izmena, nova konfiguracija se zapisuje u JSON datoteku.

Prilikom pokretanja servisa, pored učitavanja konfiguracije, na pozadinskoj niti učitava se i statistika čvorova poput zauzeća memorije i korišćenja procesorske moći uz pomoć docker.py biliboteke, kako bi korisnik imao uvid u njihov kapacitet prilikom izvršavanja algoritama. Na kraju, task manager na pozadinskoj niti otvara konekciju sa RabbitMQ servisom kako bi bio spreman da konzumira podatke koji mu pristižu od strane čvorova i prosleđuje ih klijentu.

Centralni čvor se sastoji iz četiri krajnje tačke koje pružaju funkcionalnosti platforme. Postoji krajnja tačka za dobavljanje konfiguracije koja će biti prikazana korisniku radi uvida i manipulacije sistema, zatim postoji mogućnost izmene konfiguracije pri čemu se mogu dodavati ili uklanjati čvorovi koji su specificirani u konfiguraciji i poslednje dve krajnje tačke odnose se na pokretanje algoritma i dodavanje novih algoritama.

Task manager započinje izvršavanje algoritama inicijalizacijom svih čvorova koji se nalaze u tabeli rutiranja za određeni algoritam. Pod inicijalizacijom se podrazumeva slanje podataka potrebnih za izvršavanje

algoritma. Kada su svi čvorovi uspešno inicijalizovani, task manager na slučajan način bira jedan čvor koji će započeti izvršavanje algoritma za biranje lidera.
Čvor sadrži pet krajnjih tačaka koje omogućavaju inicijalizaciju čvora, pokretanje algoritma, deaktivaciju čvora, prijem poruke od strane drugog čvora prilikom izvršavanja algoritma i izvršavanje krajnje funkcije nakon što je lider izabran.
Svaki čvor sadrži globalnu promenljivu koja predstavlja instancu klase Node i koja se inicijalizuje prilikom inicijalizacije samog čvora. Unutar klase Node čuvaju se podaci koje task manager šalje čvoru prilikom inicijalizacije.
Klasa Node ima nekoliko metoda, a najbitnije su metoda za započinjanje izvršavanja algoritma, metoda za slanje poruke narednom čvoru, metoda za obradu odgovora koji je naredni čvor poslao i metoda za izračunavanje sume slučajno odabranih brojeva.
Kako bi platforma bila proširiva i kako bi mogla da izvršava veliki broj različitih algoritama, sve što je zajedničko svim algoritmima nalazi se u okviru metoda ove klase, dok je sve specifično za određeni algoritam smešteno u posebnom modulu. Klasa Node pretpostavlja da svaki modul sadrži tri funkcije, a to su:

1. run_algorithm(node_data, routing_table),
2. handle_message(node_data, routing_table, data)
3. handle_result(node_data, routing_table, result)

Ove tri funkcije zajedno predstavljaju implementaciju određenog distribuiranog algoritma.
Nakon što je završen algoritam, odabrani lider šalje poruku task manager-u čime ga obaveštava da je izvršavanje završeno. Task manager zatim poziva funkciju za izračunavanje zbira brojeva za svaki od čvorova. Nakon što svi čvorovi vrate rezultat, task manager izračunava ukupan zbir koji je stigao sa svih čvorova. Konačan rezultat, zajedno sa međurezultatima prosleđuje se korisniku kao konačan korak algoritama za biranje lidera.

6. ZAKLJUČAK

Razumevanje distribuiranih algoritama je izazovno zbog paralelnog izvršavanja na velikom broju čvorova, složenih topologija i načina komunikacije među čvorovima. Ovi izazovi su motivisali razvoj platforme koja vizuelno i u realnom vremenu prikazuje izvršavanje klasičnih algoritama za izbor lidera u sinhronim mrežama, omogućavajući praćenje topologije i stanja čvorova tokom izvršavanja. Platforma se sastoji od centralnog servisa, task managera, koji upravlja konfiguracijom, komunicira sa klijentom i koordinira čvorove. Čvorovi zajednički izvršavaju algoritme, a sistem je proširiv dodavanjem novih algoritama putem Python skripti sa tri definisane funkcije. Skaliranje je omogućeno dodavanjem ili uklanjanjem čvorova. Buduća unapređenja uključuju podršku za algoritme sa kašnjenjem poruka, čime bi se omogućila šira vizualizacija, kao i debagovanje i korak-po-korak izvršavanje za bolje razumevanje algoritama.

7. LITERATURA

[1] Shirali, M., Toroghi, A. H., & Vojdani, M. (2008). Leader election algorithms: History and novel schemes. In *2008 Third International Conference on Convergence and Hybrid Information Technology* (pp. 1001-1006). IEEE. https://doi.org/10.1109/ICCIT.2008.57

[2] Huang, M., & Bode, B. (2005). A performance comparison of tree and ring topologies in distributed systems. In *19th IEEE International Parallel and Distributed Processing Symposium* (pp. 8). IEEE. https://doi.org/10.1109/IPDPS.2005.57

[3] Liu, H. (2009). The structural features of enhanced hypercube networks. In *2009 Fifth International Conference on Natural Computation* (pp. 345-348). IEEE. https://doi.org/10.1109/ICNC.2009.191

[4] Santoro, N. (2006). *Design and analysis of distributed algorithms*. Wiley.

[5] Chang, E., & Roberts, R. (1979). An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, *22*(5), 281-283. https://doi.org/10.1145/359024.359027

[6] Garcia-Molina, H. (1982). Elections in a distributed computing system. *IEEE Transactions on Computers*, *C-31*(1), 48-59. https://doi.org/10.1109/TC.1982.1675885

[7] McBryan, O. A., & Van de Velde, E. F. (1987). Hypercube algorithms and implementations. *SIAM Journal on Scientific and Statistical Computing*, *8*(2), s227-s287. https://doi.org/10.1137/0908040

[8] Gallager, R. G., Humblet, P. A., & Spira, P. M. (1983). A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, *5*(1), 66-77. https://doi.org/10.1145/357195.357200

[9] Hirschberg, D. S., & Sinclair, J. B. (1980). Decentralized extrema-finding in circular configurations of processors. *Communications of the ACM*, *23*(11), 627-628. https://doi.org/10.1145/359024.359029

Kratka biografija:

Aleksandra Nedić rođena je 09.04.2000. godine u Vranju. U školskoj 2019/20 godini upisuje osnovne studije na Fakultetu tehničkih nauka, na smeru Softversko inženjerstvo i informacione tehnologije, koje završava školske 2022/23 sa prosečnom ocenom 9.80. Master rad na Fakultetu tehničkih nauka iz oblasti Softversko inženjerstvo i informacione tehnologije, usmerenje Elektronsko poslovanje odbranila je u školskoj 2024/25 godini.
kontakt: aleksandranedic843@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE08Bajat

MEĐUREPREZENTACIJE IZVORNOG KODA RUSTC KOMPAJLERA

INTERMEDIATE REPRESENTATIONS OF THE SOURCE CODE IN THE RUSTC COMPILER

Aleksa Bajat, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

Kratak sadržaj – *Ovaj rad istražuje arhitekturu prednjeg kraja (engl. frontend) Rust kompajlera (rustc), sa fokusom na ključnu ulogu koju međureprezentacije izvornog koda igraju u procesu prevođenja. Analizirane su faze od leksičke analize i parsiranja do generisanja Apstraktnog Sintaksnog Stabla (ASS), preko Međureprezentacije Visokog Nivoa (MVN/HIR), Tipizirane MVN (TMVN/THIR), do Međureprezentacije Srednjeg Nivoa (MSN/MIR). Rad objašnjava kako svaka od ovih reprezentacija omogućava ključne funkcionalnosti Rust jezika, uključujući proveru tipova, dijagnostiku grešaka, inkrementalno kompajliranje, proveru pozajmljivanja (borrow checking) i pripremu za dalju optimizaciju i generaciju koda u LLVM-u. Cilj je da se prikaže kako generisanje infrastrukture doprinosi garancijama memorijske bezbednosti i visokim performansama Rust programskog jezika.*

Ključne reči: *Rust, rustc, kompajler, frontend, međureprezentacije, ASS, MVN, TMVN, MSN, LLVM, analiza koda, optimizacija*

Abstract – *This paper explores the architecture of the Rust compiler's (rustc) frontend, focusing on the crucial role of intermediate representations (IRs) of source code in the compilation process. Phases from lexical analysis and parsing to the generation of the Abstract Syntax Tree (AST), through the High-Level Intermediate Representation (HIR), Typed HIR (THIR), to the Mid-Level Intermediate Representation (MIR) are analyzed. The paper explains how each of these representations enables key features of the Rust language, including type checking, error diagnostics, incremental compilation, borrow checking, and preparation for further optimization and code generation in LLVM. The aim is to demonstrate how generation of compiler infrastructure contributes to Rust's guarantees of memory safety and high performance.*

Keywords: *Rust, rustc, compiler, frontend, intermediate representations, AST, HIR, THIR, MIR, LLVM, code analysis, optimization*

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dunja Vrbaški, doc.

1. UVOD

Programski jezik *Rust* je stekao značajnu popularnost zahvaljujući svom fokusu na memorijsku bezbednost bez upotrebe sakupljača smeća (engl. *garbage collector*) i visokim performansama koje pariraju jezicima kao što su *C* i *C++*. Ove karakteristike čine ga idealnim za razvoj sistemskog softvera, operativnih sistema, veb servera i drugih aplikacija gde su performanse i bezbednost kritične [1]. Postizanje ovih ciljeva zahteva sofisticiran kompajler, rustc, čiji prednji kraj (frontend) igra ključnu ulogu u analizi, validaciji i transformaciji izvornog koda.

Prednji kraj rustc kompajlera koristi niz međureprezentacija (engl. *Intermediate Representations)* kako bi postepeno transformisao izvorni kod u formu pogodnu za dalje faze kompajliranja, uključujući optimizaciju i generaciju koda od strane *LLVM* bekenda [13]. Svaka međureprezentacija služi specifičnoj svrsi, omogućavajući različite vrste analiza i transformacija. Razumevanje ovih reprezentacija je ključno za razumevanje kako *Rust* postiže svoje garancije.

2. PUT IZVORNOG KODA U RUSTC KOMPAJLERU

Proces kompajliranja u rustc frontendu započinje leksičkom analizom i parsiranjem, nastavlja se kroz nekoliko nivoa međureprezentacija gde se vrše ključne analize i transformacije, a završava se generacijom Međureprezentacije Srednjeg Nivoa (MSN) koja se potom prosleđuje *LLVM*-u za dalju optimizaciju i generaciju mašinskog koda. U tipično struktuiranom kompajleru svaka od međureprezentacija predstavlja poseban program i posebno izvršavanje. Ovakvi kompajleri se nazivaju kompajleri zasnovani na prolascima - arhitektura cevi. *Rust* kompajler je u tranziciji između kompajlera zasnovanog na prolascima i kompajlera zasnovanog na potražnji. Kompajler zasnovan na potražnji koristi upite nad izvornim kodom i simultono izvršava velike celine kompajliranja. Izvršavanje upita je memoizovano. Samo prvi poziv upita izvršava komputacije, dok je svaki naredni keširan. Sve međureprezentacije nakon ASS se zasnivaju na sistemu upita, tj. samo lekser i parser rade po principu arhitekture cevi. Dugoročni cilj je refaktorizacija kompajlera tako da celokupan proces radi na osnovu sistema upita.

2.1. OD TOKENA DO APSTRAKTNOG SINTAKSNOG STABLA (ASS)

Prvi korak je leksička analiza, gde rustc_lexer, kao lekser niskog nivoa, čita izvorni kod kao niz karaktera, grupiše ih u lekseme (npr. ključne reči, identifikatori, operatori) i proizvodi tok tokena. Svaki token obično sadrži tip i, opciono, vrednost. *Rust*-ov lekser je ručno implementiran, što omogućava finu kontrolu nad procesom tokenizacije i rezultira detaljnijim i korisnijim porukama o greškama.

Nakon toga, *rustc_parse::lexer*, lekser višeg nivoa, dodatno obrađuje ove tokene. U ovoj fazi, vrši se interniranje simbola (engl. symbol interning), gde se stringovi poput identifikatora smeštaju u posebnu memorijsku oblast (arenu) tako da svaki jedinstveni string ima samo jednu kopiju. Ovo optimizuje upotrebu memorije i ubrzava poređenje simbola. Takođe se koriste strukture kao što su Span i SpanData za praćenje lokacije svakog tokena u izvornom kodu, što je ključno za preciznu dijagnostiku grešaka. Pre samog parsiranja, vrši se i rezolucija zagrada, gde se tok tokena strukturira u stabla tokena (*TokenTree*), olakšavajući kasniju obradu.

Parser zatim koristi ovaj obrađeni tok tokena da generiše Apstraktno Sintaksno Stablo (ASS ili AST), koje predstavlja hijerarhijsku strukturu izvornog koda [3]. ASS verno odražava strukturu korisničkog koda, eksplicitno prikazujući, na primer, prvenstvo operatora. Koren ASS-a je obično Crate struktura, dok su osnovni gradivni blokovi *Item*-i (npr. funkcije, strukture, moduli), koji sadrže atribute, jedinstveni identifikator (*NodeId*) i *Span*. Iako je ASS osnova za dalje analize, nije direktno pogodan za sve operacije zbog svoje bliskosti originalnom izvornom kodu. U ovoj fazi se vrši i ekspanzija makroa – moćne *Rust*-ove karakteristike koja omogućava metaprogramiranje ("kod koji generiše kod") i smanjuje potrebu za ponavljajućim kodom. Takođe, započinje inicijalna rezolucija imena, gde kompajler pokušava da poveže imena korišćena u kodu sa njihovim definicijama, koristeći koncept "rebara" (engl. *ribs*) za upravljanje vidljivošću imena u različitim opsezima (engl. *scopes*). Pre prelaska na sledeću reprezentaciju, ASS se validira, posebno nakon ekspanzije makroa koji mogu generisati sintaktički nekompletan ili neispravan kod.

2.2. MEĐUREPREZENTACIJA VISOKOG NIVOA (MVN ili HIR)

ASS se zatim "snižava" (engl. *lowering*) u Međureprezentaciju Visokog Nivoa (MVN) [4]. MVN je apstraktnija reprezentacija koja pojednostavljuje mnoge konstrukcije iz ASS-a; na primer, for petlje se prevode u while let konstrukcije, if let izrazi u match izraze, a impl Trait u parametrima funkcija u odgovarajuće generičke argumente. Svrha ovog snižavanja je da se smanji broj različitih sintaksnih formi (sintaksni šećer) sa kojima kasnije faze kompajlera moraju da rade, čime se pojednostavljuje analiza. HIR Crate struktura sadrži informacije o celokupnom kodu paketa, uključujući i delove koji možda nisu direktno pozvani, što je važno za analizu eksternih biblioteka. HIR je ključan za sistem upita (engl. *query system*) rustc-a. Sistem upita funkcioniše kao baza podataka kompajlera gde se rezultati različitih analiza (upita) keširaju (memoizacija). Svaki upit je funkcija koja mapira jedinstveni ključ (npr. identifikator stavke) na rezultat (npr. tip stavke, telo funkcije u MVN-u). Ako se upit ponovo pozove sa istim ključem, vraća se keširani rezultat umesto ponovnog izračunavanja. Ovo je temelj inkrementalnog kompajliranja: prilikom izmene koda, samo oni upiti čije su zavisnosti promenjene moraju ponovo da se izvrše. Sistem upita zahteva da provajderi upita budu determinističke funkcije i održava direkcioni aciklični graf (DAG) poziva upita kako bi se izbegli ciklusi. Za stabilnu identifikaciju čvorova grafa između različitih kompajliranja koriste se *DefPath* i *DefPathHash* strukture, kao i "otisci prsta" (engl. *fingerprints*) za efikasno poređenje stanja. Na MVN-u se vrše mnoge semantičke analize, kao što su rezolucija osobina (engl. *trait resolution*) i provere koherentnosti.

2.3. TIPIZIRANA MEĐUREPREZENTACIJA VISOKOG NIVOA (TMVN ili THIR)

Nakon MVN-a, generiše se Tipizirana Međureprezentacija Visokog Nivoa (TMVN). TMVN obogaćuje MVN informacijama o tipovima za svaki izraz i deo koda unutar tela funkcija; dakle, TMVN se generiše samo za izvršni kod [5]. TMVN je efemerna reprezentacija, što znači da se delovi generišu "na zahtev" i ne čuvaju se kompletno u memoriji tokom celog procesa, čime se smanjuje memorijski otisak kompajlera. U TMVN-u, mnoge implicitne operacije iz izvornog koda postaju eksplicitne: automatska referenciranja i dereferenciranja, pozivi metoda na osobinama, i preklopljeni operatori se prevode u eksplicitne pozive funkcija. Takođe, uništenje opsega (engl. scope destruction) je eksplicitno predstavljeno.

TMVN služi kao osnova za dve važne provere:

1. **Provera bezbednosti (engl. *check_unsafety*)**: Algoritam analizira unsafe kontekste, proverava da li se unsafe operacije (npr. dereferenciranje sirovog pokazivača) pozivaju van unsafe bloka, i da li unsafe blokovi zaista sadrže unsafe kod (ako ne, generiše se upozorenje - *lint*).
2. **Provera iscrpnosti šablona (engl. *pattern matching exhaustiveness*):** Za konstrukcije kao što su match, if let, while let, let else, pa čak i za argumente funkcija koji koriste destrukturiranje, TMVN omogućava proveru da li su svi mogući slučajevi pokriveni. Pored iscrpnosti, proverava se i "korisnost" svake grane, kako bi se detektovale nedostižne (redundantne) grane koda.

2.4. MEĐUREPREZENTACIJA SREDNJEG NIVOA (MVN ili HIR)

Konačna reprezentacija u frontendu je Međureprezentacija Srednjeg Nivoa (MIR). Uvođenje MSN je bilo motivisano potrebom za preciznijom kontrolom nad tokom izvršavanja, lakšim sprovođenjem Rust-specifičnih optimizacija i pojednostavljivanjem procesa dokazivanja memorijske bezbednosti, što je bilo teško postići direktnim prevođenjem sa MVN-a ili TMVN-a na LLVM IR. MSN je eksplicitna reprezentacija kontrolnog toka (control flow graph - CFG) programa [5]. CFG se sastoji od osnovnih

blokova (engl. basic blocks), gde svaki blok predstavlja niz naredbi koje se izvršavaju sekvencijalno, a poslednja naredba u bloku, terminator (engl. terminator), određuje u koji sledeći blok (ili blokove) će se preći. MSN koristi LIFO stek za smeštanje argumenata funkcija, lokalnih promenljivih i privremenih vrednosti. Memorijske lokacije na steku se identifikuju preko "mesta" (engl. places, npr. _1 za prvu lokalnu promenljivu, _0 za povratnu vrednost), a pristupi poljima struktura ili dereferenciranje se predstavljaju kao "projekcije" (npr. _1.polje, *_1). Izrazi koji generišu vrednosti nazivaju se "desne vrednosti" (RValues).

U neoptimizovanom MSN, eksplicitno se koriste iskazi StorageLive i StorageDead kako bi se označio početak i kraj životnog veka svake lokalne promenljive, što kompajleru daje jasnu informaciju o tome kada je memorija za datu promenljivu validna i kada se može bezbedno osloboditi.

Na MSN se vrši ključna analiza za Rust: analiza pozajmljivanja (*borrow checking*). Ovo uključuje implementaciju Neleksičkih Životnih Vekova (*Non-Lexical Lifetimes - NLL*). Tradicionalni leksički životni vekovi su vezani za opsege u kodu i mogu biti previše restriktivni. *NLL*, s druge strane, analizira stvarnu upotrebu pozajmica unutar grafa kontrole toka, omogućavajući preciznije i fleksibilnije određivanje tačnog trajanja svake pozajmice, što često rezultira prihvatanjem koda koji bi sa leksičkim životnim vekovima bio odbijen. MSN takođe služi za Rust-specifične optimizacije; na primer, kompleksne petlje ili match izrazi mogu biti dodatno pojednostavljeni (npr. for petlja se preko *loop* { *match* ... } sa *goto* naredbama može transformisati u efikasnije switch konstrukcije) pre nego što se kod prosledi *LLVM* bekendu za dalje, generalnije optimizacije i konačnu generaciju mašinskog koda.

3. ZNAČAJ MEĐUREPREZENTACIJA

Svaka od navedenih međureprezentacija u rustc frontendu ima svoju specifičnu i ključnu ulogu:

- **ASS:** Predstavlja vernu reprezentaciju izvornog koda, služi kao osnova za makro ekspanziju i inicijalnu rezoluciju imena. Njegova bliskost izvornom kodu omogućava precizne poruke o sintaksnim greškama.
- **MVN:** Kao apstrakcija nad ASS-om, pojednostavljuje strukturu koda i ključna je za rad sistema upita, omogućavajući semantičke analize i inkrementalno kompajliranje.
- **TMVN:** Obogaćujući HIR tipovima, omogućava detaljnu proveru tipova, proveru bezbednosti unsafe koda i iscrpnosti i korisnosti šablona, što su ključne komponente Rust-ove pouzdanosti.
- **MSN:** Kao eksplicitni graf kontrolnog toka, fundamentalan je za analizu pozajmljivanja (borrow checking) i implementaciju Neleksičkih Životnih Vekova (NLL), što direktno doprinosi memorijskoj bezbednosti. Takođe, omogućava Rust-specifične optimizacije.

Ova višefazna arhitektura, gde svaka IR rešava specifičan skup problema, omogućava rustc-u da efikasno sprovodi kompleksne analize neophodne za garantovanje memorijske bezbednosti i generisanje efikasnog koda, što su dve ključne prednosti Rust jezika. Modularnost pristupa olakšava razvoj i održavanje kompajlera, dok sistem upita dodatno doprinosi ukupnoj efikasnosti kroz memoizaciju i inkrementalno kompajliranje. Precizne dijagnostičke poruke su takođe rezultat mogućnosti da se greške lociraju u odgovarajućoj fazi i na odgovarajućem nivou apstrakcije.

4. BEZBEDNO NEBEZBEDAN

Analiza je provedena 2023. godine da se 12% paketa u *Rust* ekosistemu direktno oslanja na nestabilne funkcionalnosti, dok je 44% paketa indirektno zavisilo od nestabilnih funkcionalnosti da bi se kompajliralo. Procentualna zavisnost prema nestabilnim funkcionalnostima je visoka ali analiza nije sprovedena da se izračuna procenat masivno korišćenih paketa sa direktnim ili tranzitivnim zavisonstima, kao i procenat nestabilnih funkcionalnosti na kojima se ovakvi paketi zasnivaju [9].

Kapije funkcionalnosti su mehanizam na osnovu kog se kontroliše vidljivost funkcionalnosti u određenom skupu alata (*stable*, *beta*, *nightly*). Funkcionalnost može biti prihvaćena, nestabilna, nezavršena ili obrisana. Kapije funkcionalnosti se ne brišu fizički iz koda, već uz adekvatan opis služe kao perzistentno obrazloženje odluke da funkcionalnost nije podobna za razvoj.

Odobravanje funkcionalnosti je rigorozan ali transparentan proces. Svaka značajna promena koja nije refaktorizacija ili dokumentovanje mora proći kroz sledeće faze:

1. **Zahtev za komentare (RFC):** Proces započinje kreiranjem **RFC dokumenta** koji detaljno obrazlaže svrhu nove funkcionalnosti i njen opšti dizajn. Ovaj dokument je javan i podložan diskusiji od strane Rust tima i celokupne zajednice. Odobrenje RFC-a daje zeleno svetlo za početak razvoja, ali ne garantuje konačno prihvatanje.
2. **Razvoj i testiranje:** Nakon odobrenja, funkcionalnost se implementira i detaljno testira.
3. **Proces stabilizacije:** Kada je funkcionalnost razvijena i testirana bez značajnih primedbi, pokreće se formalni zahtev za stabilizaciju, koji se sastoji iz četiri ključna dela:
    - **Ažuriranje dokumentacije:** Dokumentacija se premešta iz interne "nestabilne knjige" (*Unstable Book*) u zvaničnu dokumentaciju za korisnike (*Rust Reference*).
    - **Stabilizacioni izveštaj:** Kreira se izveštaj koji sadrži primere korišćenja, linkove ka dokumentaciji i testove koji pokrivaju granične slučajeve.
    - **Period finalnog komentara (*FCP*):** Tim ponovo pregleda ceo predlog kako bi se postigao konačni konsenzus.
    - **Stabilizacioni *Pull Request*:** Ukoliko je konsenzus pozitivan, kreira se finalni

zahtev za povlačenjem (*pull request*). Njegov cilj je da tehnički omogući funkcionalnost u stabilnoj verziji jezika, uklanjajući oznaku nestabilnosti i grešku koja sprečava njeno korišćenje van noćne (*nightly*) verzije kompajlera.

4. **Beta i stabilna verzija:** Nakon uspešne stabilizacije, funkcionalnost postaje dostupna korisnicima u **beta verziji** za finalno testiranje, pre nego što konačno postane deo sledeće **stabilne distribucije** Rust-a.

4. ZAKLJUČAK

Međureprezentacije izvornog koda u prednjem kraju rustc kompajlera čine složen, ali visoko efikasan sistem. Kroz pažljivo dizajnirane faze transformacije i analize – od ASS-a, preko MVN-a i TMVN-a, do MSN-a – kompajler postepeno prevodi, proverava i optimizuje korisnički kod. Ova slojevita arhitektura je temelj na kojem Rust gradi svoje jedinstvene prednosti: garantovanu memorijsku bezbednost bez sakupljača smeća, konkurentnost bez rizika od "data races", i visoke performanse uporedive sa C i C++.

Svaka izmena ili proširenje *Rust* kompajlera podleže rigoroznom i transparentnom procesu prihvatanja. Kroz mehanizam Zahteva za komentare (RFC) i višefaznu stabilizaciju, zajednica osigurava da se jezik razvija na kontrolisan način, čuvajući integritet i bezbednosne garancije koje kompajler pruža.

Razumevanje ovih internih mehanizama i uloge svake međureprezentacije je od velikog značaja, ne samo za dalji razvoj i unapređenje samog kompajlera (npr. poboljšanje vremena kompajliranja ili uvođenje novih optimizacija), već i za dublje razumevanje ponašanja Rust programa i efikasnije korišćenje naprednih mogućnosti jezika.

5. LITERATURA

[1] MSRC, “A proactive approach to more secure code | MSRC Blog | Microsoft Security Response Center,” Microsoft.com, Jul. 16, 2019. https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/ (pristupljeno u septembru 2024.)

[2] “Parsing” Rochester.edu, 2024. https://www.cs.rochester.edu/u/nelson/courses/csc_173/grammars/parsing.html#:~:text=Recursive%2Ddescent%20parsing%20is%20one,non%2Dterminal%20with%20a%20procedure (pristupljeno u septembru 2024.)

[3] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2006.

[4] “Getting Started - Rust Compiler Development Guide,” Rust-lang.org, 2024. https://rustc-dev-guide.rust-lang.org/getting-started.html (pristupljeno u Septembru 2024.)

[5] “Introduction - The Rust Reference,” Rust-lang.org, 2015. https://doc.rust-lang.org/reference/introduction.html (pristupljeno u septembru 2024.)

[6] “Meet Safe and Unsafe - The Rustonomicon,” Rust-lang.org, 2024. https://doc.rust-lang.org/nomicon/meet-safe-and-unsafe.html (pristupljeno u septembru 2024)

[7] “What are editions? - The Rust Edition Guide,” Rust-lang.org, 2024. https://doc.rust-lang.org/edition-guide/editions/ (pristupljeno u oktobru 2024.)

[8] “The Unstable Book - The Rust Unstable Book,” Rust-lang.org, 2024. https://doc.rust-lang.org/unstable-book/index.html (pristupljeno u oktobru 2024.)

[9] Li, Chenghao, et al. “Demystifying Compiler Unstable Feature Usage and Impacts in the Rust Ecosystem.” 26 Oct. 2023. arXiv, (pristupljeno u oktobru 2024.)

[10] Bugden W, Alahmar A. Rust: The programming language for safety and performance. arXiv preprint arXiv:2206.05503. 2022 Jun 11.

[11] P. Mainardi, “The Rising Threat of Software Supply Chain Attacks: Managing Dependencies of Open Source projects,” Linuxfoundation.eu, Aug. 15, 2023. https://linuxfoundation.eu/newsroom/the-rising-threat-of-software-supply-chain-attacks-managing-dependencies-of-open-source-p (pristupljeno u oktobru 2024.)

[12] “The Architecture of Open Source Applications (Volume 1)LLVM,” Aosabook.org, 2024. https://aosabook.org/en/v1/llvm.html (pristupljeno u novembru 2024.)

[13] “The LLVM Compiler Infrastructure Project,” Llvm.org, 2024. https://llvm.org/ (pristupljeno u novembru 2024.)

Kratka biografija:

Aleksa Bajat rođen je 2001. godine u Novom Sadu. Završio je prirodno-matematički smer na engleskom jeziku u gimnaziji ”Jovan Jovanović Zmaj” 2019. godine. Tokom sve četiri godine gimnazije uspešno je pohađao ”Centar za mlade talente” kompanije Schneider Electric. Godine 2019. upisao je Fakultet Tehničkih Nauka u Novom Sadu, gde je ispunio sve obaveze i položio sve ispite predviđene studijskim programom sa prosečnom ocenom 9.03.
kontakt: aleksabajat15@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/33BE09Pavlovic

FAGL – JEZIK SPECIFIČAN ZA DOMEN IMPLEMENTACIJE VEB APLIKACIJA

FAGL – A DOMAIN-SPECIFIC LANGUAGE FOR WEB APPLICATION IMPLEMENTATION

Lazar Pavlović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

Kratak sadržaj – *U ovom radu opisan je razvoj FAGL jezika specifičnog za domen za opis i generisanje veb aplikacija. Rad pruža uvid u konkretnu sintaksu jezika, dizajn, arhitekturu i implementaciju ovog rešenja. Poređenje sa pet postojećih rešenja, omogućilo je kritičku evaluaciju mogućnosti i funkcionalnosti razvijenog FAGL jezika koja je pokazala da su postavljeni ciljevi rada ostvareni razvojem tekstualnog JSD-a koji je jednostavan za korišćenje, lak za učenje i sadrži sve neophodne funkcionalnosti.*

Ključne reči: *Jezik specifičan za domen, textX, Python, generator koda, veb aplikacija*

Abstract – *This work describes the development of the FAGL domain-specific language for the description and generation of web applications. The paper provides insight into the specific syntax of the language, its design, architecture, and implementation. A comparison with five existing solutions enabled a critical evaluation of the features and functionalities of the developed FAGL language, demonstrating that the goals of the study were achieved through the development of a textual DSL that is user-friendly, easy to learn, and includes all necessary functionalities.*

Keywords: *Domain specific language, textX, Python, code generator, web application*

1. UVOD

Kao odgovor na sve složenije zahteve tržišta i potrebe za što većom produktivnošću, raste potražnja za automatizovanim razvojem softverskih rešenja koja se mogu brzo i efikasno implementirati. Takođe, napredak u tehnologijama kao što su veštačka inteligencija, *Internet of Things* i *Cloud computing* stvara nove prilike i zahteve u razvoju inovativnih softverskih rešenja.

Računar izvršava zadatke na osnovu instrukcija zapisanih na programskom jeziku. JON imaju primenu u izradi softvera u različitim domenima primene. Domen se može posmatrati s dva aspekta: horizontalnog, koji se odnosi na tehničke aspekte sistema, i vertikalnog, koji obuhvata poslovne aspekte i specifične potrebe organizacije.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Igor Dejanović, red. prof.

Horizontalni ili tehnički domen može biti: bekend aplikacija, frontend aplikacija, bezbednost, baze podataka, blokčejn, obrada podataka, veštačka inteligencija, mobilne aplikacije. Vertikalni ili poslovni domen je oblast u kojoj se softver primenjuje: finansije, osiguranje, zdravstvo, administracija i mnogi drugi.

Potrebe za daljim povećavanjem efikasnosti i omogućavanjem programiranja ne-tehničkom osoblju kvalifikovanom za određene oblasti, dovele su do nastanka malih specijalizovanih programskih jezika – Jezika Specifičnih za Domen (JSD). Iako JON jezici nude veliku fleksibilnost programeru prilikom razvoja softvera, upotreba JSD-a za razvoj više sličnih softvera za isti domen (bilo horizontalni ili vertikalni) donosi brojne benefite: veću produktivnost programera, kvalitetnija rešenja opisana u manje linija programskog koda, manji broj bagova, bolji kvalitet programskog koda, bolje modeliranje složenih sistema [1].

Prema vrsti konkretne sintakse JSD se dele na grafičke, tekstualne i tabelarne [2]. U literaturi su opisani JSD-ovi različite kompleksnosti, fleksibilnosti i mogućnosti primene. Jednostavnija rešenja najčešće imaju ograničene funkcionalnosti za opis sistema, generisanje programskog koda i upravljanje korisnicima [2,3]. JSD koji nude veće mogućnosti za opisivanje sistema u kontekstu mikroservisnih aplikacija i poseduju generatore za više platformi često imaju složenu sintaksu, koja se znatno razlikuje od JON. Zbog toga njihova primena zahteva značajan angažman programera i dodatno vreme za prilagođavanje [1,4,5]. Bez obzira na nivo složenosti, većini JSD-ova nedostaju mogućnosti za validaciju vrednosti atributa [1,2,4,5], kao i podrška za automatsko generisanje frontend aplikacija [2,4,5].

Polazeći od prednosti i nedostataka tekstualnih JSD-ova opisanih u literaturi, cilj ovog rada bio je razvoj tekstualnog JSD-a koji je intuitivan za korišćenje, jednostavan za učenje i koji integriše sve neophodne funkcionalnosti za sveobuhvatno rešavanje problema.

2. ARHITEKTURA I DIZAJN REŠENJA

FAGL JSD opisan u ovom radu implementiran je korišćenjem *Python* programskog jezika i biblioteka: *io.StringIO*, *os*, *textX*, *jinja2*, i *click*. Projekat je razvijen na *Python* interpreteru, verzija 3.10.0, a virtuelno okruženje kreirano je pomoću alata *virtualenv*.

Arhitektura softvera zasniva se na modularnom pristupu, a glavni moduli su: model, parser, *checker*,

generator_config, *generator_backend* i *generator_frontend*.
Faze u radu programa su:

1. Parsiranje programa napisanog na FAGL jeziku
2. Provere nad učitanim modelom
3. Generisanje programskog koda

Konkretna sintaksa FAGL JSD-a definisana je pomoću metamodela koji uključuje semantičke informacije o njemu. Za definiciju metamodela korišćena je biblioteka *textX* [6]. Parsiranje programa napisanog na FAGL jeziku obavlja se korišćenjem *textX* biblioteke i gramatike, odnosno metamodela. Rezultat parsiranja programa je učitan program konvertovan na klase definisane u modulu Model. Provere nad učitanim modelom obezbeđuju adekvatnost ulaznog programa za generisanje validnog programskog koda na JON. U poslednjoj fazi, vrši se generisanje programskog koda, koje obuhvata generisanje programskog koda bekend i frontend aplikacije, kao krajnji rezultat rada programa.

Bekend aplikacija implementirana je na *Go* programskom jeziku, pomoću *generator_backend* generatora. Oslanja se na *Gin framework* za kreiranje aplikativnog servera, dok za rad sa *sqllite* bazom podataka koristi *GORM* Objektno Relacioni Maper. Generisana frontend aplikacija implementirana je pomoću *Vue3.js framework*-a. Generisanje aplikacija vrši se na osnovu ulaznih podataka i šablona definisanih upotrebom biblioteke *jinja2*. Komunikacija između frontend i bekend aplikacije realizovana je upotrebom REST arhitekturnog stila.

3. KONKRETNA SINTAKSA FAGL JEZIKA

FAGL omogućava definisanje programa u jednom fajlu, ali je zbog kvalitetnijeg programskog koda i veće čitljivosti moguće napisati programski kod u više fajlova, koje je potrebno importovati. U svakom fajlu definiše se naziv paketa, importi i elementi jezika. Glavni elementi konkretne sintakse FAGL jezika su: konstanta (*Constant*), entitet (*Entity*), enumeracija (*Enum*), uloga (*Role*), restrikcija (*Restriction*) i servis (*Service*), koji su u metamodelu definisani posebnim pravilima.

Constant je apstraktno pravilo za izbor *RegularConst* ili *GeneralConst* elementa. *RegularConst* se koristi za definisanje konstanti čije se vrednosti prikazuju na korisničkom interfejsu, dok se *GeneralConst* primenjuje za prikaz vrednosti konstanti sa predefinisanim imenima.

Enum element predstavlja tip podatka enumeracije sa predefinisanim vrednostima (literali), što omogućava definisanje i upravljanje kolekcijom imenovanih vrednosti, poboljšava čitljivost i konzistentnost programskog koda.

Entity element opisuje klasu modela i sadrži minimalno jedan atribut i reprezentaciju instance entiteta. Kako bi se postigla jednostavnost sintakse, uvedeno je pravilo prema kojem prvi atribut svakog *Entity* elementa mora imati ime *id* i biti tipa *uuid* ili *long*. Ovo ograničenje nije implementirano na nivou sintakse JSD-a, već je provera ovog pravila obavljena u *checker* modulu, koji osigurava njegovo sprovođenje. Atributi entiteta, odnosno uloga se prepoznaju *Attribute* pravilom, definisanim u gramatici jezika. Atributi su definisani tipom, kardinalitetom, imenom i opciono validatorskim funkcijama. Za validiranje vrednosti atributa u toku izvršavanja (engl. *run-time*) generisanog programa u trenutnoj implementaciji FAGL JSD-a postoji 26 različitih validatorskih funkcija, koje su u gramatici definisane pravilom *ValidationBlock*. Tip atributa definisan je apstraktnim pravilom *Type*, koje predstavlja izbor između *SimpleType* i *ComplexTypeReference* pravila. *SimpleType* predstavlja izbor između prostih tipova definisanih u jeziku: *uuid*, *string*, *integer*, *date*, *float*, *bool* i *long*. *ComplexTypeReference* je apstraktno pravilo koje sadrži referencu na kompleksni tip predstavljen *ComplexType* pravilom koje predstavlja izbor između prethodno definisanih entiteta, enumeracija ili uloga.

Role element koristi se za opis uloge korisnika sistema. Sintaksa ovog elementa slična je sintaksi *Entity* elementa. Predefinisana su imena prva tri atributa: *id* (*uuid* ili *long* tipa), *username* (tipa *string*) i *password* (tipa *string*). Nakon atributa navodi se reprezentacija instance uloge.

Restriction element predstavlja ograničenje prethodno definisanog entiteta. Ograničenje se ogleda u navođenju atributa čije vrednosti će korisnik moći da menja (ažurira). Ovaj element jezika koristi se prilikom definisanja servisnih metoda koje ažuriraju entitet (engl. *update*).

Service element predstavlja skup CRUD servisnih metoda za jedan prethodno definisan entitet. Servisna metoda definiše jednu od CRUD operacija, uloge autorizovane za izvršavanje metode, povratni tip metode i njegov kardinalitet, jedinstveno ime servisne metode, parametre metode tipa restrikcije ili jedinstvenog identifikatora (*ID*), a mogu da sadrže i poruke o grešci ili uspehu.

4. STUDIJA SLUČAJA

Mogućnosti FAGL JSD-a demonstrirane su na primeru sistema za inventuru pića. Sistem je opisan sa dve enumeracije, pet entiteta (klasa), dve uloge, četiri servisa i 12 generalnih i 59 regularnih konstanti. Enumeracije su *PakageStatus* i *TransactionType*. Enumeracija *PakageStatus* se koristi za opis statusa paketa i sadrži pet literala: *Available*, *Reserved*, *Expired*, *InConsuming* i *Consumed*. Enumeracija *TransactionType* se koristi za opis tipa transakcije i sadrži literale *In* i *Out*. Entiteti su *Category*, *DrinkType*, *Location*, *Pakage* i *InventoryTransaction*. Definisane uloge su *Worker* i *Admin*. Za entitete specificirani su servisi *DrinkTypeService*, *LocationService*, *PackageService* i *CategoryService*.

Rad je započet unosom opisa sistema na FAGL jeziku distribuiranim u šest fajlova. Komandom *fagl* definišu se putanje ulaza i izlaza koda i pokreće program, koji generiše programski kod. U cilju pokretanja bekend aplikacije manuelno su uklonjeni neupotrebljeni import iskazi i formatiran je generisani programski kod pomoću *goimports* alata, nakon čega su instalirane zavisnosti navedene u *go.mod* fajlu. Nakon pokretanja komande *go run server.go* generisana bekend aplikacija je dostupna na portu 8080. Instaliranjem zavisnosti navedenih u *package.json* fajlu pomoću *npm install* komande, ispunjen je uslov za pokretanje frontend aplikacije. Aplikacija se pokreće komandom *npm run dev*, nakon čega je dostupna na portu 5173. Pristup početnoj stranici frontend aplikacije moguć je pomoću veb pretraživača, na adresi *http://localhost:5173/*.

Programski kod sistema za inventuru pića na FAGL jeziku sadrži 269 linija. Generisana bekend aplikacija ima ukupno 2883 linija, a generisana frontend aplikacija 2469, što je ukupno 5352 linija. Jednoj liniji koda na FAGL JSD-u odgovara 19,9 linija generisanog programskog koda na JON jezicima *Go* i *JavaScript*. Ova metrika prikazuje veliku efikasnost i ubrzanje rada ostvareno upotrebom FAGL JSD-a.

5. POREĐENJE SA DRUGIM REŠENJIMA

Analizom dostupne literature identifikovano je pet relevantnih radova koji opisuju JSD slične namene: *MaGiC* [1], *CRUDyLeaf* [2], *DOMMLite* [3], *Silvera* [4] i *MicroBuilder* [5]. Svi ovi jezici kao i FAGL imaju tekstualnu konkretnu sintaksu. Za razvoj JSD-a (metaprogramiranje) koriste se različiti alati, biblioteke i platforme. Na primer, jezik *MaGiC* razvijen je korišćenjem alata *Jetbrains MPS*, dok su *CRUDyLeaf*, *DOMMLite* i *MicroBuilder* kreirani pomoću *Xtext*-a. Za razliku od njih, jezici *Silvera* i FAGL JSD, implementirani su pomoću biblioteke *textX*, koja je inspirisana alatom *Xtext*.

Iako svi navedeni JSD imaju sličan domen, značajno se razlikuju po svojim mogućnostima i načinu modelovanja. Na primer, *MaGiC* koristi tri različita JSD-a za specifične celine: mikroservisnu backend aplikaciju, klijentsku aplikaciju i jezik za definisanje komunikacije između ove dve aplikacije (eng. *gateway*). *Silvera* i *MicroBuilder* su ograničeni na opis mikroservisnog backenda i ne podržavaju modelovanje klijentskih aplikacija. S druge strane, FAGL JSD, razvijen u ovom radu, dizajniran je za modelovanje monolitne backend aplikacije. Po svojim mogućnostima sličan je jezicima *CRUDyLeaf* i *DOMMLite*, ali istovremeno nudi unapređene funkcionalnosti uz veću jednostavnost korišćenja.

Generisanje programskog koda može se realizovati na više načina. Autori jezika *MaGiC* koristili su *Plaintextgen*, M2T generator, za kreiranje koda svih komponenti aplikacije. Za generisanje koda jezici *CRUDyLeaf* i *MicroBuilder* oslanjaju se na biblioteku *Xtend*, dok *DOMMLite* koristi specijalizovani jezik *Xpand* za definisanje šablona prema kojima se generiše kod. Za jezik *Silvera*, navedeno je da se generisanje koda zasniva na M2T transformacijama i korišćenju šablona za generisanje *Java* koda, ali nije precizirano koja je biblioteka korišćena za ovu svrhu. Za razliku od njih, FAGL JSD koristi M2T transformaciju pomoću *jinja2* biblioteke, čime se postiže jednostavnost i fleksibilnost u radu sa šablonima.

Generatori na osnovu modela generišu programski kod na JON jezicima, koji se može izvršavati. Kako postoji više različitih JON jezika, jedno rešenje može da sadrži generatore za više JON jezika. Iako je za osnovnu funkcionalnost dovoljan samo jedan, prisustvo generatora za više jezika povećava fleksibilnost i kvalitet rešenja. Rešenja kao što su *MaGiC* i *Silvera* sadrže generatore za različite JON jezike. Na primer, *MaGiC* nudi tri generatora, po jedan za svaki JSD. Bekend aplikacije (mikroservisi) i *gateway* generišu se u *Node.js* korišćenjem *Express* frejmvorka ili u *Python*-u sa *Flask* frejmvorkom, dok se frontend aplikacija generiše korišćenjem *React* frameworka. Sa druge strane, jezik *Silvera* uključuje generator za *Java* programski kod koji koristi *Spring Boot* frejmvork. Dodatno, zahvaljujući modularnoj arhitekturi, razvijeni su generatori i za druge JON jezike, poput *Python*-a, *C#* i *Go*.

Ostala rešenja analizirana u ovom radu implementiraju generatore samo za jedan JON jezik. Na primer, generator *CRUDyLeaf*-a generiše bekend aplikaciju na *Java* programskom jeziku koja koristi *Spring Boot* frejmvork. *DOMMLite* generiše bekend aplikaciju u *Python* programskom jeziku, koja koristi *Django* frejmvork, dok *MicroBuilder* generiše *Java* programski kod koji se oslanja na *Spring*, *Spring Cloud* i *NetflixOSS* frejmvorke. FAGL razvijen u ovom radu, ima dva generatora: jedan za monolitnu bekend aplikaciju na *Go* programskom jeziku i drugi za frontend aplikaciju na *JavaScript* jeziku.

Osim *MaGiC* i FAGL rešenja, ostala razmatrana rešenja nemaju implementirane generatore za frontend aplikacije. Ovaj nedostatak generatora za frontend aplikacije predstavlja značajan problem, jer iako se generisanje bekend aplikacija može automatizovati, nije moguće automatski generisati frontend kod, koji omogućava interakciju sa aplikacijom krajnjim korisnicima, posebno onim koji nisu tehnički osposobljeni. Iako *DOMMLite* ne sadrži generator za frontend aplikaciju, zbog korišćenja *Django* frejmvorka, bekend aplikacija generiše administrativni interfejs koji se može koristiti za upravljanje aplikacijom tokom razvoja ili nakon isporuke krajnjim korisnicima. Međutim, ovo rešenje nije potpuno, jer izmene u dizajnu ili funkcionalnostima frontend aplikacije postaju znatno složenije i zahtevaju dodatno manuelno podešavanje, što može biti prepreka za brzu kastomizaciju i razvoj.

JSD-ovi *CRUDyLeaf, Silvera* i *MaGiC* omogućavaju automatsko generisanje dokumentacije. *CRUDyLeaf* u te svrhe koristi *OpenAPI* i *Swagger*, a *Silvera* generiše dokumentaciju zasnovanu na *OpenAPI* specifikaciji. *MaGiC* ide korak dalje, jer osim generisanja dokumentacije pomoću *SwaggerUI*, omogućava i generisanje infrastrukture za kontejnerizaciju i skripti koji poboljšavaju korisničko iskustvo prilikom upotrebe alata. S druge strane, FAGL slično *DOMMLite* i *MicroBuilder*, ne podržava generisanje dokumentacije i infrastrukture za kontejnerizaciju.

FAGL kao i svi razmatrani JSD-ovi podržava CRUD operacije. *MaGiC* pored toga nudi funkcionalnosti za definisanje specifičnih operacija poput *GetEntitiesBy* i *GetEntity*. *DOMMLite* i *MicroBuilder* omogućavaju pretragu entiteta pomoću operacije *Search*, što je naročito korisno u scenarijima kada se obim podataka u sistemu značajno poveća. *DOMMLite* i *Silvera* omogućavaju definisanje metoda koje nemaju unapred definisano ponašanje, što uvećava mogućnost operacija. U tom slučaju nakon generisanja programskog koda, korisnik manuelno implementira metode. U *MicroBuilder*-u, operacija *create* je preimenovana u *insert*, a operacija *read* zamenjena je funkcionalnošću *search*. Svi analizirani jezici omogućavaju ažuriranje vrednosti svih atributa entiteta, ali ne pružaju mogućnost definisanja atributa čije vrednosti se mogu izmenjeni. Ovaj nedostatak uspešno je rešen u FAGL JSD-u, uvođenjem restrikcija nad entitetima, što omogućava preciznu kontrolu promena nad podacima.

Nijedan od prethodno analiziranih jezika ne nudi mogućnost definisanja korisnika sistema i opisivanja

dozvola (permisija) koje regulišu pristup određenim metodama. Za razliku od njih, FAGL JSD omogućava obe funkcionalnosti, čime značajno pojednostavljuje rad korisnicima, eliminiše potrebe za naknadnim izmenama generisanog koda radi implementacije autorizacije i omogućava preciznije i efikasnije opisivanje sistema.

Još jedna slabost većine analiziranih JSD-ova je odsustvo podrške za validacione funkcije, koje su od ključnog značaja za proveru ispravnosti podataka unetih u sistem. Ipak treba istaći da su validacione funkcije implementirane u FAGL-u kao i u *DOMMLite*-u, čime oba jezika omogućavaju kontrolu i pouzdanost prilikom obrade podataka.

Sintakse analiziranih rešenja značajno se razlikuju po složenosti. *CRUDyLeaf* se ističe izuzetno jednostavnom sintaksom koja omogućava brzo usvajanje i skraćuje vreme razvoja softvera. Međutim, ova jednostavnost dolazi sa ograničenjima — *CRUDyLeaf* ne podržava opis mikroservisne arhitekture, što ga čini neprikladnim za kompleksnije projekte. *MicroBuilder*, takođe nudi jednostavnu sintaksu, koja za razliku od *CRUDyLeaf*-a omogućava opisivanje mikroservisne arhitekture. Ipak, njegova sintaksa za definisanje tipova atributa (npr. „%Single String%“) nije intuitivna, a entiteti se mogu definisati samo unutar mikroservisa, što otežava rad sa većim modelima i može rezultirati nepreglednim kodom. *Silvera* JSD ima složeniju sintaksu koja je pogodna za iskusne programere upoznate sa mikroservisnom arhitekturom i njenim mogućnostima. Suprotno ovim jezicima, *MaGiC* ima kompleksniju sintaksu, koja više podseća na govorni jezik nego na standardne programske jezike. Pored toga, korisnici *MaGiC*-a moraju da savladaju čak tri različita JSD-a kako bi opisali ceo sistem, što predstavlja značajnu prepreku za njegovu primenu. Sintaksa FAGL jezika podseća na JON i po jednostavnosti i preglednosti slična je sintaksi *DOMMLite*-a. Definisanje entiteta i CRUD operacija u FAGL jeziku sintaksno je najsličnije onome opisanom u *Silvera*-i.

Analizirana rešenja pružaju različite nivoe udobnosti za programere. Rad sa *MaGiC*-om je zahtevan jer uključuje instalaciju *MPS* softvera, nakon čega sledi čak 17 koraka kako bi se kreirala najjednostavnija aplikacija. Autori *CRUDyLeaf*, *DOMMLite* i *MicroBuilder*-a preporučuju rad u okviru *Eclipse* alata, koji je poznat po brojnim dokumentovanim nedostacima i često izaziva frustracije kod korisnika [7]. S druge strane, rešenja *Silvera* i FAGL su značajno jednostavnija za upotrebu. Ova rešenja omogućavaju programerima da koriste bilo koji tekstualni editor po sopstvenom izboru, čime se eliminiše potreba za specijalizovanim softverom. Nakon instalacije u *Python* virtuelno okruženje, FAGL se slično *Silvera*-i pokreće intuitivno i brzo.

6. ZAKLJUČAK

Rad pruža uvid u tehničke mogućnosti FAGL jezika specifičnog za domen, koji uključuje dva generatora programskog koda. Detaljno su analizirani konkretna sintaksa jezika, kao i dizajn, arhitektura i implementacija ovog rešenja. Posebna pažnja posvećena je definisanju atributa čije vrednosti korisnik može da ažurira, specifikaciji korisnika sistema i implementaciji autorizacije za pristup CRUD operacijama nad entitetima. Rezultati rada prikazani su studijom slučaja, koja ilustruje proces opisivanja sistema i njegovog generisanja na konkretnom primeru. Takođe, izvršeno je poređenje razvijenog rešenja sa pet postojećih JSD-ova, što je omogućilo kritičku evaluaciju mogućnosti i funkcionalnosti razvijenog FAGL jezika. Na osnovu ove analize, može se zaključiti da su postavljeni ciljevi rada ostvareni razvojem tekstualnog JSD-a koji je jednostavan za korišćenje, lak za učenje i sadrži sve neophodne funkcionalnosti. Fleksibilnost i lakoća korišćenja čine FAGL pogodnim izborom, posebno za programere koji traže efikasne alate bez suvišnih komplikacija.

Ovaj rad postavio je temelje za dalje unapređenje ovog rešenja, razvojem generatora programskog koda za više JON jezika, uvođenjem koncepata mikroservisne arhitekture i dodavanjem novih funkcionalnosti kao što je pretraga. Takođe, implementacija alata za generisanje dokumentacije i infrastrukture za kontejnerizaciju otvorila bi mogućnosti za dalja poboljšanja, čineći rešenje još efikasnijim i fleksibilnijim.

7. LITERATURA

[1] A. Bucchiarone, C. Ciumedean, K. Soysal, N. Dragoni, and V. Pech, "MaGiC: a DSL framework for implementing language-agnostic microservice-based web applications," *Journal of Object Technology*, vol. 22, no. 1, 2023.

[2] O. S. Gómez, R. H. Rosero, and K. Cortés-Verdín, "CRUDyLeaf: a DSL for generating Spring Boot REST APIs from entity CRUD operations," *Cybernetics and Information Technologies*, vol. 20, no. 3, pp. 3–14, 2020.

[3] I. Dejanović, G. Milosavljević, B. Perišić, and M. Tumbas, "A domain-specific language for defining static structure of database applications," *Computer Science and Information Systems*, vol. 7, no. 3, pp. 409–440, 2010.

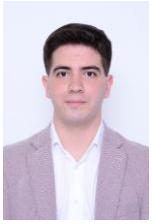
[4] A. Suljkanović, B. Milosavljević, V. Inđić, and I. Dejanović, "Developing microservice-based applications using the silvera domain-specific language," *Applied Sciences*, vol. 12, no. 13, p. 6679, 2022.

[5] B. Terzić, V. Dimitrieski, S. Kordić, G. Milosavljević, and I. Luković, "MicroBuilder: a model-driven tool for the specification of REST microservice architectures," in *Proc. Int. Conf. Inf. Soc. Technol.*, 2017, pp. 179–184.

[6] I. Dejanović, R. Vaderna, G. Milosavljević, and Ž. Vuković, "TextX: A Python tool for Domain-Specific Languages implementation," *Knowledge-Based Systems*, vol. 115, pp. 1-4, 2017.

[7] https://medium.com/@ashaytikekar/why-eclipse-sucks-dd70e572c675 (pristupljeno u novembru 2024.)

Kratka biografija:



Lazar Pavlović rođen je u Požarevcu 2001. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Softversko inženjerstvo i informacione tehnologije odbranio je 2024.god.
kontakt: lp.pavlovic.001@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE10Ubiparip

PRIMENA LINEARNOG PROGRAMIRANJA U OPTIMIZACIJI PLANIRANJA I UPRAVLJANJA PROJEKTIMA

THE APPLICATION OF LINEAR PROGRAMMING IN THE OPTIMIZATION OF PROJECT PLANNING AND MANAGEMENT

Jelena Ubiparip, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad je u potpunosti posvećen optimalnom upravljanju projektima posebno iz oblasti upravljanja IT projektima uz oslonac na metodologiju i tehnike linearnog programiranja. Da bismo upotpunili rad i upotrebu projekta optimizacionog softvera učinili dostupnom, razvili smo jedno veb-bazirano rešenje koje omogućuje optimalno planiranje svih procesa u projektu.*

Ključne reči: *Linearno programiranje, Veb aplikacija, Naučni menadžment, Projektni menadžment*

Abstract – *This paper is fully dedicated to optimal project management, especially in the field of IT project management, based on the methodology and techniques of linear programming. In order to make the use of the optimization software project available, we have developed a web-based solution that enables optimal planning of all processes in the project.*

Keywords: *Linear programming, Web application, Scientific management, Project management*

1. UVOD

Uspeh i opstanak svake organizacije zavisi od kvaliteta odluka koje donose njeni menadžeri. Osnovno pitanje u mnogim problemima koji se javljaju u poslovanju i industriji je kako raspodeliti ograničene resurse na različite aktivnosti. Odgovornost menadžera (donosioca odluke) je da rasporedi resurse na određene aktivnosti kako bi se postigli najbolji rezultati za organizaciju. Iz tog razloga, razvijeni su alati i tehnike koje mogu pomoći u potrazi za optimalnim raspodelom resursa.

Upravo iz tog razloga je razvijena aplikacija „Upravljanje projekta“. Pre svega, korisniku je omogućeno da unosi podatke o zadacima koje je potrebno izvršiti tokom trajanja projekta na lak način i nakon toga sledi računanje najbolje putanje toka projekta. Odnosno, koliko bi se cena projekta mogla uvećati ukoliko se redukuje vreme trajanja projekta. Osim toga, određuje se i u kom trenutku je najbolje započeti određeni zadatak.

Bilo bi pogrešno smatrati da ova aplikacija oslobađa korisnika od donošenja odluka. Ona mu samo pomaže u procesu odlučivanja, dajući perspektivu odluke koju treba da donese.

NAPOMENA:

Ovaj rad proistekao je iz master rada čiji mentor je bio dr Željko Kanović, red. prof.

Kada se ova komponenta (rezultati) doda kvalitativnoj, iskustvenoj i intuitivnoj – odluka korisnika nesumnjivo postaje bolja.

2. OPERACIONA ISTRAŽIVANJA I PRIMENA LINEARNOG PROGRAMIRANJA U OPTIMIZACIJI

Osnovna karakteristika savremenog društva je postojanje velikog broja složenih sistema, kao što su tehnološko-proizvodni sistemi, energetski sistemi, komunikacioni sistemi, transportni sistemi, razni sistemi u poljoprivredi, industriji, i dr.

U ovakvim uslovima, upravljanje se mora vršiti metodično, sa programom koji podrazumeva planiranje odluka, odnosno izrada plana i upravljanje realizacijom plana.

Operaciona istraživanja su nastala upravo na tim temeljima i zahtevima sa ciljem da primenom postojećih i razvojem novih naučnih metoda na kvantitativnim osnovama daju odgovore po pitanju najboljeg funkcionisanja složenih sistema u postojećim uslovima. Danas postoji čitav niz raznih naučnih metoda i tehnika koje su razvijene i koje imaju primenu rešavanja raznih problema upravljanja sistemima.

Metode i tehnike operacionih istraživanja su prvenstveno namenjene donosiocima odluka (menadžeri), ali i ekspertima i specijalistima koji dobro poznaju prirodu problema koji se rešava i koji, zajedno sa menadžerima, trebaju biti uključeni u ceo proces donošenja optimalnih odluka.

2.1. Funkcija cilja ili kriterijum upravljanja

U procesima upravljanja postoje tri jedinstvena pojma sa kojima se često operiše. Ti pojmovi naročito dolaze do izražaja kod primene matematičkih metoda u izučavanju procesa upravljanja, a to su funkcija cilja (kriterijum upravljanja), skup ograničenja i matematički model.

Nijedan upravljački zadatak ne može da postoji bez definisanog cilja. Drugim rečima, bez definisane funkcije cilja nemoguće je ostvariti konkretno upravljanje.

Po svom karakteru i sadržaju ciljevi mogu biti veoma različiti. U odnosu na definisani cilj meri se kvalitet upravljanja i vrši komparacija u procesu odabiranja najbolje ili zadovoljavajuće upravljačke akcije.

U matematičkom smislu funkcija cilja izražava se nekom funkcijom $F(X)$ gde je $X=(x_1,x_2\ldots,x_n)$ n-dimenzioni vektor, pri čemu treba odrediti njen minimum ili maksimum. Drugim rečima, funkcija cilja predstavlja funkciju više

promenljivih za koju je potrebno odrediti ekstremnu vrednost.

Funkcija cilja se minimizira ukoliko su njome izraženi troškovi, vreme realizacije zadatka, utrošak materijala, gubitak, škart u proizvodnji, vreme transporta, itd.

Funkcija cilja se maksimizira ako odražava dobit, prinos po jedinici površine, dohodak po glavi stanovnika, učinak, itd. Svaki konkretni zadatak zahteva poseban pristup u realizaciji funkcije cilja, čiji oblik zavisi od specifičnih uslova zadatka.

Optimalno ostvarivanje cilja, posredstvom traženja ekstremne vrednosti funkcije *F(X)*, u realnim uslovima ne može se ni zamisliti bez skupa ograničenja, koja karakterišu granice potencijalnih mogućnosti određenog sistema. Svaki organizacioni sistem karakterisan je brojnim ograničenjima, čija se priroda najčešće vezuje za različite kategorije ograničenih resursa (radna snaga, mašine, materijal, novčana sredstva, prostor, itd.). Pored navedenog skupa ograničenja postoji još i prirodni skup ograničenja koji se sastoji u tome da komponente vektora *X* moraju biti nenegativne veličine $x_i \geq 0$, $(i=1,2,\ldots,n)$.

Metode odabiranja upravljačkih odluka koje se zasnivaju na angažovanju matematičkih modela pružaju mogućnost sveobuhvatnijeg sagledavanja problema, boljeg iskorišćenja resursa, bržeg obavljanja procesa, smanjenje troškova i povećanje efikasnosti. Vrednost upravljačke odluke, dobijene pomoću matematičkog modela zavisi od adekvatnosti modela sa procesom koji se izučava.

Matematički model koji realno odražava sistem koji se istražuje omogućava da se pronađu uticaji upravljačkih parametara na izmenu karakteristika sistema u cilju postizanja uslova njegovog optimalnog funkcionisanja.

2.2. Linearno programiranje

Zadaci koji se sa aspekta odgovarajućeg matematičkog modela M svode na linearno programiranje, karakterišu se funkcijom cilja *F(X)* koja predstavlja linearnu kombinaciju nepoznatih, kao i skupom ograničenja koji se zadaje sistemom linearnih jednačina ili nejednačina.

Ako se sa stanovišta matematičkog modela osvrnemo na linearno programiranje, problem se sastoji u tome kako naći minimum ili maksimum jedne linearne funkcije *F(X)*, pri određenom skupu ograničenja L zadanih linearnim vezama. Broj nepoznatih i ograničenja može da bude veoma različit.

Linearno programiranje predstavlja metodu određivanja takve kombinacije uzajamno povezanih faktora, koja od niza mogućih kombinacija predstavlja najpovoljniju. Drugim rečima, traži se takva kombinacija koja će, pored toga što će zadovoljiti data ograničenja, dati kriterijumu optimalnosti najbolju moguću (optimalnu) vrednost.

Opšta matematička formulacija zadatka linearnog programiranja sastoji se u sledećem:

Treba odrediti takav skup vrednosti $x_1,x_2,\ldots,x_n$, tj. komponenata n-dimenzinog vektora $X=(x_1,x_2,\ldots,x_n)$ iz oblasti koja je zadata sistemom linearnih jednačina

$$\sum_{k=1}^{n} a_{ik}x_k + b_i \geq 0, \quad (\mathrm{i} = 1,2,\ldots m) \qquad (1)$$

$$x_k \geq 0, \quad (k = 1,2,\ldots,n) \qquad (2)$$

za koje funkcija cilja

$$F(x) = F(x_1, x_2, \ldots, x_n) = \\ = C_1x_1 + C_2x_2 + \cdots + C_nx_n \qquad (3)$$

koja predstavlja linearnu kombinaciju nepoznatih x_k dostiže maksimalnu (minimalnu) vrednost.

Različite metode rešavanja zadataka linearnog programiranja imaju niz specifičnosti u pretraživanju mogućih varijanti plana u cilju izbora optimalnog plana. Pored toga, postoje opšte metode koje omogućavaju da se nađe rešenje bilo kog zadatka linearnog programiranja. U takve metode spada Dantzig-ova metoda koja je više poznata kao simpleks metoda kao i njene modifikacije razvijene vremenom.

2.3. Simpleks metoda

Ideja simpleks metode sadrži tri bitna elementa:

1. mogućnost određivanja bar jednog dopustivog plana X
2. mogućnost provere da li je određeni dopustivi plan X optimalan ili ne
3. mogućnost da se u slučaju izbora dopustivog plana X koji nije optimalan odredi novi plan koji je bliži optimalnom

U slučaju da postoje sve tri ove mogućnosti, može se u okviru konačnog broja koraka (računski korak u iteracionom procesu traženja rešenja) dobiti optimalni plan koji predstavlja rešenje formulisanog zadatka. Prema tome, simpleks metoda zasniva se na sukcesivnom poboljšanju dopustivog plana, sve dok se ne dobije optimalan plan. Ovakav prilaz u formiranju algoritma simpleks metode, takođe omogućava da se u procesu rešavanja bilo kog zadatka ustanovi da li je on rešiv ili ne, što znači da postoji mogućnost ispitivanja postojanja protivrečnosti u ograničenjima i ispitivanje da li je funkcija cilja *F(X)* neograničena u oblasti.

2.4. Optimizacioni proces u Phyton-u

Korišćenje Python-a za linearno programiranje obuhvata nekoliko ključnih koraka:

1. Definisanje problema
2. Definisanje promenljivih
3. Definisanje funkcije cilja
4. Definisanje ograničenja
5. Rešavanje problema
6. Prikaz rezultata

Cilj definisanja problema je da se dođe do formalnog opisa modela. Uzimajući u obzir dostupne podatke, često problem može da bude mnogo zahtevniji nego što je potrebno. Iz tog razloga je potrebno dobro proučiti koji su podaci zaista potrebni jer ograničenja dostupnih podataka mogu značajno promeniti opis modela i kasniju formulaciju.

Za rešavanje problema u Python-u najčešće se koriste biblioteke „Scipy“ i „PuLP“. Prednost biblioteke „PuLP“ nad „Scipy“ je u tome što omogućava lakše definisanje i rešavanje problema linearnog programiranja.

„PuLP“ je, kao što je navedeno, biblioteka koja se koristi za rešavanje problema optimizacije, posebno u kontekstu linearnog programiranja i celobrojnog programiranja. Ova biblioteka je vrlo korisna iz nekoliko razloga:

- Podrška za različite vrste optimizacije - linearno programiranje, celobrojno programiranje, mešovito celobrojno programiranje
- Kompatibilnost sa različitim solverima – podržava širok spektar solvera za rešavanje optimizacionih problema kao što su COIN-OR CBC, GLPK, CPLEX i Gurobi
- Prilagodljivost – omogućava korisnicima da kreiraju različite vrste problema, bilo da se radi o logistici, proizvodnji, finansijama, planiranju…
- Integracija sa drugim Python alatima – lako se integriše sa drugim alatima i bibliotekama kao što su Pandas (za rad sa podacima), NumPy (za numeričke operacije), i mnoge druge.

3. APLIKACIJA „UPRAVLJANJE PROJEKTOM“

Aplikacija “Upravljanje Projektom” je osmišljena tako da unos projektnih zadataka kao i podataka o samom zadatku budu jednostavni za korišćenje. Cilj aplikacije je da upotrebu projekta optimizacionog softvera učini dostupnom.

Zamisao je da aplikacija bude veb-bazirana, što omogućava lakši pristup i upotrebu sa različitih uređaja. Aplikacija nije namenjena svim zaposlenima, već isključivo menadžerima, kako bi imali uvid u ključne zahteve za izvršavanje projekta. Pored toga, aplikacija, koristeći prethodno objašnjeno linearno programiranje, izračunava najbolju putanju toka projekta, odnosno određuje optimalne trenutke za započinjanje svakog zadatka.

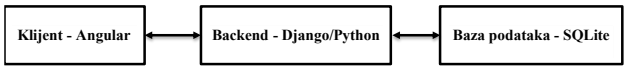
1. Ova aplikacija je veb aplikacija a za njeno razvijanje je korišćeno:
2. SQLite – ugradna baza podataka koji koristi standardni SQL jezik za upravljanje podacima.
3. Angular - open-source framework za izgradnju veb aplikacije.
4. Python – korišćen je za proces optimizacije.
5. Django veb okvir – koristi Phython za razvoj veb aplikacija.

Na Slici 1. je prikazan primer projekta sa manjim brojem zadataka, u cilju lakšeg objašnjenja i razumevanja.

Slika 1. *Aplikacija „Upravljanje Projektom“*

Aplikacija „Upravljanje Projektom“ razvijena je kao veb-bazirana aplikacija koja koristi modernu arhitekturu klijent-server modela. Ova arhitektura omogućava efikasnu obradu podataka, brzu interakciju sa korisnicima i skalabilnost sistema.

Dijagram arhitekture aplikacije dat je na Slici 2.



Slika 2. *Arhitektura aplikacije*

4. PRIMENA – PLANIRANJE I UPRAVLJANJE PROJEKTIMA

U radu je iskorišćeno prethodno opisano softversko rešenje, koje je primenjeno na konkretne podatke iz kompanije „Yris Solutions“ sa ciljem analize mogućnosti optimizacije vremenskih i resursnih parametara. Projekat čiji se podaci koriste u radu predstavlja skup zadataka koje je neophodno rešiti da bi se projektovala a posle i proizvela štampana ploča (engl. *Printed Circuit Board - PCB*).

U podacima koji su dostavljeni od strane kompanije svaki zadatak ima 8 obeležja:

1. ID (jedinstveni broj)
2. Ime (skraćeni opis zadatka)
3. Opis (detaljan opis zadatka)
4. Od kog prethodnog zadatka zavisi (ne može početi izvršavanje zadatka dok se navedeni zadatak ne izvrši)
5. Trajanje izvršavanja zadatka
6. Cena
7. Ubrzano izvršavanje zadatka
8. Cena ubrzanog izvršavanja zadatka

Cilj je odrediti koje zadatke treba vremenski skratiti, uzimajući u obzir da je cena ukoliko se zadatak skrati veća. „Ubrzano izvršavanje zadatka“ određuje zaposleni koji izvršava zadatak i predstavlja najkraće vreme u kojem zadatak može biti uspešno završen ako se poveća broj resursa, odnosno ako zaposleni bude u potpunosti posvećen samo tom zadatku.

Cena se povećava ukoliko se vreme izvršavanja zadatka skrati, jer ubrzavanje procesa obično zahteva dodatne resurse, kao što su dodatni radnici, rad u vanrednim uslovima ili korišćenje specijalizovane opreme. Ove dodatne potrebe povećavaju ukupne troškove projekta, što dovodi do viših cena u slučaju skraćivanja vremena izvršenja zadatka.

4.1. Određivanje kriterijuma optimalnosti na osnovu podataka

Da bi formulisanje funkcije bilo jednostavnije objašnjeno, upotrebićemo uopštenu formulaciju problema koja je prikazana u Tabeli 1.

ID Zadatka	Zavisi od zadatka	Trajanje [T]	Cena [C]	Ubrzano trajanje [TU]	Cena ubrzanog trajanja [CU]
A		TA	CA	TUA	CUA
B	A	TB	CB	TUB	CUB
C	B	TC	CC	TUC	CUC
n	n-1	Tn	Cn	TUn	CUn

Stvarno trajanje zadatka definiše se kao standardno trajanje umanjeno za vremenski period za koji je moguće njegovo skraćenje:

$$Dj = Tj - xj \qquad (4)$$

Tj predstavlja standardno vreme trajanja zadatka, odnosno vreme koje je potrebno da se zadatak završi bez ikakvog ubrzanja; *xj* predstavlja broj nedelja (ili bilo koja druga jedinica vremena) koliko se trajanje zadatka može skratiti ako se ulože dodatni resursi; *j* je ID zadatka.

Dakle, formula (4) predstavlja novo trajanje zadatka nakon što je njegovo standardno vremensko razdoblje *Tj* skraćeno

za *xj*. Ako uzmemo uopštene vrednosti iz tabele gore, trajanje zadatka je:

$$B = TB - xB \quad (5)$$

TB predstavlja standardno vreme trajanja zadatka B, odnosno vreme koje je potrebno za izvršavanje zadatka B bez bilo kakvog ubrzanja; *xB* predstavlja vreme skraćenja za zadatak B, tj. vreme koje se oduzima od standardnog vremena *TB* kako bi se skratilo trajanje zadatka B kroz dodatne resurse ili ubrzanje bilo koje vrste; *B* je novo vreme trajanja zadatka B nakon što se primeni skraćivanje *xB*.

Veza između zadatka C i zadatka B od kojeg zadatak C zavisi je takva da početak zadatka C (*yC*) ne može nikako biti pre nego što zadatak B završi, tj:

$$yC = yB + TB - xB \quad (6)$$

yC predstavlja novo vreme ili trenutak kada će zadatak C biti završen; *yB* predstavlja vreme ili trenutak kada je zadatak B završen; *TB* i *xB* su prethodno objašnjeni.

Na osnovu prethodno definisanih veličina, funkcija kriterijuma optimalnosti, čija se minimalna vrednost traži, formuliše se na sledeći način:

$$minZ = \sum (CUi - Ci) \cdot (Tj - Dj) \quad (7)$$

za $i = 1,\ldots,n$. U ovom izrazu, Z označava ukupni trošak koji treba minimizirati. Parametar *CUi* predstavlja cenu ubrzanog izvršenja zadatka, odnosno dodatni trošak koji nastaje kada se vreme trajanja zadatka smanji, dok je *Ci* standardna cena izvršenja zadatka. Veličine *Tj* i *Dj* označavaju redom standardno i stvarno (ubrzano) trajanje zadatka.

Konkretizacijom opšte funkcije kriterijuma optimalnosti date u formuli (7), dobija se sledeća funkcija za slučaj razmatranog projekta:

$$\begin{aligned} Z = {} & 1000x1 + 600x2 + 800x5 + 1200x7 \\ & + 106{,}67x6 + 106{,}67x8 + 160x9 + 160x10 \\ & + 160x11 + 177{,}77x12 + 145{,}45x13 \\ & + 160x14 + 160x15 + 200x16 + 400x17 \\ & + 400x18 + 177{,}78x19 + 80x20 + 80x21 \\ & + 80x22 + 160x23 + 228{,}57x24 + 160x25 \\ & + 533{,}33x26 + 160x27 + 400x28 \\ & + 320x29 + 333{,}33x30 + 320x31 \\ & + 320x32 + 320x33 + 320x34 + 320x35 \\ & + 400x36 + 400x37 + 320x38 + 320x39 \\ & + 733{,}33x4 + 160x40 + 160x41 + 160x42 \\ & + 160x43 + 160x44 + 160x45 + 160x46 \\ & + 160x47 + 80x48 + 1200x49 + 1600x50 \\ & + 17542{,}86x51 + 500x5 \end{aligned} \quad (8)$$

Pored funkcije cilja, model uključuje i sledeća ograničenja:

1. Ograničenja maksimalne redukcije vremena

Za svaki zadatak unapred je određeno koliko njegovo trajanje može maksimalno da se skrati. Na primer:

$x10 \leq 1$, $x11 \leq 1$, $x12 \leq 0.9$, ..., $x9 \leq 1$

2. Uslov nenegativnosti

Sve promenljive koje predstavljaju trajanja i vremenske pomake zadataka moraju biti nenegativne:

$xi \geq 0$, $yi \geq 0$, za $i = 1, \ldots, 52$.

3. Vremenska ograničenja (zavisnost između zadataka)

Redosled izvršavanja zadataka uslovljen je logičkim zavisnostima. Na primer, ako zadatak *y10* ne može da počne dok se ne završi zadatak *y9*, to se izražava nejednačinom : $y9 + y10 - x9 \geq 15$

5. DISKUSIJA REZULTATA

Na osnovu analize ključnih zadataka projekta, procenjeno je da će njegovo trajanje biti 242 nedelje. Ovaj rezultat pokazuje ukupan vremenski okvir potreban za završetak svih zadataka, uzimajući u obzir njihove međusobne zavisnosti i raspored. Ipak, postoji mogućnost optimizacije tog vremena.

Ukoliko bi se maksimalno smanjio vremenski period izvršavanja projekta na 228 nedelja, uz odgovarajuće povećanje resursa i ubrzanje određenih zadataka, troškovi skraćivanja vremena izvršavanja bi bili 44.325,81. Ovaj iznos predstavlja dodatne troškove koji bi bili neophodni za ubrzanje izvršenja, uključujući povećanje broja radnih sati, angažovanje dodatnih resursa i primenu tehnologija koje omogućavaju brže završavanje zadataka.

6. ZAKLJUČAK

Ovaj rad istražuje primenu linearnog programiranja u optimizaciji planiranja i upravljanja projektima, sa posebnim fokusom na IT projekte i projekte proizvodnje štampanih ploča. Kroz analizu teorijskih osnova, razvoja matematičkih modela i implementacije softverskih rešenja, postignuti su značajni rezultati koji unapređuju proces donošenja odluka i upravljanja resursima.

Razvijeni alat pokazao je visok stepen fleksibilnosti i prilagodljivosti za primenu u različitim industrijama i vrstama projekata.

Rad je uspešno demonstrirao kako primena linearnog programiranja može unaprediti planiranje i upravljanje projektima. Matematički modeli, softverska rešenja i praktične primene pokazali su vrednost optimizacije u smanjenju troškova i vremena, doprinoseći efikasnijem korišćenju resursa i donošenju boljih odluka. Ova istraživanja pružaju solidnu osnovu za dalji razvoj u oblasti upravljanja projektima.

7. LITERATURA

[1] P.E. Amiolemhen, J. Akpomwomwo „*Building Project Activities/Tasks Time Scheduling using a Linear Programming Model*“, Department of Production Engineering, University of Benin, Nigeria

[2] Jovan J.Petrić „*Operaciona istraživanja*“

[3] Mokhtar S. Bayaraa, John J. Jarvis, Hanif D. Sherali „*Linear Programming and Network Flows*“

[4] Dimitris Bertsimas, John N. Tsitsiklis „*Introduction to Linear Optimization*“

[5] Erić D.“*Uvod u menadžment*”, Ekonomski fakultet, Beograd

Kratka biografija:

Jelena Ubiparip rođena je u Novom Sadu 1995. godine. Osnovne akademske studije završila je 2019. godine na studijskom programu Biomedicinsko inženjerstvo.

Kontakt: jelena.antelj1@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.633
DOI: https://doi.org/10.24867/33BE11Knezevic

ПРОШИРЕЊЕ АЛАТА AUTOPSY СА МОДУЛОМ ЗА ДЕТЕКЦИЈУ ОБЈЕКАТА НА ФОТОГРАФИЈАМА

EXTENDING AUTOPSY TOOL WITH OBJECT DETECTION MODULE

Исидора Кнежевић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИЧКО И РАЧУНАРСКО ИНЖЕЊЕРСТВО

Кратак садржај – *Дигитална форензика се суочава са изазовом анализе великих колекција фотографија које могу садржати кључне доказе у криминалистичкој истрази. У овом раду је представљено решење модул за Autopsy алат, „Детектор оружја и новца“, који користи YOLOv8 модел дубоког учења за детекцију објеката попут пиштоља, ножева, новчаница и кредитних картица, чиме се значајно убрзава процес анализе. Предложено решење показује знатно боље перформансе у детекцији објеката у односу на постојећа решења.*

Кључне речи: *дигитална форензика, детекција објеката, YOLOv8, Autopsy*

Abstract – *Digital forensics faces the challenge of analyzing large collections of photographs that may contain key evidence in a criminal investigation. This paper presents a solution, module for the Autopsy tool, “Weapon and Money Detector”, which uses the YOLOv8 deep learning model to detect objects such as guns, knives, banknotes and credit cards, significantly speeding up the analysis process. The proposed solution shows significantly better performance in object detection compared to existing solutions.*

Keywords: *digital forensics, object detection, YOLOv8, Autopsy*

1. УВОД

Дигитална форензика је научна дисциплина чији предмет су идентификација, прикупљање, чување, прегледање, анализа и презентација дигиталних доказа коришћењем научно и правно ваљаних метода и алата [1]. У савременој дигиталној форензици, фотографије представљају важан извор доказа који могу садржати информације од суштинског значаја за истрагу. Пораст броја рачунара и других рачунарских система које константно користимо у свакодневном животу, довео је до стварања великих количина слика. Међутим, ручна анализа великих колекција фотографија може бити изузетно временски захтевна за дигиталне форензичаре.

У овом раду акценат је стављен на проналажењу фотографија на којима се налазе објекти као што су пиштољи, ножеви, новчанице и кредитне картице. Њихова аутоматска детекција значајно смањује време потребно за анализу слика и пружа истражитељима могућност да се фокусирају на друге аспекте истраге.

Да би се ова аутоматизација постигла, коришћен је напредни модел дубоког учења YOLOv8 за детекцију наведених објеката. Модел је трениран и евалуиран на скупу података који садржи фотографије са објектима од интереса. Развијен је додатак за форензички алат Autopsy верзије 4.21.0, назван Детектор оружја и новца (енг. Weapons and Money Detector), који омогућава примену овог тренираног модела за аутомаско проналажење фотографија на којима се налазе наведени објекти.

2. ПРЕГЛЕД СЛИЧНИХ СОФТВЕРА

Главни критеријуми за избор сличних апликација били су њихова примена у форензичким истрагама, могућност аутоматске анализе мултимедијалног садржаја и подршка за идентификацију одређених објеката од интереса. Посебна пажња је посвећена алатима који укључују методе машинског учења за детекцију објеката. На основу ових критеријума, идентификовани су и анализирани алати као што су Autopsy, верзије 4.21.0, у поглављу 2.1, и Magnet Axiom, верзије 6.8, у поглављу 2.2.

2.1. Autopsy

Autopsy [2] је алат за дигиталну форензику отвореног кода, који омогућава анализу дигиталних доказа са чврстих дискова рачунара и мобилних уређаја. Дизајниран и имплементиран је модуларно, што значи да постоје модули који проширују функције алата. Један од модула јесте и Object Detection модул који служи за детекцију објеката на сликама. Модул користи каскадни класификатор (енг. Cascade Classifier) алгоритам машинског учења из библиотеке за рачунарски вид и машинско учење OpenCV. Пошто библиотека OpenCV, верије 3.4.16, има само класификаторе за детекцију лица, очију, тела, мачака и регистарских таблица, а и Autopsy не садржи своје класификаторе уз модул, било је потребно тренирати и евалуирати класификаторе за објекте које желимо да детектујемо, а то су: пиштољи, ножеви, новчанице и кредитне картице. Метрике које су коришћене за евалуацију су прецизност, одзив и F1 мера. На табели 1 се могу видети вредности евалуације за сваки класификатор.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Стеван Гостојић, ред. проф.

Табела 1. *Резултати евалуације класификатора*

Објекти	Прецизност	Одзив	F1 мера
Пиштољи	0.114	0.472	0.184
Новчанице	0.048	0.245	0.08
Ножеви	0.007	0.087	0.013
Кред. карт.	0.036	0.193	0.06

Резултати евалуације показују да каскадни класификатори имају ниске вредности прецизности и одзива, што указује на велики број лажних позитивних резултата и пропуштање стварних објеката.

2.2. Magnet Axiom

Magnet Axiom [3] је комерцијални софтвер за дигиталну форензику, који је специјализован за анализу података са различитих извора као што су чврсти дискови, мобилни уређаји, cloud сервиси и друштвене мреже. Једна од функционалности је и напредна функција за категоризацију текста у разговорима, слика и видео записа помоћу Magnet AI. Magnet AI је интегрисани систем који користи машинско учење за аутоматску анализу и категоризацију података. Може да идентификује слике које садрже специфичне објекте или сцене, као што су оружје, документа, новац, возила, неподобан садржај и разне друге објекте, У раду [4] Василарас и сарадници су извршили евалуацију функционалности категоризације слика, алата Magnet Axiom, верзије 6.8. Метрике које су коришћене за евалуацију су тачност, прецизност, одзив и F1 мера. Категорије које су евалуиране су: дрога, оружје, голотиња, новац и возила. У табели 2 су приказани резултати евалуације.

Табела 2. *Резултати евалуације алата Magnet Axiom*

Објекти	Тачност	Прецизност	Одзив	F1 мера
Дрога	0.988	0.066	0.279	0.107
Оружје	0.993	0.156	0.052	0.078
Голотиња	0.992	0.231	0.146	0.179
Новац	0.980	0.021	0.333	0.040
Возила	0.991	0.810	0.545	0.652

Резултати евалуације показују да Magnet AI има ограничене перформансе у неким категоријама, попут оружја, дроге и новца, где су ниске вредности прецизности и одзива. Ово указује на повећан број лажних позитивних и пропуштених објеката у тим категоријама. Функционалност категоризације слика помоћу Magnet AI је најпоузданија за категорију возила.

3. AUTOPSY

Овај одељак се бави анализом Autopsy алата за дигиталну форензику. У одељку 3.1 ће бити објашњене основне карактеристике алата и могућностима које нуди корисницима, док ће у одељку 3.2 бити објашњено како се могу проширити функционалности овог алата.

3.1. Основне карактеристике алата

Autopsy се базира на следећим кључним концептима:

- Случај (енг. case): дефинисан је као контејнер који садржи један или више извора података.
- Извор података (енг. data source): могу бити форензичке копије диска, локални дискови и логички фајлови и фолдери.
- Модули за обраду података (енг. ingest modules): обављају анализе фајлова из извора података и парсирају њихов садржај.
- Црна табла (енг. blackboard): представља начин комуникације између модула. Модули постављају своје резултате на црну таблу у облику артефаката, а кориснички интерфејс их приказује.

Неки од модула за обраду података који постоје у Autopsy-у су:

- File Type Identification – идентификује датотеке на основу њихових интерних потписа и не ослања се на екстензије датотека.
- Picture Analyzer – овај модул издваја Exchangeable Image File Format (EXIF) информације из слика, тј. метаподатке. Ове информације могу да садрже податке о геолокацији слике, времену и датуму настанка, моделу камере и њеним подешавањима као и друге информације.
- Object Detection – користи OpenCV да детектује објекте на сликама.

Корисници да би покренули дигиталну форензичку истрагу помоћу Autopsy-а потребно је да направе случај и попуне све неопходне податке о њему. Затим додају извор података и бирају модуле за обраду података који ће се извршавати над тим извором. Након завршетка рада модула, корисници могу ручно да прегледају садржај фајлова и резултате модула како би идентификовали доказе. Када је завршена анализа корисник може да покрене генерисање финалног извештаја на основу одабраних ознака и резултата. Ово укључује креирање HTML или Excel извештаја.

3.2. Проширивост Autopsy-а

Autopsy је препознатљив као један од најприлагодљивијих алата за дигиталну форензику, због могућности проширења функционалности кроз различите модуле. Неке од врста модула које постоје у оквиру Autopsy-а су:

- Модули за обраду података (енг. Ingest Modules): ови модули се покрећу када се нови извор података дода у случај, и могу се поново покренути након тога. Ови модули постоје у два облика: Модули за обраду датотека (енг. File Ingest Modules) – позивају се за сваку датотеку у извору података, и модули за обраду извора података (енг. Data Source Ingest Modules) – позивају се једном за сваку слику диска или сет логичких датотека.
- Модули за извештаје (енг. Report Modules): ови модули се (типично) покрећу након што је корисник прегледао резултате и означио датотеке од интереса. Њихова сврха је креирање извештаја о резултатима, али могу се користити и за анализу.

Autopsy је писан у Java програмском језику, док модули могу бити писани у Java или Python језику (тј.

Jython језику, који претвара код писан у Python-у у Java код). Сваки Python модул, који се пише у једној Python скрипти, треба да буде у фолдеру за Python скрипте у оквиру Autopsy-а. У Autopsy GitHub репозиторијуму [5] се налазе примери за модул за обраду датотека, модул за обраду извора података и модул за извештаје. Сви примери имају TODO напомене које указују на то шта треба да се промени. Све детаљније информације о имплементацији модула, укључујући кораке, примере кода и додатне могућности, могу се пронаћи у званичној Autopsy документацији за девелопере [6].

4. ДЕТЕКЦИЈА ОБЕЈАКТА

Детекција објеката представља један од кључних задатака рачунарског вида и вештачке интелигенције, чији је циљ да препозна и лоцира објекте на дигиталним сликама или видео-снимцима [7]. У одељку 4.1 биће објашњено шта је детекција објеката и како функционише, док ће у одељку 4.2 бити представљени неки од модела за детекцију објеката.

4.1. Основни концепти детекције објеката

Детекција објеката је општи термин који описује скуп повезаних задатака у области рачунарског вида, а који подразумевају идентификацију објеката на дигиталним фотографијама. Општи принцип рада детекције објеката је следећи:

1. Улазна слика: Процес детекције објеката почиње анализом слике или видеа.

2. Предобрада: Слика се предобрађује како би се обезбедио одговарајући формат за модел који се користи.

3. Издвајање карактеристика: Модел разлаже слику на регионе и из сваког региона извлачи карактеристике како би детектовао обрасце различитих објеката.

4. Класификација: Сваки регион слике се класификује у категорије на основу извађених карактеристика. Задатак класификације може да се обавља помоћу SVM-а (support vector machines – тип алгоритма за надгледано учење који се може користити за задатке класификације или регресије) или неуронске мреже која рачуна вероватноћу сваке категорије присутне у региону.

5. Локализација: Истовремено са процесом класификације, модел одређује оквире за сваки детектовани објекат. Ово укључује израчунавање координата оквира који окружује сваки објекат, чиме се објекат прецизно лоцира унутар слике.

6. Сузбијање преклапања (енг. non-max suppression): Када модел идентификује више оквира за исти објекат, примењује се сузбијање преклапања. Ова техника задржава само оквир са највишим нивоом поузданости и уклања све остале који се преклапају.

7. Излаз: Процес се завршава оригиналном сликом на којој су означени оквири и називи класа, које илуструју детектоване објекте и њихове одговарајуће категорије.

4.2. Методе и алгоритми детекције објеката

Детекција објеката ослања се на различите методе и алгоритме који се углавном деле на традиционалне (пре 2014. године) и модерне методе (после 2014. године), засноване на дубоком учењу. Традиционалне методе које се издвајају су: Виола-Џонс детектори (тј. каскадни класификатори), HOG (histogram of oriented gradients) детектор и DPM (deformable part-based model) модел. Модерне методе се деле на две категорије: „двостепена детекција“ и „једностепена детекција“. Код „двостепених детекција“ издвајају се методе: RCNN, SPPNet, Fast RCNN, Faster RCNN, док се код „једностепених детекција“ издвајају методе: you only look once (YOLO) и single shot multibox detector (SSD).

4.2.1. YOLOv8

YOLOv8 је развијен од стране Ultralytics тима. Архитектура YOLOv8 се састоји од два главна дела, кичме и главе, која чине две конволуционе неуронске мреже.

У раду [8] Билоус и сарадници су поредили перформансе модела, као што су YOLOv4-v8, Faster RCNN, SSD и EfficientDet, за детекцију људи и техничких објеката. Главне метрике за поређење биле су тачност (mAP), прецизност, одзив, F1 мера и брзина обраде (FPS). На табели 3 су приказани резултати евалуације модела. YOLOv8 се показао као најефикаснији модел захваљујући својој високој тачности, брзини обраде и способности прилагођавања разноврсним сценаријима. И због тога је YOLOv8 модел изабран за решавање проблема овог рада.

Табела 3. *Резултати евалуације модела*

Модел	Прецизност	Одзив	mAP	F1 мера	FPS
YOLOv4	0.81	0.83	0.82	0.82	40
YOLOv5	0.73	0.73	0.73	0.73	45
YOLOv6	0.62	0.62	0.62	0.62	42
YOLOv7	0.68	0.68	0.68	0.68	43
YOLOv8	**0.87**	**0.89**	**0.88**	**0.88**	**48**
Faster RCNN	0.84	0.82	0.83	0.83	10
SSD	0.76	0.75	0.75	0.75	35
EfficientDet	0.82	0.80	0.81	0.81	30

5. ДЕТЕКТОР ОРУЖЈА И НОВЦА НА ДИГИТАЛНИМ СЛИКАМА

Детектор оружја и новца (енг. Weapons and Money Detector) је модул развијен за форензички алат Autopsy, верзије 4.21.0, који анализира дигиталне слике из извора података и издваја слике на којима се налазе пиштољи, ножеви, новчанице и кредитне картице. Модул је имплементиран у Python скрипти и за детекцију наведених објеката користи YOLOv8 модел, који је трениран и евалуиран на скупу података који садржи фотографије са објектима од интереса. Скуп података над којим је трениран модел преузет је из рада [9] Перез-Ернандеза и сарадника. На табели 4 је приказан скуп података који је коришћен за тренирање и евалуацију модела.

Табела 4. *Скуп података*

	Пиштољи	Новчанице	Ножеви	Кредитне картице
Тренирање	855	425	621	222
Тест	214	102	172	57

Модел је трениран у 25 епоха. Након тренирања је урађена евалуација. Метрике које су коришћене за евалуацију су прецизност, одзив и F1 мера. На табели 5 се могу видети резултати евалуације модела.

Табела 5. *Резултати евалуације YOLOv8 модела*

Објекти	**Прецизност**	**Одзив**	**F1 мера**
Пиштољи	0.913	0.937	0.925
Новчанице	0.911	0.931	0.921
Ножеви	0.930	0.820	0.872
Кред. карт.	0.882	0.474	0.617

Модул Weapons and Money Detector у Autopsy алату садржи фолдер utils, у коме се налазе два фолдера model и dist. У фолдеру model се налази фајл best.pt, који представља YOLOv8 модел, док се у фолдеру dist налази weaponsMoneyDetector.exe датотека.
Корисник када започиње дигиталну форензичку истрагу потребно је прво да направи нови случај и дода извор података. Након тога покреће Weapons and Money Detector модул, који издваја све слике са подржаном екстензијом и копира их у привремени фолдер. Модул затим позива weaponsMoneyDetector.exe датотеку која на сликама детектује објекте од интереса (пиштоље, ножеве, новчанице и кредитне картице) и резултате записује у текстуалну датотеку report.txt. Модул обрађује резултате из текстуалне датотеке и слике са детектованим објектима додаје на Blackboard у секцији Interesting Files груписане у листу под називом Weapons and Money in Images. Након завршетка рада модула, модул обавештава корисника да је процес обраде завршен и корисник може да генерише финални HTML или Excel извештај.

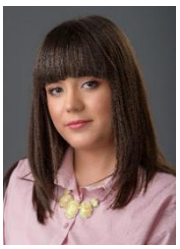
6. ЗАКЉУЧАК

Проблем који је анализиран у овом раду односи се на детекцију потенцијално опасних и значајних објеката, као што су пиштољи, новчанице, ножеви и кредитне картице, у дигиталним сликама. Као метод решавања проблема коришћен је YOLOv8 модел за дубоко учење и развијен је модул за форензички алат Autopsy верзије 4.21.0, Детектор оружја и новца (енг. Weapons and Money Detector), који омогућава примену овог модела за аутоматско проналажење фотографија на којима се налазе наведени објекти. Мотивација за овакав приступ лежи у потреби за ефикаснијом анализом дигиталних доказа у форензичким истрагама, као и у ограничењима постојећих решења, као што су Autopsy и Magnet Axiom, која показују недовољне резултате у прецизности и одзиву њихових модела. Иако YOLOv8 модел показује знатно боље резултате од конкуренције, његова детекција кредитних картица и даље остаје изазов. Правци даљег развоја обухватају побољшање детекције кредитних картица, кроз повећање скупа података за тренирање са већим бројем слика на којима се налазе кредитне картице. Такође, тренирање модела да детектује још више категорија објеката, што ће проширити примену решења у другим областима.

7. ЛИТЕРАТУРА

[1] A. Arnes, Digital Forensics: An Academic Introduction, John Wiley & Sons Ltd, 2018.
[2]"Autopsy User Documentation 4.21.0," [Online]. Available: https://sleuthkit.org/autopsy/docs/user-docs/4.21.0/index.html. [Accessed 7 December 2024].
[3] "Magnet Axiom," Magnet Forensics, [Online]. Available: https://www.magnetforensics.com/products/magnet-axiom/. [Accessed 9 December 2024].
[4] A. Vasilaras, N. Papadoudis and P. Rizomiliotis, "Artificial intelligence in mobile forensics: A survey of current status, a use case analysis and AI alignment objectives," Forensic Science International: Digital Investigation, vol. 49, no. 301737, 2024.
[5] https://github.com/sleuthkit/autopsy/tree/autopsy-4.21.0/pythonExamples [Accessed 9 December 2024].
[6]"Autopsy Forensic Browser Developer's Guide and API Reference," [Online]. Available: http://sleuthkit.org/autopsy/docs/api-docs/4.21.0//. [Accessed 10 December 2024].
[7] J. Murel and E. Kavlakoglu, "What is object detection?" IBM, 3 January 2024. [Online]. Available: https://www.ibm.com/think/topics/object-detection. [Accessed 12 December 2024].
[8] N. Bilous, V. Malko, M. Frohme and A. Nechyporenko, "Comparison of CNN-Based Architectures for Detection of Different Object Classes," AI, vol. 5, pp. 2300-2320, 2024.
[9] F. Pérez-Hernández, S. Tabik, A. Lamas, R. Olmos, H. Fujita and F. Herrera, "Object Detection Binary Classifiers methodology based on deep learning to identify small objects handled similarly: Application in video surveillance," Knowledge-Based Systems, vol. 194, no. 105590, 2020.

Кратка биографија:



Исидора Кнежевић рођена је у Новом Саду 2000. год. Дипломирала је на Факултету техничких наука у Новом Саду 2023. године и исте године уписала мастер студије на Факултету техничких наука, студијски програм Рачунарство и аутоматика, смер Примењене рачунарске науке и информатика – Електронско пословање.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE12Vasic

RAZVOJ SAVREMENIH VEB APLIKACIJA U .NET EKOSISTEMU PRIMENOM *WEBASSEMBLY* I *BLAZOR* TEHNOLOGIJA

DEVELOPMENT OF MODERN WEB APPLICATIONS IN THE .NET ECOSYSTEM THROUGH THE USE OF WEBASSEMBLY AND BLAZOR TECHNOLOGIES

Nataša Vasić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

Kratak sadržaj – *U ovom radu prikazano je istraživanje WebAssembly i Blazor tehnologija, sa ciljem da se predstave njihovi osnovni koncepti, prednosti i praktične primene. WebAssembly bajtkod može se izvršavati u različitim okruženjima, ali u ovom radu fokus je na .NET okviru zbog njegovih širokih mogućnosti i popularnosti među programerima. Dat je pregled mogućnosti koje ove tehnologije pružaju uz doprinos boljem razumevanju njihove primene u savremenoj industriji.*

Ključne reči: *WebAssembly, Blazor, .NET, veb tehnologije, veb aplikacija*

Abstract – *This paper presents a study of WebAssembly and Blazor technologies, aiming to introduce their fundamental concepts, advantages and practical applications. WebAssembly bytecode can be executed in various environments, however this paper focuses on its use within the .NET ecosystem due to its extensive capabilities and popularity among developers. The paper provides an overview of the opportunities these technologies offer and contributes to a better understanding of their application in modern industry.*

Keywords: *WebAssembly, Blazor, .NET, web technologies, web application*

1. UVOD

Razvoj veb tehnologija omogućio je pojavu kompleksnih aplikacija kao što su igrice, različiti audio i video softveri i slične aplikacije, te zahtevi za efikasnošću i sigurnošću rastu. JavaScript, HTML i CSS i dalje dominiraju u razvoju veb aplikacija, ali zbog svojih ograničenja ne mogu u potpunosti da ispune te zahteve. WebAssembly, otvoreni standard koji omogućava izvršavanje binarnog koda direktno u pregledaču, uspeo je da ispuni zahteve [1, 2]. U okviru .NET Blazor tehnologije WebAssembly se koristi kao ključna tehnologija za pokretanje aplikacija u veb pregledačima. Blazor je UI framework koji omogućava razvoj interaktivnih veb aplikacija koristeći C# i .NET umesto korišćenja JavaScript jezika.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Dunja Vrbaški, docent

Blazor omogućava programerima da koriste poznate alate i biblioteke iz .NET okruženja. Ovaj pristup eliminiše potrebu za dva programera, jednog za JavaScript, a drugog za backend u klasičnom veb projektu, povećava produktivnost i omogućava kreiranje aplikacija koje mogu da se pokrenu na svim modernim pregledačima [3].

2. WEBASSEMBLY

WebAssembly (Wasm) je tip koda koji se može izvršavati u modernim veb pregledačima. To je jezik niskog nivoa sličan asembleru koji koristi kompaktan binarni format i omogućava izvršavanje sa performansama bliskim nativnim. Pruža mogućnost kompajliranja programskih jezika kao što su C/C++, C#, Rust i drugi u oblik pogodan za izvršavanje unutar veb pregledača. Dizajniran je da funkcioniše uporedo sa JavaScript jezikom, omogućavajući im da rade zajedno [4]. Razvijen je u saradnji četiri glavne kompanije koje se bave razvojem pregledača – Google, Microsoft, Mozilla i Apple [1]. Prema W3C organizaciji, WebAssembly je četvrti jezik za veb koji omogućava izvršavanje koda u pregledaču [5]. JavaScript, HTML i CSS su ostala tri jezika [2]. WebAssembly je dizajniran da bude kompaktan, brz i siguran, a da pritom omogućava validaciju, kompilaciju i bezbedno izvršavanje uz minimalne troškove. Nezavisan je od programskih jezika, hardvera i platforme. Od samog početka dizajniran je sa formalnom semantikom [1].

2.1. Primene

Google, eBay i Norton implementirali su WebAssembly u svoje projekte kako bi unapredili performanse. Primeri primene uključuju čitače bar kodova [7], mašinsko učenje sa TensorFlow.js bibliotekom [8], prepoznavanje obrazaca, grafičke alate, kompresiju podataka, kriptografske biblioteke, igrice, obradu slika, numeričke proračune i druge specijalizovane zadatke [6]. Jednostavnost i univerzalnost podstakla je njegovu primenu u različitim domenima, uključujući serversku stranu u kombinaciji sa Node.js, serverless računarstvo u oblaku, Internet stvari (IoT) i integrisane uređaje, pa čak i kao samostalno okruženje za izvršavanje [9]. Unutar pregledača koristi se i za uređivanje slika i video sadržaja, podržava različite vrste igrica, razvoj aplikacija, prepoznavanje slika, VPN i drugo. Izvan pregledača koristi se za distribuciju računarskih igara, izvršavanje nepoverljivog koda na serveru, server-side aplikacije i slično [10].

2.2. Razlozi za razvoj novih veb tehnologija

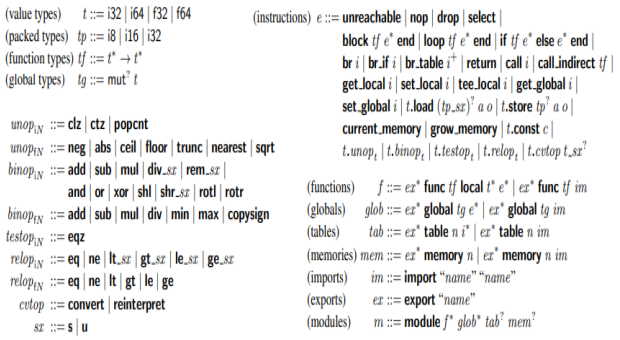
JavaScript je dugo bio jedina opcija za kreiranje interaktivnih aplikacija u pregledaču. Međutim, razvoj aplikacija koje zahtevaju visoke performanse procesora, kao što su igre, bio je ograničen zbog slabih performansi. Da bi se rešili problemi napravljeni su brojni pokušaji da se prednosti nativnog koda prenesu na veb. Adobe je promovisao Flash platformu, Microsoft je predložio ActiveX, Google uvodi Native Client tehnologiju [11].

2.3. Prethodna rešenja

ActiveX kojeg je uveo Microsoft, Flash platforma koju je promovisao Adobe, Native Client kojeg je predložio Google i asm.js, prvi su pokušali da reše izazove sigurnog, brzog i prenosivog koda niskog nivoa. Međutim, to nisu u potpunosti uspeli jer je svaki od njih zahtevao dodatne plugin-ove i bio vezan za specifičan pregledač. Ove tehnologije često su imale problema sa sigurnošću, kompatibilnošću i performansama što je ograničavalo njihovu upotrebu [1, 11].

2.4. Struktura i funkcionalnost binarnih fajlova

Binarni format koda definisan je tako da može da se posmatra kao jezik sa sintaksom i strukturom. Ovaj format olakšava razumevanje, a da pritom ne ugrožava kompaktnu formu ili jednostavnost dekodiranja. Struktura u smislu apstraktne sintakse može se videti na Slici 1 [1].



Slika 1. *WebAssembly pravila sintakse [1]*

Neki od osnovnih koncepata koje WebAssembly koristi su moduli, funkcije, instrukcije, lokalne varijable, globalne varijable i tako dalje [1].

Moduli su binarni fajlovi koji sadrže funkcije, globalne promenljive, tabele i memoriju. Sastoje se od različitih tipova sekcija kojih ima ukupno 11, od kojih su četiri najvažnije: sekcija koda, sekcija podataka i sekcije importa i eksporta. Dok moduli predstavljaju statičku strukturu, instanca modula omogućava dinamičko izvršavanje sa memorijom i stekom. Instanciranje modula obezbeđuje okruženje kao što je JavaScript VM ili operativni sistem. Sekcija koda predstavlja najveću sekciju obuhvatajući sve funkcije modula. Slika 2 prikazuje primer funkcije iz ove sekcije, koja uključuje osnovne operacije poput kontrolnih naredbi, oduzimanja i množenja [1, 11].

C program code	Binary	Text representation
	20 00	get_local 0
	42 00	i64.const 0
	51	i64.eq
int factorial(int n) {	04 7e	if i64
if (n == 0)	42 01	i64.const 1
return 1	05	else
else	20 00	get_local 1
return n * factorial(n-1)	20 00	get_local 1
}	42 01	i64.const 1
	7d	i64.sub
	10 00	call 0
	7e	i64.mul
	0b	end

Slika 1. *Jednostavna C funkcija sa leve strane i odgovarajući WebAssembly bajtkod zajedno sa tekstualnom reprezentacijom poznatom kao Wat format sa desne strane [11]*

Kod u modulu organizovan je u funkcije koje primaju vrednosti kao parametre i vraćaju vrednosti kao rezultate, prema definisanom tipu funkcije. Funkcije mogu međusobno pozivati jedna drugu, uključujući rekurziju, ali ne mogu biti ugnježdene [1].

Izvršavanje operacija u WebAssembly tehnologiji bazira se na stek mašini. Funkcije se sastoje od instrukcija koje manipulišu vrednostima na steku. Sistem sa tipom omogućava statičko određivanje rasporeda steka, što omogućava direktnu kompilaciju tokova podataka bez stvarnog materijalizovanja steka. Ova organizacija steka omogućava kompaktno predstavljanje programa [1].

Neke instrukcije mogu izazvati izuzetke koji odmah prekidaju izvršavanje. WebAssembly kod ne može obraditi izuzetke direktno, ali ih JavaScript okruženje može obraditi. WebAssembly izuzetak će generisati (eng. throw) JavaScript izuzetak koji sadrži trag steka (eng. stacktrace) sa JavaScript i WebAssembly stekom. Trag steka se može uhvatiti i pregledati uz pomoć JavaScript koda [1].

U funkcijama, lokalne promenljive se inicijalizuju na nulu i njima se upravlja pomoću instrukcija get_local i set_local. Instrukcija tee_local omogućava upis u lokalnu promenljivu dok ulazna vrednost ostaje na steku [1].

Moduli mogu deklarisati globalne promenljive koje se čitaju i pišu pomoću instrukcija get_global i set_global. Globalne promenljive mogu biti promenljive ili nepromenljive i moraju imati početnu vrednost koja je konstantan izraz [1].

WebAssembly koristi linearnu memoriju koja predstavlja veliki niz bajtova. Svaki modul može definisati samo jednu memoriju koja se može deliti između različitih instanci putem uvoza/izvoza. Memorija se kreira sa određenom veličinom, ali se može dinamički proširivati pomoću instrukcije za povećanje memorije. Usled nedostatka memorije, proširenje neće uspeti i biće vraćena vrednost -1 kao signal neuspeha. Trenutna veličina memorije može se proveriti korišćenjem instrukcije za ispitivanje memorijskog stanja. WebAssembly memorija definisana je da koristi little-endian redosled bajtova, što znači da platforme sa big-endian redosledom zahtevaju eksplicitne konverzije endian redosleda. Ove konverzije mogu biti optimizovane od strane WebAssembly kompajlera [1].

2.5. Arhitektura

WebAssembly je dizajniran tako da ne definiše način na koji se programi učitavaju u izvršno okruženje, niti kako se obavljaju I/O operacije. Ovakva arhitektura omogućava fleksibilnu integraciju WebAssembly tehnologije u različita izvršna okruženja. Sistem koji implementira WebAssembly preuzima odgovornost za učitavanje modula, povezivanje uvoznih i izvoznih funkcija, obezbeđivanje pristupa I/O operacijama i tajmerima, kao i rukovanje izuzecima [1].

2.6. Performanse

Kompaktni binarni format omogućava brzo učitavanje i dekodiranje. Analiza je dovela do sledećih zaključaka:

• WebAssembly kompajleri uglavnom su zasnovani na LLVM-u, gde optimizacije nisu specifično prilagođene za WebAssembly.

• JIT optimizacije značajno utiču na performanse JavaScript jezika, dok za WebAssembly nema značajne razlike u performansama između verzija sa i bez JIT-a.

• Performanse JavaScript-a i WebAssembly-ja variraju u zavisnosti od pregledača i platforme. Na desktop računarima Firefox pruža bolje rezultate u izvršavanju WebAssembly koda u odnosu na Chrome, dok Edge postiže najlošije rezultate. Nasuprot tome, na mobilnim uređajima, Firefox je sporiji od Chrome-a, dok Edge nadmašuje oba pregledača. Performanse JavaScript jezika takođe variraju u zavisnosti od platforme. Tako je na desktop računarima Firefox sporiji u odnosu na Chrome, dok je na mobilnim uređajima brži.

• WebAssembly zahteva više memorije u poređenju sa JavaScript jezikom. Ovo se može pripisati činjenici da WebAssembly koristi linearni model memorije koji ne oslobađa memoriju automatski, dok JavaScript koristi sakupljanje smeća za automatsko oslobađanje memorije [6].

Rezultati pokazuju da iako se očekuje da WebAssembly bude brži od JavaScript jezika, to nije uvek slučaj i performanse mogu značajno varirati. Kada WebAssembly želi da pristupi ili manipuliše DOM-om, mora da zatraži od JavaScript-a da izvrši te operacije. Ova upotreba uvodi dodatno opterećenje, što može smanjiti performanse WebAssembly koda. WebAssembly se iz tog razloga koristi za zadatke koji zahtevaju intenzivno računanje [12]. JavaScript često postiže bolje rezultate u odnosu na WebAssembly, naročito ako je ulaz u program velik. Prilikom instanciranja WebAssembly modula, veliki deo linearne memorije se inicijalizuje kako bi se simulirale memorijske lokacije. Kada se linearna memorija potpuno popuni, umesto oslobađanja memorije koja više nije u upotrebi, ona se proširuje na veću veličinu. Nasuprot tome, JavaScript koristi automatsko upravljanje memorijom, koje dinamički prati alokaciju memorije i oslobađa onu koja više nije potrebna. Ova dinamika čini JavaScript efikasnijim u korišćenju memorije u poređenju sa WebAssembly tehnologijom [6].

Još jedno od ključnih poboljšanja koje WebAssembly nudi odnosi se na energetsku efikasnost, koja je u proseku poboljšana za 30%. Iako JavaScript u nekim situacijama daje bolje rezultate, opšte je prihvaćeno da je WebAssembly ne samo brži od njega nego i koristi manje energije, što ga čini boljim izborom za razvoj aplikacija [2].

3. BLAZOR

Blazor je .NET frontend veb framework koji omogućava kreiranje interaktivnih korisničkih interfejsa korišćenjem programskog jezika C#, uz mogućnost deljenja aplikativne logike između serverske i klijentske strane. Renderovanje korisničkog interfejsa ostvaruje primenom HTML i CSS koda, čime se omogućava široka podrška za različite pregledače, uključujući i one na mobilnim uređajima. Blazor pruža značajne prednosti kao što su pisanje koda u C#, korišćenje postojećeg .NET ekosistema i integraciju sa savremenim alatima kao što su Docker, Visual Studio i Visual Studio Code, što doprinosi produktivnosti i sigurnosti u procesu razvoja veb aplikacija [13].

Blazor aplikacije zasnivaju se na komponentama, koje predstavljaju osnovne jedinice korisničkog interfejsa, poput stranica, dijaloga ili formi za unos podataka. Komponente su klase implementirane u C# jeziku koje omogućavaju fleksibilno definisanje logike za prikaz korisničkog interfejsa, upravljanje događajima, ugnježdavanje i ponovno korišćenje. Ove komponente mogu se deliti i distribuirati putem Razor biblioteka ili NuGet paketa. Komponente se pišu u formi Razor stranica sa ekstenzijom .razor i koriste Razor sintaksu koja kombinuje HTML i C# kod. Ovaj pristup doprinosi većoj produktivnosti i omogućava efikasnije programiranje unutar integrisanih razvojnih okruženja kao što je Visual Studio. U formalnom kontekstu ove komponente se nazivaju Razor komponente, dok se neformalno često nazivaju Blazor komponente [13].

3.1. Tipovi

Blazor pruža različite tipove implementacije, među kojima su server-side, client-side i hosted. Svaki od ovih tipova dolazi sa specifičnim šablonima koji su dostupni unutar Visual Studio okruženja. Nije moguće odrediti koji tip je najbolji, budući da izbor zavisi od zahteva i potreba projekta [3].

Server-side Blazor omogućava izvršavanje celokupne aplikacione logike na serverskoj strani, koristeći WebSocket tehnologiju za uspostavljanje komunikacije između klijenta i servera. Prednost ovog pristupa leži u mogućnosti pisanja frontend logike u programskom jeziku C#, čime se pojednostavljuje razvoj aplikacije. Međutim, ključni nedostatak ovog tipa je eliminacija potrebe za API pozivima, jer se sve potrebne biblioteke direktno integrišu u frontend, što može dovesti do smanjene efikasnosti ovog rešenja [3].

Client-side Blazor funkcioniše isključivo na klijentskoj strani unutar pregledača. Iako su stranice host-ovane na serveru, sva logika se izvršava na klijentu. Ova opcija je posebno pogodna za prezentacione veb sajtove ili jednostavne veb aplikacije, ali može postati neefikasna u situacijama kada je potrebna interakcija sa bazama podataka ili kada aplikacija već koristi postojeće API servise [3].

Hosted Blazor predstavlja najefikasnije rešenje, gde se logika aplikacije izvršava na klijentskoj strani, čime se optimizuje korišćenje resursa servera. Ovaj pristup

integriše client-side Blazor sa posebnim API projektom, omogućavajući im da funkcionišu zajedno kao jedna celina. Ova kombinacija pruža optimalno rešenje za aplikacije koje zahtevaju intenzivnu interakciju sa serverom [3].

3.2. Blazor WebAssembly Hosted projekat – struktura i sastavni delovi

Blazor WebAssembly šablon automatski kreira inicijalne fajlove i strukuru direktorijuma prilikom generisanja početnog projekta, uključujući demonstracioni kod koji služi kao primer implementacije osnovnih funkcionalnosti. Struktura projekta sastoji se od tri glavna segmenta: klijentski deo (Client), serverskog dela (Server) i deljenih resursa (Shared). Ova struktura omogućava jasno razdvajanje poslovne logike aplikacije od korisničkog interfejsa, pri čemu se zajednički kod održava u okviru posebnog modula kako bi se olakšala njegova ponovna upotreba i konzistentnost između klijentskog i serverskog dela aplikacije.

3.3. Opšti pregled

Blazor pruža širok spektar funkcionalnosti, ali i dalje postoje situacije u kojima je neophodno koristiti JavaScript. Blazor omogućava jednostavnu i efikasnu integraciju sa JavaScript kodom, olakšavajući pristupanje skladištu podataka, rad sa fajlovima i korišćenje postojećih JavaScript biblioteka. Interakcija sa JavaScript kodom u Blazor aplikacijama ostvaruje se putem IJSRuntime interfejsa koji se injektuje u odgovarajuću stranicu. HTML elementi podržavaju širok spektar događaja, od kojih su neki od njih generički, a drugi specifični za određene elemente. Ovi događaji se mogu koristiti direktno u Blazor-u bez potrebe za interakcijama sa JavaScript kodom [3].

4. ZAKLJUČAK

Rezultati ovog rada potvrdili su važnost integracije WebAssembly i Blazor tehnologija u modernom razvoju veb aplikacija. Implementacija WebAssembly tehnologije omogućava visok nivo performansi i fleksibilnosti u aplikacijama, omogućavajući izvršavanje koda na strani klijenta mnogo brže nego tradicionalni pristup zasnovan na JavaScript jeziku. Blazor koristeći C# jezik pojednostavljuje razvoj, pružajući programerima mogućnost da koriste poznat ekosistem i alate i smanjuje potrebu za korišćenjem više programskih jezika. Korišćenje ovih tehnologija dovelo je do ubrzanja razvoja aplikacija, poboljšanja održavanja koda i smanjenja upotrebe JavaScript koda.

5. LITERATURA

[1] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. F. Bastien, „Bringing the Web up to Speed with WebAssembly“, In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 185-200, 2017.

[2] J. De Macedo, R. Abreu, R. Pereira, and J. Saraiva, „WebAssembly versus JavaScript: Energy and Runtime Performance“, In 2022 International Conference on ICT for Sustainability (ICT4S), pp. 24-34, 2022.

[3] T. Litvinavicius, Exploring Blazor: Creating Hosted, Server-side, and Client-side Applications with C#, 1st ed., 2019.

[4] Mozilla Developer Network, „WebAssembly“, https://developer.mozilla.org/en-US/docs/WebAssembly, (pristupljeno u septembru 2024.)

[5] World Wide Web Consortium, „World Wide Web Consortium (W3C) brings a new language to the Web as WebAssembly becomes a W3C Recommendation“, https://www.w3.org/press-releases/2019/wasm/, (pristupljeno u septembru 2024.)

[6] Y. Yan, T. Tu, L. Zhao, Y. Zhou, and W. Wang, „Understanding the Performance of WebAssembly Applications“, In Proceedings of the 21st ACM Internet Measurement Conference, pp. 533-549, 2021.

[7] S. Padmanabhan, and P. Jha, „WebAssembly at eBay: A Real-World Use Case“, https://innovation.ebayinc.com/tech/engineering/webassembly-at-ebay-a-real-world-use-case/, (pristupljeno u oktobru 2024.)

[8] D. Smilkov, N. Thorat, and A. Yuan, „Introducing the WebAssembly backend for TensorFlow.js“, https://blog.tensorflow.org/2020/03/introducing-webassembly-backend-for-tensorflow-js.html, (pristupljeno u oktobru 2024.)

[9] D.Lehmann, J. Kinder, and M. Pradel, „Everything Old is New Again: Binary Security of WebAssembly“, In 29th USENIX Security Symposium (USENIX Security 20), pp. 217-234, 2020.

[10] WebAssembly, „WebAssembly“, https://webassembly.org/, (pristupljeno u septembru 2024.)

[11] M. Musch, C. Wressnegger, M. Johns, and K. Rieck, „New Kid on the Web: A Study on the Prevalence of WebAssembly in the Wild“, In Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, Springer International Publishing, pp. 23-42, 2019.

[12] D. Kievits, „What effect does applying WebAssembly have on a compute intensive client-side application versus JavaScript?“, 2021.

[13] Microsoft, „ASP.NET Core Blazor“, https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0, (pristupljeno u avgutu 2024.)

Kratka biografija:

Nataša Vasić rođena je u Novom Sadu 1999. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva odbranila je 2025.god.
kontakt: natasavas00@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.92
DOI: https://doi.org/10.24867/33BE13Ruzicic

RAZVOJ INTERPETERA U PROGRAMSKOM JEZIKU GO

DEVELOPMENT OF AN INTERPRETER IN THE GO PROGRAMMING LANGUAGE

Ratko Ružičić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *Ovaj rad bavi se izradom interpretera hipotetičkog programskog jezika, on predstavlja nastavak prethodnog rada autora na temu kompajlera izrađenog u programskom jeziku C, koji je koristio alate "flex" i "bison" za faze leksičke analize i parsiranja. Za razliku od tog rada, interpretator predstavljen u ovoj tezi razvijen je isključivo korišćenjem standardne biblioteke programskog jezika Go, bez oslanjanja na dodatne alate ili postojeći kod.*

Ključne reči: *Programski jezici, Interpreteri, Dinamički tipovi*

Abstract – *This paper explores the implementation of a hypothetical programming language, it is an extension of a previous paper by the same author of a compiler implemented in C programming language using tools "flex" and "bison" for lexical analysis and parsing. In contrast to that paper, an interpreter presented here was developed exclusively using a standard library of Go programming language without relying on any other tools or existing code.*

Keywords: *Programming languages, Interpreters, Dynamic types*

1. UVOD

1.1. Definicija problema

Za početak neophodno je definisati programski jezik koji bi imao sledeće funkcionalne osobine:

- dinamički sistem tipova
- podrška za kondicionale(if...else)
- podrška za funkcije
- osnovni aritmetički i logički operatori
- nizovi
- petlje
- osnovni tipovi podataka: integer, double, boolean, string
- interaktivni(REPL) mod i učitavanje programa iz fajla

Pored funkcionalnih zahteva interpreter bi trebao da zadovolji i određene nefunkcionalne zahteve. Najbitniji među njima jeste lako distribuiranje interpretera i omogućavanje krajnjim korisnicima da preuzmu interpreter i da ga pokrenu bez potrebe za mukotrpnim podešavanjem radnog okruženja i instaliranja dodatnih programa. Kao dodatni zahtev može se izdvojiti i pokretanje interpretera uz pomoć samo jedne komande.

1.1. Opseg i ograničenja

Polje kompajlera i interpretera je jako složeno i vrlo dobro proučavano, takođe postoji vrlo dobra povezanost sa industrijom pa je mnogo novca i vremena uloženo u proučavanje ove oblasti od strane vrlo uspešnih kompanija. Uzevši to u obzir može se zaključiti da su današnji interpeteri često rezultat višedecenijskog napora što od strane samih kompanija, što od strane nezavisnih kontributora što samo potvrđuje činjenicu da je dizajniranje novog programskog jezika i implementiranje interpretera za njega ogroman poduhvat i da je put od rane implementacije do prihvatanja od strane industrije i entuzijasta vrlo dug i jako nepredvidiv [1].

2. TEORIJA

2.1. Leksička analiza

Leksička analiza je prvi korak prilikom interpretacije (ili kompilacije) programskog jezika. Ulaz u leksičku analizu predstavlja "sirovi" tekst sačinjen od niza karaktera a izlaz predstavlja niz tokena. Tako na primer ako postoji ulaz sadržine "var a = 3;". Kao izlaz se može očekivati niz tokena: "VAR, IDENTIFIER, ASSIGN, NUMBER, SEMICOLON". Pored toga očekuje se da leksička analiza javi grešku u slučaju da za dati ulaz nije moguće generisati listu tokena ili da pak dostavi token koji bi označio da za dati unos postoji greška. Na primer, za ulaz "var @ = 3;", očekivani izlaz bi bio: "VAR, ERROR". Dakle lekser (nekada u literaturi nazvan i skener) je javio da je prvi token VAR a da njega sledi ERROR token pošto @ karakter nije podržan u gramatici jezika. U ovom primeru može se uočiti da je lekser efektivno prestao sa radom u momentu kada je naišao na karakter koji je uzrokovao grešku, u praksi ovo često nije slučaj. Skoro sve implementacije modernih jezika omogućavaju dalji rad leksera kako bi pronašao što više grešaka i o tome obavestio krajnjeg korisnika koji bi onda dobio listing grešaka koje treba da reši.

Današnji lekseri takođe uz sam token generišu i značajnu količinu metapodataka o samom tokenu, osnovni metapodaci mogu biti broj linije i broj karaktera unutar fajla nad kojim je vršeno leksiranje, dok neki napredniji metapodaci mogu biti dužina same lekseme, sadržaj lekseme i ime fajla u kom je leksema pronađena.

NAPOMENA:

Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dunja Vrbaški, docent

2.2. AST

Kako bi opisali značenje i značaj AST-a moguće je osvrnuti se na sam akronim(u obrnutom redosledu): stabla (*eng. Tree*) označavaju da se radi o običnoj strukturi podataka stabla koje su jako česte u kompjuterskoj nauci, sintaksna (*eng. Syntax*) označava da se radi o sintaksi samog jezika tj. da se sintaksa samog jezika može opisati jednim takvim stablom i apstraktna(*eng. Abstract*) u ovom slučaju znači da ova stabla predstavljaju strukturu a ne da mapiraju svaki karakter na svaki čvor u datom stablu.

2.3. Parsiranje

Sledeća faza u interpretaciji programa jeste parsiranje, zadatak parsera programskog jezika jeste da definiše sintaksu jezika tj. da preuzme tokene dobijene iz prethodnog koraka (leksičke analize) i da utvrdi da li je redosled tih tokena ispravan i da dodatno vrati neku strukturu podataka koja bi bila reprezent samog programa.

Parseri se uglavnom mogu ručno implementirati i većina implementacija programskih jezika ima ručno napisane parsere ali takođe postoje alati za generisanje koda parsera poput “ANTLR”, “bison” ili “yacc” alata koji za definisanu formalnu gramatiku generišu skelet koda parsera a korisnik može definisati svoju logiku provere semantike jezika, pa čak i generisanje koda.

Parseri se ugrubo mogu podeliti u 2 različite kategorije, “top-down” i “bottom-up”. Glavna razlika između ova dva tipa jeste da “top-down” parseri kreću od korena abstraktnog stabla i pokušavaju da dođu do listova samog stabla pri tom proveravajući da li su zadovoljena pravila gramatike jezika, dok “bottom-up” parseri kreću od samih tokena (tj. listova stabla) i pokušavaju da "izgrade" stablo.

Generalno govoreći “top-down” parseri su lakši za implementaciju ako su ručno implementirani, dok su alati za generisanje parsera najčešće napravljeni da generišu neki derivat “bottom-up” parsera.

Klasičan primer “top-down” parsera je Pratov parser (takođe poznat kao "recursive descent parser") koji je osmislio Von Prat 1970-ih godina. Pratov parser se u suštini bavi parsiranjem izraza i rešavanjem problema redosleda aritmetičkih operacija nad izrazima, on nudi elegantan način da se predstave odnosi između različitih operacija. Sve što je potrebno jeste da se definišu operatori i njihov prioritet i Pratov parser će znati kako tačno da grupiše operatore. Ako se na primer uzme sledeći izraz za razmatranje “1 + 2 * 3 - 10 / 5”, onda izlaz Pratovog parsera može biti sama vrednost izraza, u ovom slučaju “5”, ili isti izraz ali u formatu grupisanom uz pomoć zagrada tj. “(1 + ((2 * 3) - (10 / 5)))” ili na kraju krajeva kao apstraktno sintaksno stablo koje će prikazati koje podizraze treba evaluirati prvo.

2.4. Evaluacija

Evaluacija se odnosi na korak u interpretaciji u kom se u apstraktnom sintaksnom stablu posećuju svi čvorovi i na osnovu tipa čvora (i njegovih atributa) vrši evaluacija izraza. Radi o jednostavnoj implementaciji "šetanja" kroz stablo i implementiranje logike evaluacije.
Sve do koraka evaluacije razlike između interpretera i kompajlera nije bilo ali u procesu evaluacije je došlo do bitnog razdvajanja. U ovom koraku su zaista bile evaluirane vrednosti izraza ali ako bi cilj bio da se implementira kompajler onda to ne bi bilo moguće učiniti. Kod implementacije kompajlera isti bi generisao nekakav kod, bilo da je u pitanju mašinski kod, asemblerski kod ili nekakav bajtkod za neku hipotetičku (ili pravu) virtuelnu mašinu. Razlika deluje suptilno, i u primeru jednog "hobi" programskog jezika ona suštinski to i jeste ali ako se sagleda šira slika primetno je da interpreter zapravo izvršava sve što mu je dato što znači da ako bi interpreter želeo da pristupi fajlovima ili da otvori nekakvu datoteku on bi morao da to uradi preko jezika u kome je implementiran, dok bi kompajler za taj korak samo generisao mašinski kod koji bi uputio par sistemskih poziva i otvorio fajl a taj mašinski kod bi kasnije bio i izvršen. Svaki moderan programski jezik današnjice podržava pristup fajlovima sa fajl sistema tako da to nije realan problem ali i dalje treba biti svestan činjenice da izbor jezika u kome se implementira interpreter može mnogo da doprinese performansama i funkcionalnostima rezultujućeg interpretera.

3. DIZAJN

Prilikom faze dizajniranja jezika pristupilo se istraživanju koje su popularne osobine jezika današnjice, u tome je najviše pomogao “StackOverflow Developer Survey” tj. anketa koju na kraju svake godine popunjavaju programeri. Između svih programskih jezika najviše su se izdvajali “Python” i “JavaScript” kako među profesionalcima tako i među početnicima pa je prilikom dizajniranja sintakse i odabira osobina jezika bilo logično pokušati imitirati određene osobine tih jezika [2].

3.1. Interpretacija

Prva i glavna osobina ovog programskog jezika jeste to da je isti interpretiran a ne kompajliran. Postoji više razloga zašto je interpretirani jezik ponekad dobar izbor: veća fleksibilnost, prenosivost, manja kriva učenja za početnike itd. No, za krajnjeg korisnika najbitniji razlog jeste prenosivost samog jezika, prilikom konstruisanja kompajlera bitna je odluka koju će arhitekturu pogađati kompajler: ARM, x86 ili PowerPC itd., mada je u današnje vreme donošenje takve odluke olakšano postojanjem radnih okvira za kompajlere poput LLVM. Bilo kako bilo, definitivno je lakše napisati interpreter u nekom dobro podržanom jeziku koji se može izvršavati na skoro svakoj arhitekturi (a takvih jezika danas ne manjka pogotovu zbog prethodno pomenutog LLVM) i time obezbediti da se i ovaj interpreter može pokrenuti na istim tim arhitekturama.

3.2. Dinamički tipovi

Inicijalna zamisao bila je da interpreter podržava statičke tipove, i prva implementacija interpretera zaista jeste imala statičke tipove, ali tokom korišćenja interpretera došlo se do zaključka da je ipak bolje slediti primere čisto interpretiranih jezika koji podržavaju dinamičke tipove. Većina interpretera zaista jeste dinamički tipizirana i iako možda jeste moguće naći primere statički tipiziranog jezika koji je interpreteran, npr. “TypeScript” (doduše pitanje je koliko je ovo dobar primer pošto se “TypeScript” zapravo transpajlira u “JavaScript” koji je interpretiran i dinamički tipiziran), glavni razlog jeste lakoća upotrebe jezika i to je definitivno tačno kada se radi o malim programima i

kratkim skriptama, a budući da će većina programa pisanim u jeziku koji je implementirao interpreter biti mali programi ili kratke skripte onda je logično da se sa korisnika skine "teret" tipiziranja promenljivih.

3.3. Automatsko upravljanje memorijom

Tokom uobičajenog rada programa isti zauzima i oslobađa određenu količinu radne memorije. U računarstvu, memorija je ograničen a često i veoma vredan resurs o kome se mora pažljivo voditi računa, pa je jako bitno minimizovati količinu upotrebljene memorije. Postoje dva pristupa u rukovođenju radnom memorijom. Prvi pristup jeste da se rukovođenje memorijom prepusti programeru koji implementira program. On će određivati trenutke kada i koliko memorije će biti zauzeto pozivanjem funkcija iz programskog jezika koje će tokom izvršavanja vršiti sistemske pozive ka operativnom sistemu na kom se program izvršava. U onom trenutku kada određeni komad memorije bude nepotreban, ta memorija se potom može osloboditi. Postoje dva ključna aspekta na koje programer u ovom slučaju uvek mora da misli: prvi, da ne zauzme više memorije nego što mu je neophodno i drugi, da ne zaboravi da oslobodi memoriju koju je zauzeo. Drugi pristup u rukovođenju memorije jeste da se programeru oduzme sloboda zauzimanja i oslobađanja memorije i da se rukovođenje prepusti kompajleru ili interpreteru. Ovaj deo posla vrši zasebna komponenta interpretera koja se zove modul za automatsko upravljanje memorijom, a češće se koristi pojam na engleskom tj. "garbage collector".

Kada bi se jezik interpretera implementirao u jeziku koji ne podržava automatsko upravljanje memorijom (npr. C) onda bi bilo nepohodno da se isti i implementira (zapravo postoji i druga opcija a to je da prepustimo zauzimanje/oslobađanje memorije krajnjem korisniku što nije uobičajeno ili treća opcija da nikad ne oslobađamo memoriju što je suludo). Budući da je odabran jezik koji ima ugrađeno automatsko upravljanje memorijom ovaj deo posla je već odrađen pošto svako zauzimanje memorije za promenljive u jeziku koji implementira automatski znači da će se ista operacija prevesti u zauzimanje memorije od strane jezika koji implementira interpreter.

4. ZAKLJUČAK

Interpreteri možda nisu najefikasnije rešenje za izvršavanje koda ali oni nude određene karakteristike koje ih čine vrlo privlačnim u današnjim razvojnim okruženjima poput brzine razvoja aplikacija i lakšeg otkrivanja grešaka prilikom razvoja ali to dolazi po cenu nižih performansi u odnosu na kompajlere.

Današnji intereteri postigli su mnogo u pogledu performansi, pa je ta razlika je dosta manja u odnosu na period pre par decenija što ih čini odličnim kandidatima za započinjanje novih projekata [3].

Svrha ovog rada bila je da se proširi znanje stečeno tokom master i osnovnih studija kao i upoznavanje sa novim konceptima prilikom dizajniranja i implementacije jezika, radi se o kompleksnoj temi koja je predmet istraživanja i industrije i akademije i spada u red onih tehnologija koji predstavljaju temelj modernih informacionih tehnologija.

5. LITERATURA

[1] L. A. Meyerovich and A. S. Rabkin, ”Empirical analysis of programming language adoption”, Association for Computing Machinery, vol. 48, pp. 1-18, Oktobar 2013.

[2] StackOverflow, https://survey.stackoverflow.co/2024/, (pristupljeno u junu 2025.)

[3] T. H. Romer, D. Lee, G. M. Voelker, A. Wolman, W. A. Wong, J. Baer, B. N. Bershad, H. M. Levy, ”The structure and performance of interpreters”, Association for Computing Machinery, vol. 31, pp. 150-159, Septembar 1996.

Kratka biografija:

Ratko Ružičić rođen je 2000. godine u Čačku. Završio Tehničku školu u istom gradu 2019. godine. Diplomirao je na Fakultetu Tehničkih Nauka Univerziteta u Novom Sadu 2023. godine.
kontakt: rruzicic@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.3
DOI: https://doi.org/10.24867/33BE14Maksic

РЈЕШАВАЊЕ ПРОБЛЕМА АУТОМАТИЗАЦИЈЕ ПРЕГЛЕДАЊА VHDL ЗАДАТАКА КРОЗ ИНТЕГРАЦИЈУ SYSTEM VERILOG И PYTHON АЛАТА

SOLVING THE PROBLEM OF AUTOMATING VHDL ASSIGNMENT REVIEW THROUGH THE INTEGRATION OF SYSTEM VERILOG AND PYTHON TOOLS

Миленко Максић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Рад се бави развојем система за аутоматизовано прегледање студентских задатака написаних у VHDL-у, кроз интеграцију SystemVerilog алата за верификацију и Python скрипти за анализу резултата. Циљ система је убрзавање и уједначавање процеса оцјењивања, посебно у условима великог броја студената. SystemVerilog се користи за дефинисање тврдњи и тест сценарија, док Python омогућава аутоматско покретање симулација, парсирање лог фајлова и генерисање извјештаја са бодовима. Систем је примјењен на архиву од 223 студентска рада, а добијени резултати су упоређени са ручним прегледањем. Анализа показује да аутоматизовано оцјењивање може значајно смањити вријеме прегледања и повећати објективност, али и да постоје изазови у праведном бодовању дјелимично тачних рјешења, што је посебно важно у образовном контексту.*

Кључне ријечи: *аутоматизација прегледања, VHDL, SystemVerilog, Python, тврдње, симулација, студентски задаци, дигитални дизајн, верификација хардвера, образовање*

Abstract – *This paper presents a system for automated evaluation of student assignments written in VHDL, integrating SystemVerilog verification tools with Python scripts for result analysis. The goal is to accelerate and standardize the grading process, especially in large academic groups. SystemVerilog is used to define assertion and test scenarios, while Python automates simulation execution, log parsing, and score report generation. The system was applied to a dataset of 223 student submissions, and the results were compared with manual grading. The analysis shows that automated evaluation can significantly reduce grading time and improve objectivity, but also highlights challenges in fairly assessing partially correct solutions – an important consideration in educational environments.*

Keywords: *automated evaluation, VHDL, SystemVerilog, Python, assertions, simulation, student assignments, digital design, hardware verification, education*

НАПОМЕНА:
Овај рад је проистекао из мастер рада чији ментор је био др Небојша Пјевалица, редовни професор.

1. УВОД

Брзи развој интегрисаних кола и дигиталних система значајно је утицао на хардверско инжењерство. Савремени дизајн дигиталних система ослања се на језике за опис хардвера (HDL), као што су VHDL и Verilog, који омогућавају прецизно пројектовање, симулацију и верификацију сложених компоненти. Сложеност система захтијева темељну верификацију, која често траје дуже од самог дизајна. Због тога се користе напредни алати попут SystemVerilog-a, који комбинује објектно оријентисано програмирање и алате за тестирање. Ови алати побољшавају квалитет производа, убрзавају развој и смањују ризик од грешака, што је кључно за индустријску примјену и тржишну конкурентност.

2. ТЕОРИЈСКЕ ОСНОВЕ

Језици за хардверски опис (HDL), као што су VHDL и SystemVerilog, представљају темељ савременог дизајна дигиталних система. Они омогућавају описивање структуре, понашања и временских аспеката хардвера, уз подршку за симулацију и синтезу. VHDL се истиче својом структуром заснованом на ентитетима и архитектурама, што омогућава јасно раздвајање интерфејса и функционалности. Његова примјена обухвата симулацију, прототиповање и документацију, чиме се убрзава развој и смањује ризик од грешака. SystemVerilog, као надоградња Verilog-a, уводи напредне могућности за верификацију, укључујући тврдње (assertions), објектно оријентисано програмирање и подршку за сложене тестне сценарије. Његова интеграција дизајна и тестирања омогућава рано откривање грешака и већу поузданост система. Верификација је кључна фаза у развоју дигиталних система. Поред симулације, све више се користе аутоматизоване методологије и алати као што је SVUnit, који омогућава модуларно тестирање у SystemVerilog-у. SVUnit подржава аутоматизацију, генерисање извјештаја и интеграцију са симулационим окружењима, чиме доприноси ефикаснијем и поузданијем развоју.

3. МОТИВАЦИЈА ПРОБЛЕМА СА КОНЦЕПТОМ РЈЕШЕЊА

Обука студената у домену техничких дисциплина подразумјева објективну провјеру знања кроз задатке у којима се сагледава оспособљеност у самосталном раду на доменским проблемима у ограниченом

времену. Курс „Логичко пројектовање рачунарских система 1“, уводи студенте у основе пројектовања дигиталних система коришћењем VHDL језика, слушају га студенти друге године основних академских студија на три студијска програма. Битна карактеристика курса је релативно велики број студената, што последично доноси веома масовне провјере знања. Ручно оцјењивање великог броја студентских радова представља значајан изазов због сложености задатака и ризика од људских грешака. Као рјешење, предложен је систем за аутоматизовано прегледање, који комбинује Python скрипте за обраду података и SystemVerilog алате за симулацију и верификацију.

Асистенти креирају прилагођене тестне сценарије и тврдње (SVA) како би се осигурала тачна и објективна процјена. Овај приступ омогућава брже, поузданије и конзистентније оцјењивање, уз бољу повратну информацију за студенте, што позитивно утиче на њихово учење и развој.

4. ПРОГРАМСКО РЈЕШЕЊЕ

Основу програмског рјешења чини раније имплементирана инфраструктура за чување студентских задатака, покретање симулација и техничку валидацију, уз коришћење алата Siemens Questa и Quartus. У овом раду додати су модули за обраду добијених извјештаја, систематско бодовање и аутоматизовано формирање резултата, што унапређује објективност и ефикасност оцјењивања.

4.1. Студентска архива

Архива садржи студентске директоријуме са рјешењима постављеног задатка и пратећим лог фајловима. Ови логови настају као резултат аутоматске валидације и детекције грешака као што су комбинационе петље, лечеви и непотпуне листе осјетљивости.

4.2. Аутоматска провјера и валидација студентских рјешења

Скрипта sva.py покреће симулацију у Siemens Questa алату, тестирајући DUT и тестбенч. Резултати се биљеже у лог фајлове run_dut.log и run_tb.log, што омогућава детаљну и објективну процјену сваке компоненте.

4.3. Идентификација комбинационих петљи и лечева

Скрипта qaas.py анализира дизајн у Quartus окружењу, генеришући извјештаје о грешкама као што су лечеви, петље и непотпуне листе. Ови подаци се чувају у top_map_rpt.log, што омогућава бодовање по унапријед дефинисаним критеријумима.

4.4. Креирање извјештаја са коначним резултатима

Процес бодовања се одвија у два корака: парсирање лог фајлова и формирање табеларног приказа бодова по студенту.

4.4.1. Парсирање лог фајлова

Скрипта провјерава постојање логова и анализира тврдње (assertions), број исправних DFF-ова и присуство грешака. Резултати се чувају у JSON фајловима:

- assertions_summary.json
- synth_vho_rst_summary.json
- synth_3problem_summary.json

4.4.2. Сумарно формирање бодова

Бодови се сабирају на основу тачних тврдњи и исправних DFF-ова, а затим се умањују за број детектованих грешака. Коначни резултат представља укупан број бодова по студенту.

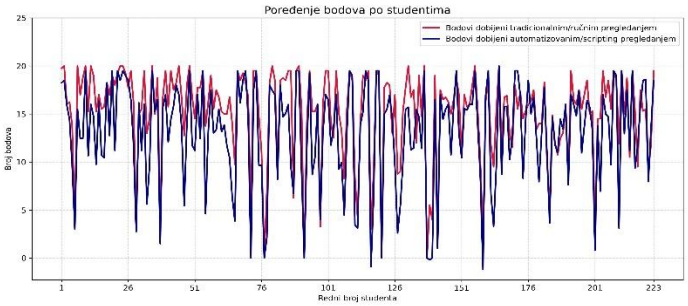
5. РЕЗУЛТАТИ

Програмско рјешење примјењено је на базу од 223 студентска рада, при чему су резултати аутоматске анализе упоређени са оцјенама добијеним путем ручног прегледа. Циљ поређења био је утврђивање тачности, досљедности и мјере разлике између ова два приступа. Машинско бодовање изведено је према два критеријума, базирана на присуству и исправности тврдњи у рјешењима студената.

5.1. Резултати добијени критеријумом 1

Критеријум 1 додјељује бод ако се појединачна тврдња у рјешењу бар једном појављује тачно, без обзира на евентуалне нетачне појаве у остатку кода. На примјер, ако студент у свом дизајну има тврдњу попут ASSERT_sCNT1, која провјерава инкрементацију бројача sCNT на позитивну ивицу сигнала iCLK, бод се додјељује ако је та тврдња најмање једном била тачно симулирана.

- Машинско бодовање је стабилније и досљедније; ручно оцјењивање показује варијације због субјективних оцјена.
- Просјечно оштећење: 2.11 бодова по студенту
- Највеће оштећење: −10.05 бодова
- Највећа добит: +4.00 бода
  Потпуно поклапање резултата код 13 студената

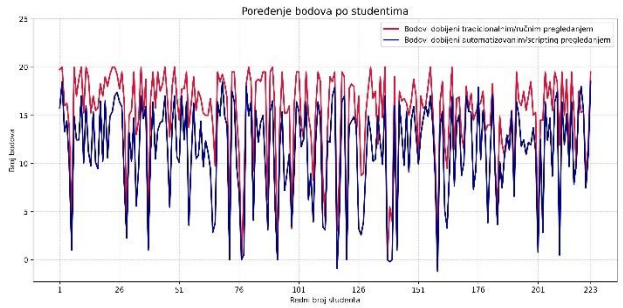


Слика 1 Анализа бодовања – критеријум 1

5.2. Резултати добијени критеријумом 2

Критеријум 2 додјељује бод само ако се тврдња ни једном не појављује нетачно – чак и ако је једном била тачна, бод се укида ако је током других симулација тврдња погрешно активирана. На примјер, за ASSERT_sSIFRA2, која провјерава правилно додавање цифре у шифру приликом уноса, студент губи бод ако се у било ком тренутку током симулације та тврдња испостави као нетачна.

- Строжија оцјена доводи до већег губитка бодова, али с већом прецизношћу у техничкој процјени исправности.
- Просјечно оштећење: 3.70 бодова по студенту
- Највеће оштећење: −13.75 бодова
- Највећа добит: +2.95 бодова
  Потпуно поклапање резултата код 2 студента



Слика Анализа бодовања – критеријум 2

6. ЗАКЉУЧАК

У хардверској индустрији, верификација система се ослања на методе као што су coverage, functional coverage и тврдње (assertions), које омогућавају бинарну процјену исправности – систем је или исправан или није. Овај приступ је ефикасан у техничком контексту, али када се примјењује у образовању, јављају се специфични изазови. Студентска рјешења често нису потпуно исправна, али ни потпуно погрешна, што захтијева флексибилнији и нијансиранији приступ бодовању.

Coverage у индустрији значи да је систем бар једном радио исправно. Functional coverage провјерава да ли су све функционалности тестиране, док тврдње служе за формалну потврду да је систем реаговао у складу са очекивањима у одређеним условима. У академском контексту, овакви критеријуми могу бити недовољни јер не одражавају степен разумјевања студента, нити сложеност његовог рјешења.

Кључна дилема у образовању је: да ли студент заслужује пуне бодове ако је тврдња једном тачна, али у другим случајевима није? У индустрији, то би се сматрало успјехом, али у настави је потребно увести парцијално бодовање које узима у обзир и дјелимичну исправност. На примјер, ако рјешење функционише у већини тест случајева, али садржи мању грешку у једном сценарију, треба размотрити да ли заслужује дио бодова.

Да би се овај проблем ријешио, потребно је развити систем класификације својстава студентских рјешења и дефинисати критеријуме успјеха. То укључује:

- идентификацију критичних својстава која морају бити задовољена,
- одређивање степена исправности (нпр. 90% тачности),
- и формулисање флексибилног модела оцјењивања који уважава различите нивое тачности.

У даљем истраживању, потребно је развити објективан модел бодовања који ће балансирати између индустријских стандарда и академских потреба. Такав модел треба да буде праведан, транспарентан и педагошки оправдан, омогућавајући студентима да добију јасну повратну информацију и подстицај за даље усавршавање.

7. ЛИТЕРАТУРА

[1] „C. Spear, SystemVerilog for Verification: A Guide to Learning the Testbench Language Features,“ 3rd ed. Springer, 2012.

[2] „Н. Пјевалица: Верификација дигиталних интегрисаних кола, System Verilog са основама UVM-а“, Факултет Техничких наука у Новом Саду, 2022, ISBN 9788660224073

[3] „B. Cohen, SystemVerilog Assertions Handbook,“ 3rd ed. VhdlCohen Publishing, 2010.

[4] „D. L. Perry, VHDL: Programming by Example,“ 4th ed. McGraw-Hill, 2002.

[5] „IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language, IEEE Std 1800™-2017,“ IEEE Computer Society, 2017.

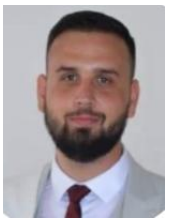
[6] „Intel Corporation, Intel Quartus Prime Pro Edition User Guide: Design Compilation, 2023.“ [На мрежи]. Available: https://www.intel.com

[7] „Siemens Digital Industries Software, QuestaSim User’s Manual, 2023.“ [На мрежи]. Available: https://eda.sw.siemens.com

[8] „AgileSoC Inc., SVUnit User Guide, 2023.“ [На мрежи]. Available: https://github.com/svunit/svunit

[9] „Python Software Foundation, Python 3 Documentation, 2023.“ [На мрежи]. Available: https://docs.python.org/3/

Кратка биографија



Миленко Максић је рођен 13. септембра 1997. године у Бијељини. У школској 2016/17 уписује студијски програм Рачунарство и аутоматика на Факултету техничких наука у Новом Саду. Након завршених основних студија, 2022. године, уписује мастер студије на истом студијском програму, смјер Софтвер за потрошачку електронику.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 7.52
DOI: https://doi.org/10.24867/33BE15Pavkovic

PRIMENA VEŠTAČKIH NEURONSKIH MREŽA U ESTIMACIJI UNUTRAŠNJE TELESNE TEMPERATURE

APPLICATION OF ARTIFICIAL NEURAL NETWORKS IN ESTIMATING CORE BODY TEMPERATURE

Vukašin Pavković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Rad se bavi procenom unutrašnje telesne temperature čoveka na osnovu neinvazivnih fizioloških signala. Primenom različitih arhitektura veštačkih neuronskih mreža razvijeni su modeli za estimaciju unutrašnje temperature. Nakon obrade i normalizacije podataka, modeli su obučeni i testirani, a njihova tačnost procenjena primenom standardnih metrika*

Ključne reči: *Veštačke neuronske mreže, unutrašnja telesna temperatura, obrada podataka*

Abstract – *This thesis addresses the estimation of core body temperature using non-invasive physiological signals. Machine-learning models based on different artificial neural-network architectures were developed to estimate temperature. After data preprocessing and normalization, the models were trained and tested, and their accuracy assessed with standard regression metrics.*

Keywords: *Artificial neural networks, core body temperature, data processing*

1. UVOD

Precizno praćenje unutrašnje telesne temperature je od suštinskog značaja u mnogim situacijama od intenzivne nege pacijenata do praćenja stanja sportista i radnika u ekstremnim uslovima [1]. Unutrašnja telesna temperatura pouzdan je indikator zdravstvenog stanja; čak i relativno mala odstupanja mogu ukazivati na ozbiljne probleme (npr. hipertermiju ili hipotermiju). Ipak, direktno merenje temperature jezgra tela obično zahteva invazivne metode (npr. rektalne ili ezofagealne sonde) koje nisu pogodne za kontinuirani nadzor ili terensku primenu [2]. Razvoj neinvazivnih metoda za procenu unutrašnje temperature postao je važan istraživački izazov. Jedan od pristupa je korišćenje veštačkih neuronskih mreža (VNM) koji na osnovu lako merljivih fizioloških signala mogu predvideti unutrašnju temperaturu. Motivi za primenu VNM leže u njihovoj sposobnosti da modeluju složene nelinearne odnose između ulaznih parametara (npr. temperatura kože, puls, spoljašnja temperatura okoline) i telesne temperature. Na taj način, moguće je dizajnirati sistem za kontinuirano praćenje koji je istovremeno neinvazivan i dovoljno precizan.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Nikola Jorgovanović, red. prof.

U ovom radu istražene su dve strukture neuronskih mreža: klasična feedforward mreža i njena proširena verzija rekurentna feedforward mreža sa ciljem estimacije unutrašnje telesne temperature.

2. METODOLOGIJA

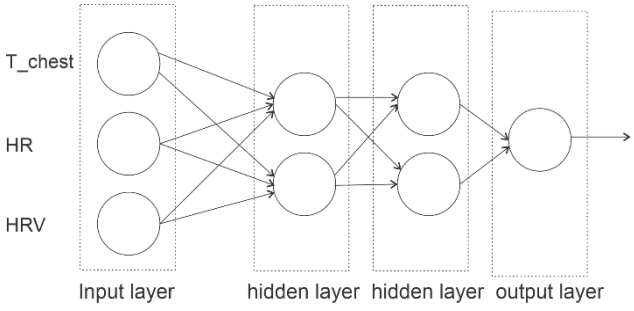
2.1. Prikupljanje i obrada podataka

U istraživanju su korišćeni preuzeti podaci prikupljeni sa više ispitanika tokom kontrolisanih fizičkih aktivnosti u okviru projekta SixthSense. Merene su spoljašnje fiziološke veličine, uključujući temperaturu na površini grudnog koša, srčana frekvencija i varijabilnost srčanog ritma, dok je referentna unutrašnja temperatura tela merena invazivnom metodom (kapsulama). Eksperimenti su obuhvatili više scenarija, sa variranjem uslova okoline i stanja ispitanika (nivo fizičke aktivnosti, hidracije, aklimatizacije), kako bi model mogao da nauči odnose u različitim okolnostima [3, 4]. Prikupljeni sirovi podaci filtrirani su radi uklanjanja šuma, a zatim normalizovani i podeljeni na skup za obuku i testiranje.

2.2. Razvoj modela

Formirana su dva modela neuronskih mreža za predikciju unutrašnje temperature:

1. *Feedforward neuronska mreža (FFNN)* – klasična višeslojna perceptronska mreža, sa ulaznim čvorovima koji predstavljaju trenutne vrednosti fizioloških parametara: temperatura kože - T_chest, srčana frekvencija – HR i varijabilnost srčanog ritma - HRV. Mreža se sastoji od jednog ili više skrivenih slojeva sa nelinearnim aktivacionim funkcijama, i jednim izlaznim čvorom koji estimira unutrašnju telesnu temperaturu – Slika 1.

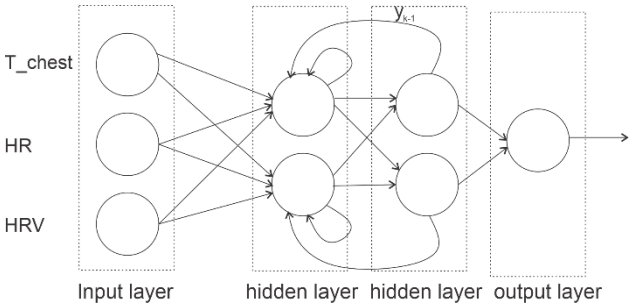


Slika 1. *Struktura feedforward neuronske mreže*

Kao najjednostavnija struktura razmatrana je mreža sa samo dva skrivena sloja sa po jednim neuronom i jedan izlazni neuron. Takođe je isprobana arhitektura sa dva skrivena sloja sa po dva neurona radi povećanja kapaciteta

modela, kao i složenija konfiguracija sa tri skrivena sloja. Poređenjem ovih struktura obezbeđeno je da se ispita uticaj kompleksnosti modela na tačnost estimacije. FFNN model nema internu memoriju stanja odnosno svaka predikcija se vrši samo na osnovu trenutnih ulaza, bez eksplicitnog uvažavanja prethodnih vrednosti signala. Veći broj neurona u skrivenom sloju omogućava modelu da nauči složenije nelinearne odnose, dok minimalistička mreža služi kao baseline za upoređivanje. Sve neuronske jedinice u eksperimentima koristile su nelinarnu aktivacionu funkciju (ReLU), osim izlaznog neurona koji je bio linearni regresor, čime se omogućava predviđanje kontinuirane vrednosti temperature.

2.*Rekurentna feedforward neuronska mreža (RFFNN)* ili proširenje FFNN modela dodatkom jednostavne povratne veze. U RFFNN arhitekturi, originalnom skupu ulaza dodat je još jedan ulaz koji nosi informaciju o prethodnoj predikciji unutrašnje temperature. Na taj način, model pri predviđanju sledeće vrednosti uzima u obzir i neposredno prethodnu izlaznu vrednost, omogućavajući mu osnovni vid "pamćenja" dinamike sistema – Slika 2.



Slika 2. *Rekurentna veštačka neuronska mreža*

Tokom treniranja, za ovaj dodatni ulaz koriste se stvarne prethodno izmerene vrednosti temperature (jer su poznate u trening skupu), dok se prilikom testiranja koristi prethodna predikovana vrednost (feedback petlja). Ovakva jednostavna rekurentna povratna sprega omogućava RFFNN modelu da bolje uhvati vremenske zavisnosti u podacima u poređenju sa običnim FFNN [5, 6].

2.3. Obuka modela

Za treniranje obe mreže korišćen je skup označenih podataka (ulazi sa pripadajućom merenom unutrašnjom temperaturom kao izlazom). Modeli su trenirani metodom nadgledanog učenja, minimizacijom funkcije greške između predikcija mreže i stvarne temperature. Isprobano je više konfiguracija hiperparametara (broj slojeva i neurona, parametri učenja) i različitih funkcija greške MAE, MSE, kao i Huber-ova funkcija kako bi se pronašla optimalna arhitektura za dati problem. Korišćena je k-fold unakrsna validacija, pri čemu su podaci jednog ispitanika ostavljani za validaciju (pristup *Leave-One-Subject-Out*) da bi se proverila sposobnost generalizacije modela. Treniranje je ponavljano za svaku arhitekturu (FFNN i RFFNN), a modeli su upoređeni po performansama na nezavisnom test skupu.

2.4. Evaluacija tačnosti

U radu su izvršene dve vrste evaluacije i odabira modela u cilju analize performansi i izbora optimalnog rešenja. Kod obe arhitekture urađena je evaluacija korišćenjem globalno najboljeg prosečnog modela i evaluacija korišćenjem individualno optimalnog modela za svakog subjekta. U nastavku teksta razmatraće se rezultati dobijeni kod individualno optimalnog modela.

Za kvantitativnu ocenu performansi modela korišćene su standardne metrike regresije: *srednja apsolutna greška* (MAE), *srednja kvadratna greška* (MSE), *koren srednje kvadratne greške* (RMSE) i *koeficijent determinacije* (R^2). MAE i RMSE izražavaju prosečno odstupanje predikcije od stvarne vrednosti (u °C), dok R^2 pokazuje udeo varijanse objašnjen modelom (vrednost bliža 1 označava bolju usklađenost predikcije sa realnim podatkom). Nakon treniranja, za svakog ispitanika izračunate su navedene metrike posebno, a zatim je izvršeno poređenje rezultata FFNN i RFFNN modela.

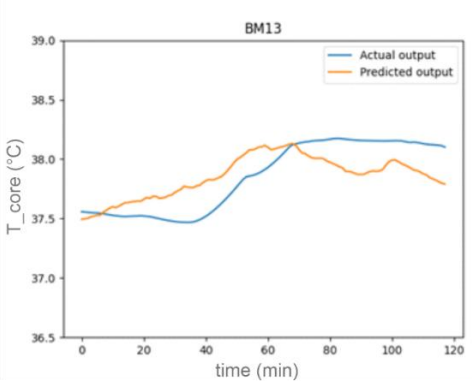
3. REZULTATI I DISKUSIJA

Dobijeni rezultati potvrđuju da model sa povratnom vezom (RFFNN) uspešnije predviđa unutrašnju temperaturu u odnosu na klasičnu neuronsku mrežu. Tabela 1. prikazuje vrednosti evaluacionih metrika za jednog reprezentativnog ispitanika (oznaka *BM13*) pri korišćenju oba tipa modela. RFFNN model ostvaruje znatno manje greške (niže MAE, MSE, RMSE) i viši R^2 u poređenju sa FFNN, što znači da bolje prati stvarne promene temperature.

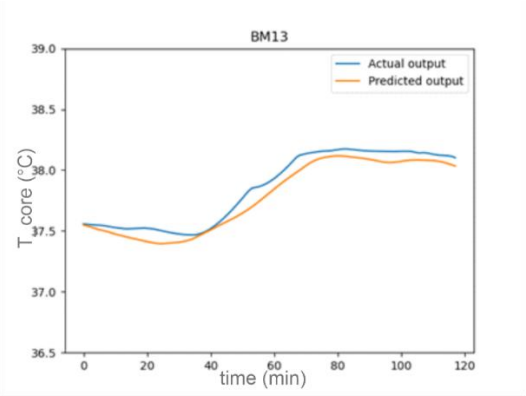
Tabela 1. *Metrike predikcije unutrašnje temperature za ispitanika BM13 (poređenje arhitektura)*

Model	MAE	MSE	RMSE	R^2
FFNN	0.186	0.042	0.205	0.501
RFFNN	0.071	0.006	0.078	0.928

Na slikama ispod prikazane su krive stvarne temperature tela (*Actual output*, plava linija) i predikcije neuronske mreže (*Predicted output*, narandžasta linija) za ispitanika BM13. Slika 3. prikazuje model FFNN. Vidljivo je da postoje izvesna odstupanja kao i da reaguje sa zakašnjenjem na nagli pad temperature. Suprotno tome, Slika 4. predstavlja RFFNN model koji prati unutrašnju temperaturu kroz vreme. Takođe, može se primetiti da su promene pravilnije predviđene.



Slika 3. *Rezultati predikcije za ispitanika BM13 korišćenjem FFNN modela*



Slika 4. *Rezultati predikcije za ispitanika BM13 korišćenjem RFFNN modela*

4. ZAKLJUČAK

U ovom radu prikazana je primena veštačkih neuronskih mreža za estimaciju unutrašnje telesne temperature na osnovu spoljašnjih fizioloških merenja. Upoređene su dve arhitekture – feedforward i rekurentna feedforward neuronska mreža – i pokazano je da dodavanje povratne veze (memorije) značajno poboljšava performanse predikcije. RFFNN model se pokazao bolji u odnosu na običnu feedforward neuronsku mrežu, što potvrđuje hipotezu da uvođenje vremenskog konteksta u modele poboljšava preciznost predikcija. Prednost ovog pristupa jeste sistem za praćenje telesne temperature zasnovan na ovakvim modelima bio bi neinvazivan i pogodan za implementaciju u različite uređaje.

4. LITERATURA

[1] P. N. Stuart Russell, Artifical Intelligence: A Modern Approach, Pearson, 2016..

[2] Y. B. G. H. Yann LeCun, "Deep Learning," *Nature,* 2015.

[3] H. B. Youngjoo Kim, Introduction to Kalman Filter and Its Applications, InTechOpen, 2018.

[4] S. S. K. G. M. F. Shahin Tasoujian, Real-Time Cubature Kalman Filter Parameter Estimation of Blood Pressure Response Characteristics Under Vasoactive Drugs Administration, IEEE, 2020.

[5] W. J. T. S. N. C. S. J. M. R. W. K. J. C. W. A. L. W. S. R. M. R. O. C. J. R. W. H. Mark J Buller 1, Estimation of human core temperature from, PubMed, 2013.

[6] E. W. S. A. A. P. S. D. R. M. R. Reto Niedermann 1, "Prediction of human core body temperature using non-invasive," *Springer Nature Link,* 2013.

Kratka biografija:

Vukašin Pavković rođen je u Sr. Mitrovici 1999. god. Fakultet tehničkih nauka upisuje 2018. godine. Položio je sve ispite predviđene studijskim programom i ispunio sve fakultetske obaveze na master akademskim studijama kontakt: vukasinpavkovic@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.4

DOI: https://doi.org/10.24867/33BE16Pasanovic

ANALIZA ARTEFAKTA KOLABORATIVNOG RAZVOJA SOFTVERA PUTEM VELIKIH JEZIČKIH MODELA ZA UNAPREĐENJE TIMSKOG RADA

ANALYSIS OF COLLABORATIVE SOFTWARE-DEVELOPMENT ARTIFACTS USING LARGE LANGUAGE MODELS TO ENHANCE TEAMWORK.

Halid Pašanović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Efikasnost timova direktno utiče na uspeh organizacija, naročito u oblasti softverskog inženjerstva gde je uspešna saradnja ključna. Ovaj rad detaljno istražuje primenu velikih jezičkih modela (VJM-a) kao alata za analizu i poboljšanje timske dinamike, posebno kroz razvoj deljene svesti (Shared Cognition). Eksperimentalni pristup je korišćen za analizu različitih podataka iz sprint retrospektiva, GitHub-a i Trello platforme kako bi se identifikovali obrasci ponašanja članova tima i predložile konkretne mere za njihovo unapređenje.*

Ključne reči: *VJM, Shared Cognition, timska dinamika, softversko inženjerstvo*

Abstract – *Team efficiency directly impacts organizational success, especially within software engineering contexts, where effective collaboration is critical. This paper comprehensively explores the use of Large Language Models (LLMs) as tools for analyzing and improving team dynamics, emphasizing the development of Shared Cognition. An experimental approach was employed to analyze various data sources, including sprint retrospectives, GitHub histories, and Trello activity logs, identifying team behavior patterns and proposing specific improvement strategies.*

Keywords: *LLM, Shared Cognition, team dynamics, software engineering*

1. UVOD

U savremenom poslovnom okruženju timovi predstavljaju osnovne jedinice koje omogućavaju efikasno rešavanje kompleksnih zadataka zahvaljujući sinergiji različitih veština, znanja i kompetencija svojih članova.

Efikasnost timskog rada direktno utiče na ukupne performanse i konkurentnost organizacije, posebno u tehničkim disciplinama poput softverskog inženjerstva. Ovaj rad istražuje primenu velikih jezičkih modela (VJM-a) kao inovativnog alata za analizu i unapređenje timske dinamike, sa posebnim akcentom na razvoj deljene svesti kao ključnog faktora uspeha.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Nikola Luburić, docent

2. TIMSKA DINAMIKA I SHARED COGNITION

Timska dinamika se odnosi na međusobne interakcije članova tima koje omogućavaju efikasno postizanje zajednički definisanih ciljeva [1]. U okviru ovih interakcija posebno su važni faktori poput komunikacije, koordinacije, poverenja, zajedničkog donošenja odluka, kao i međusobnog razumevanja.

Efikasan timski rad može se analizirati kroz tri osnovne dimenzije:

- Stavovi (Attitudes): Ova dimenzija obuhvata međusobno poverenje, koheziju i kolektivnu efikasnost [2].
- Ponašanja (Behaviors): Odnosi se na konkretne aktivnosti i interakcije koje tim obavlja tokom rada. To uključuje razmenu informacija, pružanje podrške članovima tima u kriznim situacijama i kontinuirano praćenje napretka kako bi se pravovremeno identifikovali i rešavali problemi [2].
- Kognicija (Cognition): Podrazumeva deljenu svest unutar tima, odnosno zajedničko razumevanje uloga, odgovornosti, ciljeva, normi, kao i veština i sposobnosti ostalih članova [2].

2.1. Deljena svest kao ključni aspekt timske dinamike

Deljena svest (Shared Cognition) predstavlja zajedničko razumevanje među članovima tima, koje nastaje kroz kontinuiranu interakciju i saradnju [1]. Ona uključuje dva važna koncepta: deljene mentalne modele i transaktivne memorijske sisteme.

Deljeni mentalni modeli omogućavaju članovima tima da precizno predvide ponašanja i odluke drugih članova na osnovu zajedničkih očekivanja i razumevanja zadataka [3]. Ovakvi modeli su ključni za efikasnu koordinaciju aktivnosti i pravovremenu reakciju na promene u okruženju.

Transaktivni memorijski sistemi se odnose na zajedničku memoriju tima koja omogućava optimalnu raspodelu zadataka prema individualnim sposobnostima i specijalizacijama članova tima [4]. Ovi sistemi pomažu timu da efikasnije koristi svoje unutrašnje resurse, smanjujući redundansu i povećavajući ukupnu produktivnost.

Istraživanja pokazuju da razvijena deljena svest značajno doprinosi adaptabilnosti tima, omogućavajući implicitnu koordinaciju i bržu reakciju na izazove [1]. Nedostatak zajedničke svesti, s druge strane, može rezultirati

nesporazumima, konfliktnim situacijama i smanjenom efikasnošću tima [1].

Kroz upotrebu velikih jezičkih modela, moguće je analizirati i unaprediti deljenu svest u timovima, identifikujući obrasce ponašanja i nudeći konkretne savete za poboljšanje timske dinamike.

3. VELIKI JEZIČKI MODELI (VJM-I)

Veliki jezički modeli (engl. LLM - Large Language Models) predstavljaju kompleksne sisteme veštačke inteligencije bazirane na dubokim neuronskim mrežama, koje omogućavaju obradu, razumevanje i generisanje ljudskog jezika [5]. Ovi modeli koriste ogromne količine tekstualnih podataka za učenje statističkih obrazaca i veza između reči i izraza, čime stiču sposobnost predviđanja, analize i kreiranja novog, smislenog sadržaja [6].

Najvažnija arhitektura na kojoj se zasnivaju savremeni VJM-i je arhitektura transformera, koja je 2017. godine uvedena kao revolucionarno rešenje za obradu prirodnog jezika zahvaljujući primeni mehanizma pažnje (engl. attention mechanism) [7]. Ovaj pristup omogućava modelima da efikasnije prepoznaju i analiziraju veze između reči unutar konteksta, prevazilazeći ograničenja prethodnih tehnika baziranih na rekurentnim ili konvolucionim mrežama.

Zahvaljujući sposobnosti obrade velikih količina podataka, VJM-i poput GPT (engl. Generative Pretrained Transformer) modela, posebno GPT-4 i ChatGPT, pokazali su izuzetnu efikasnost u raznim primenama, uključujući analizu i unapređenje timske dinamike u oblasti softverskog inženjerstva. Ovi modeli mogu detaljno analizirati podatke iz različitih izvora kao što su komunikacija između članova tima, istorije verzionisanja koda i izveštaji sa sastanaka (npr. sprint retrospektive), identifikovati obrasce ponašanja i predložiti konkretne mere za optimizaciju zajedničke svesti i timskog rada.

U kontekstu timske dinamike, primena VJM-a pruža mogućnost dubokog uvida u kvalitet i obrasce komunikacije, omogućavajući blagovremeno prepoznavanje potencijalnih problema i formulisanje preporuka za njihovo rešavanje, što direktno doprinosi boljoj koordinaciji, efikasnosti i ukupnoj uspešnosti timova.

4. METODOLOGIJA

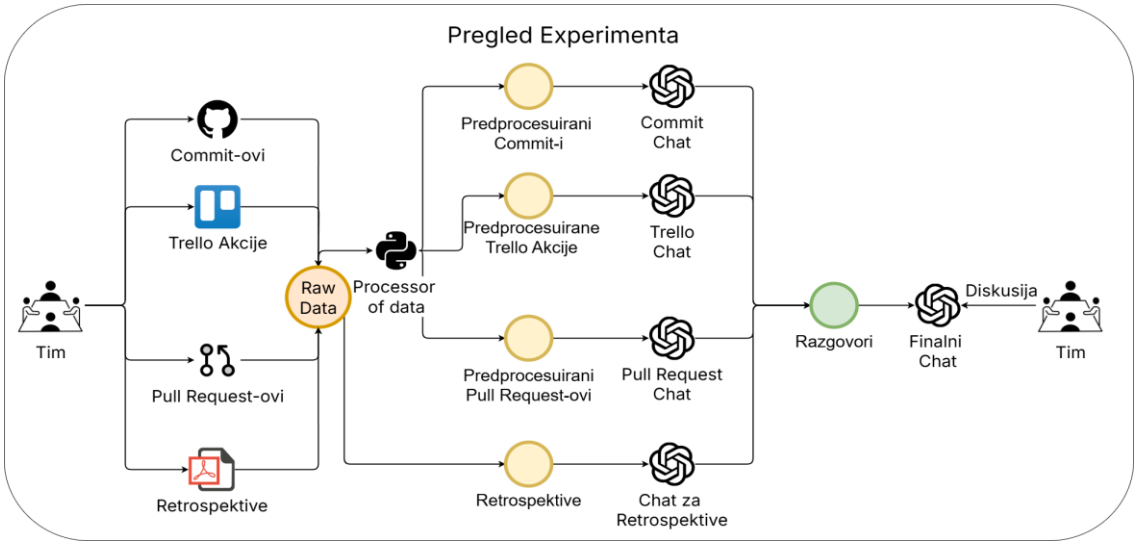
U ovom istraživanju korišćen je eksperimentalni pristup zasnovan na detaljnoj analizi podataka koji dokumentuju različite aspekte timskih aktivnosti. Proces generisanja, prikupljanja, obrade i analize podataka je ilustrovan na slici 1. Eksperimentalna postavka obuhvatila je nekoliko jasno definisanih koraka: prikupljanje, predobradu i analizu podataka pomoću velikih jezičkih modela.

Prvi korak podrazumevao je prikupljanje podataka iz različitih izvora, koji uključuju sprint retrospektive, istorije GitHub komitova, komentare sa pull request-ova, kao i aktivnosti zabeležene putem Trello platforme. Nakon prikupljanja, svi podaci su prošli kroz fazu predprocesiranja, tokom koje su uklonjeni nevalidni, nepotpuni i redundantni podaci, dok je preostali sadržaj strukturiran i formatiran radi efikasnije analize.

Zatim je za svaki pojedinačni tip podataka inicirana zasebna sesija sa VJM-om (u ovom slučaju ChatGPT), čiji zadatak je bio da identifikuje ključne pozitivne i negativne obrasce ponašanja unutar svake od analiziranih kategorija podataka. Dakle, zasebno su obrađivani podaci iz sprint retrospektiva, aktivnosti na GitHub platformi (komitovi i pull request-ovi) i aktivnosti na Trello platformi, čime je omogućena detaljna i specijalizovana analiza za svaku vrstu podataka.

Svakom VJM-u prosleđuje se odlomak iz knjige Scotta Tannenbauma „*Teams That Work: The Seven Drivers of Team Effectiveness*“, konkretno poglavlje 8 — „*Cognitions – Are We on the Same Page*“ [8]. Time modelima obezbeđujemo jasan teorijski okvir i smernice za prepoznavanje relevantnih koncepata pri analizi dostavljenih podataka.

Po završetku pojedinačnih sesija, rezultati dobijeni iz svih inicijalnih analiza objedinjeni su i prosleđeni završnoj (finalnoj) sesiji VJM-a. Finalni ChatGPT je imao zadatak da objedini rezultate prethodnih analiza, pruži integrativan pogled na celokupnu timsku dinamiku, identifikuje dominantne obrasce ponašanja koji značajno utiču na



Slika 1. *Tok koji opisuje kako su podaci generisani, prikupljani, obrađivani i analizirani u eksperimentu*

razvoj zajedničke svesti, te da formuliše konkretne preporuke za unapređenje efikasnosti i koordinacije unutar tima.

Ovakva eksperimentalna postavka omogućila je dubinski uvid u timske interakcije kroz višestepenu analizu, čime su osigurani visok kvalitet i relevantnost dobijenih preporuka.

5. PROMPT FINALNOG GPT-A

Finalni GPT funkcioniše kao meta-analizator: ne radi nad sirovim podacima, već nad sažecima konverzacija prethodnih GPT modela (Pull Request-ovi, commit istorija, Trello aktivnosti, retrospektive), sa ciljem da objedini nalaze i formira celovitu sliku timske dinamike. Listing 1 prikazuje sistemski prompt za dati GPT.

```
Role Description:
You are an assistant specializing in
supporting young software engineering teams.
Your task involves summarizing specific
content from a book about teamwork, with a
focus on Chapter 8 - "Shared Cognitions." It
is crucial that you comprehend the factors
that positively and negatively influence
Shared Cognitions.

Context and Inputs:
Your users, software engineering teams, will
provide you with summaries of conversations
they had with other GPT models. These
conversations will analyze Pull Requests,
Commit History, Trello History and Sprint
Retrospectives, identifying behaviors that
affect Shared Cognition within the team.

Tasks and Output Requirements:
Behavior Analysis:
When requested to analyze these summaries:
List all behaviors that align with the
analysis in question.
For each behavior, specify the parts of the
provided conversations that led to your
conclusion.
Cite Chapter 8 of the book to explain why each
identified behavior influences Shared
Cognition positively or negatively.

By following these guidelines, you will help
software engineering teams understand the
dynamics of Shared Cognition and how to foster
a collaborative environment effectively.
```

Listing 1. *Prompt Finalnog GPT-a*

Prompt je strukturisan kroz tri celine: (1) Opis Uloge – model se pozicionira kao asistent za mlade softverske timove; (2) Contekst i Ulazi – ulaz čine sažeci konverzacija drugih GPT-eva o Pull Request-ovima, commit-ima, Trello istoriji i retrospektivama; (3) Zadatci i specifikacija izlaza – pri analizi model treba da navede sva relevantna ponašanja, za svako pokaže delove sažetaka koji vode zaključku i obrazloži uticaj na deljenu svest pozivanjem na poglavlje 8 knjige „*Teams That Work*“ [8].

Ovakav dizajn prompta obezbeđuje dosledne, teorijski utemeljene izlaze i omogućava dublju, precizniju sintezu timskih obrazaca od pojedinačnih modela.

6. REZULTATI ISTRAŽIVANJA

Rezultati eksperimentalne analize jasno pokazuju da VJM efikasno identifikuje pozitivne obrasce timskog ponašanja, uključujući jasnu i transparentnu komunikaciju, konstruktivne i detaljne revizije koda, kao i proaktivnu međusobnu podršku članova tima. Takođe, identifikovani su i negativni obrasci poput nedovoljne jasnoće u raspodeli odgovornosti, neprecizne komunikacije i nedostatka redovnog praćenja aktivnosti.

Za potrebe procene kvaliteta identifikovanih obrazaca od strane VJM-a, uvedena je skala ocenjivanja u rasponu od 1 do 3, gde pojedinačna ocena znači:

- 1: Model je u potpunosti pogrešno protumačio ulaz.
- 2: Odgovor delimično odgovara kontekstu deljene svesti.
- 3: Analiza je temeljna i adekvatno adresira zadati problem.

Pregledači izlaza modela primenili su ovu skalu prilikom evaluacije rezultata dobijenih za svaki od analiziranih setova podataka. Ocene su dodeljene posebno za slučaj kada su podaci individualno obrađivani, zatim za slučaj kada su podaci prosleđivani u segmentima, te na kraju za rezultate finalnog GPT-a (meta-analizatora).

6.1 Git Commit GPT Model

Rezultati analize pojedinačnih Git commit-ova jasno ukazuju na sposobnost GPT modela da precizno identifikuje pozitivne i negativne obrasce timskog ponašanja čak i na nivou pojedinačnih akcija programera. Pregledači izlaza modela ocenili su da je model u 31 slučaju (94%) ponudio analizu koja je ocenjena kao potpuni pogodak (ocena 3), dok su preostala dva ocenjena kao delimičan pogodak (ocena 2).

Kada je analiza vršena nad segmentima commit-ova, rezultati evaluacije pokazuju još viši nivo uspešnosti modela. Ukupno je analiziran 21 segment, pri čemu su svi segmenti dobili maksimalnu ocenu (ocena 3). Ovaj pristup omogućio je jasnije prepoznavanje kontekstualnih veza između različitih commit-ova, kao i bolju identifikaciju dugoročnih pozitivnih trendova u timskoj komunikaciji, koordinaciji i rešavanju problema.

6.2 Pull Request GPT Model

Analizom individualnih Pull Request-ova utvrđeno je da je u 8 od ukupno 9 slučajeva (89%) model pružio analizu koja je ocenjena kao potpun pogodak (ocena 3), dok je u jednom slučaju analiza ocenjena kao delimičan pogodak (ocena 2).

Kada su podaci analizirani u formi segmenata, rezultati su još konzistentniji – od ukupno 9 segmenata, svi su ocenjeni maksimalnom ocenom (3), što ukazuje na visok nivo preciznosti i pouzdanosti GPT modela pri obradi kontekstualno bogatijih ulaza.

6.3 Trello GPT Model

Rezultati analize pojedinačnih Trello kartica pokazali su da je model u 20 od ukupno 21 analiziranog slučaja (95,2%) pružio analizu ocenjenom kao potpuni pogodak (ocena 3), dok je jedna analiza ocenjena kao delimičan pogodak (ocena 2).

Segmentna analiza podataka sa Trello platforme pokazala je potpunu konzistentnost i tačnost, jer je svih 8 analiziranih segmenata ocenjenom maksimalnom ocenom (3), čime je potvrđen visoki nivo sposobnosti GPT modela da efikasno prepozna timske obrasce u širem kontekstu.

6.4 GPT Model za Retrospektive

Analiza pojedinačnih rečenica retrospektiva pokazala je relativno visoku pouzdanost modela. Od ukupno 102 rečenice, u 74 slučaja (72,5%) pružena je analiza ocenjena kao potpuni pogodak (ocena 3), 16 slučajeva (15,7%) ocenjeno je kao delimičan pogodak (ocena 2), dok je 12 slučajeva (11,8%) ocenjeno kao potpuni promašaj (ocena 1).

U segmentnoj analizi retrospektiva, evaluacija je pokazala izuzetnu stabilnost i preciznost GPT modela. Svi analizirani segmenti (ukupno 4) dobili su maksimalnu ocenu (ocena 3), ukazujući na efikasnost modela u kontekstualno agregiranim analizama.

6.5 Finalni GPT Model

Za razliku od prethodnih modela koji su analizirali isključivo sirove podatke, finalni GPT model funkcioniše kao meta-analizator: njegov ulaz predstavljaju konverzacije koje su već prošle kroz pregled ranijih modela.

Tokom analize, model je imao zadatak da izdvoji sve pozitivne segmente koji osnažuju deljenu svest i sve negativne segmente koji je narušavaju. Od ukupno 18 detektovanih ponašanja, 17 je ocenjeno najvišom ocenom 3, dok je jedno dobilo ocenu 1, što ukazuje na visoku tačnost i konzistentnost u klasifikaciji složenih obrazaca.

Pored kategorizacije ponašanja, od modela je zatraženo da na skali 1-100 proceni nivo deljene svesti u timu. Dok je GPT model za retrospektive u finalnoj retrospektivi svest ocenio sa 90, meta-analizator je, uključujući širi kontekst celokupnih konverzacija, dao znatno niži rezultat – 68. Ova razlika osvetljava jaz između percepcije GPT modela na osnovu subjektivne percepcije tima i objektivno izmerenog stanja, čime model pruža dragocene uvide koji prevazilaze ono što pojedinačni GPT-evi mogu da uoče.

Sumirano, finalni GPT meta-analizator dokazuje sposobnost da sintetizuje multimodalne izvore podataka i isporuči dublje, preciznije procene stanja timske dinamike, u odnosu na individualne GPT-e.

7. ZAKLJUČAK

Upotreba VJM-a kao analitičkih alata za unapređenje timske dinamike pokazala se kao izuzetno efikasan pristup. Rezultati istraživanja ukazuju na značajan potencijal VJM-a za razvoj zajedničke svesti u timu, omogućavajući blagovremeno identifikovanje problema i ciljano rešavanje izazova.

Istraživanje pokazuje da primena VJM-a na pojedinačnim skupovima podataka daje solidne uvide u deljenu svest tima, ali da se objedinjavanjem analiza više skupova i njihovim prosleđivanjem završnom VJM-u – koji funkcioniše kao meta-analizator – problemi u timu otkrivaju jasnije, što rezultira znatno kvalitetnijom analizom. Buduća istraživanja trebalo bi proširiti uključivanjem dodatnih izvora podataka i proverom primenljivosti ovog pristupa u raznovrsnijim timovima i organizacionim okruženjima, a istovremeno usmeriti se na razvoj preciznijih modela kako bi se utvrdilo donose li pouzdanije zaključke.

8. LITERATURA

[1] Understanding and improving teamwork in organizations: A scientifically based practical guide – Eduardo Salas, Marissa L. Shuffler, Amanda L. Thayer, Wendy L. Bedwell, Elizabeth H. Lazzara.

[2] Factors that influence Teamwork – Julie V. Dinh and Eduardo Salas.

[3] Shared Mental Models: A Conceptual Analysis. Catholijn M. Jonker, M. Birna van Riemsdijk, Bas Vermeulen - https://www.researchgate.net/publication/221456658_Shared_Mental_Models_-_A_Conceptual_Analysis

[4] Routines and transactive memory systems: Creating, coordinating, retaining, and transferring knowledge in organizations - https://www.sciencedirect.com/topics/psychology/transactive-memory#:~:text=A%20transactive%20memory%20system%20(TMS,information%20than%20they%20individually%20possess.

[5] What are large language models (LLMs)? - https://www.techtarget.com/whatis/definition/large-language-model-LLM

[6] Large Language Model - https://en.wikipedia.org/wiki/Large_language_model

[7] Attention Is All You Need - https://arxiv.org/abs/1706.03762

[8] Tannenbaum S., Salas E. (2021). Teams that work: The seven drivers of team effectiveness. Oxford University Press.

Kratka biografija:

Halid Pašanović rođen je 2000. godine u Prijepolju, gde je završio osnovnu i srednju školu. Studije na smeru Računarstvo i automatika na Fakultetu tehničkih nauka započeo je školske 2019/20. godine, a osnovne akademske studije uspešno je okončao 2023. godine. Iste godine upisao je master akademske studije Primenjene računarske nauke i informatika, na modulu Elektronsko poslovanje.

kontakt: halidpasanovic1000@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.3
DOI: https://doi.org/10.24867/33BE17Sekeres

RAZVOJ QGIS PLUGINA ZA VIZUALIZACIJU I ANALIZU KRETANJA OBJEKTA U PROSTORU

DEVELOPMENT OF A QGIS PLUGIN FOR VISUALIZATION AND ANALYSIS OF OBJECT MOVEMENT IN SPACE

Ivana Sekereš, Srđan Popov, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je prikazan proces razvoja QGIS plugina namenjenog vizualizaciji i analizi kretanja objekta u prostoru, ilustrovan na primeru simulacije putanje bespilotne letelice (UAV) unutar definisane oblasti interesa. Plugin je razvijen u programskom jeziku Python i integriše PostgreSQL sistem za upravljanje bazama podataka sa PostGIS ekstenzijom, čime se omogućava efikasno skladištenje, obrada i analiza prostornih podataka.*

Ključne reči: *QGIS, GIS, plugin, UAV, Planiranje putanje pokirvanja, PostgreSQL, PostGIS, Python*

Abstract – *The paper presents the development process of a QGIS plugin designed for the visualization and analysis of object movement in space, illustrated through a simulation of an unmanned aerial vehicle (UAV) trajectory within a defined area of interest. The plugin is developed in Python and integrates a PostgreSQL database system with the PostGIS extension, enabling efficient storage, processing, and analysis of spatial data.*

Keywords: *QGIS, GIS, plugin, UAV, Coverage Path Planning, PostgreSQL, PostGIS, Python*

1. UVOD

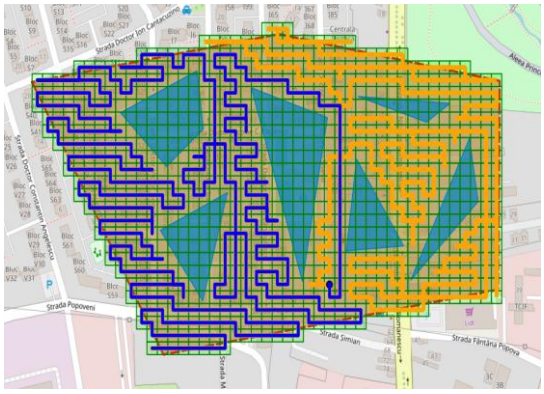
Geografski informacioni sistemi (GIS) predstavljaju skup alata, tehnologija i metoda namenjenih za prikupljanje, skladištenje, obradu, analizu i vizualizaciju prostornih i geografskih podataka. GIS tehnologija se primenjuje u naučnim istraživanjima, upravljanju resursima i prostornom planiranju u različitim oblastima. Da bi se GIS koncepti primenili u praksi koriste se specifični softverski alati, poznati i kao GIS aplikacije. Jedna od najpoznatijih i najraširenijih aplikacija otvorenog koda je QGIS.

Cilj rada je razvoj dodataka (eng. plugins) koji omogućavaju proširenje standardnih funkcionalnosti QGIS platforme. Rad prikazuje praktičnu primenu korišćenja QGIS-a u analizi kretanja objekta u prostoru. U nastavku rada, radi lakše terminološke doslednosti, koristiće se engleski izraz *plugins* za označavanje dodataka.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Popov, redovni profesor

2. TEHNOLOGIJE

U razvoju plugina korišćene su sledeće tehnologije: QGIS (za vizualizaciju i rad sa prostornim podacima), Python (za implementaciju logike), PostgreSQL sa PostGIS ekstenzijom (za čuvanje prostornih podataka). Algoritmi planiranja kretanja koji su obrađeni u radu uključuju Nearest Neighbor (NN) i A* algoritam, dok je vizualizacija kretanja realizovana putem animacije u QGIS interfejsu. Plugin omogućava povezivanje sa bazom, unos i čuvanje koordinata, kao i prikaz animacije putanje, kao što je to prikazano na slici 1.



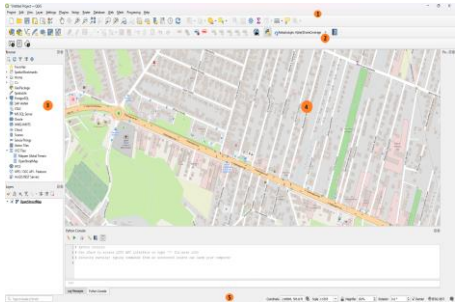
Slika 1. *Generisana putanja UAV-a*

3. QGIS

QGIS je besplatan geoinformacioni sistem (GIS) otvorenog koda koji korisnicima omogućava kreiranje, uređivanje, vizualizaciju, analizu i objavljivanje geoprostornih podataka [2].

3.1 QGIS GUI

Interfejs QGIS-a, prikazan na slici 2, je dizajniran tako da korisnicima omogući jednostavan pristup različitim funkcionalnostima i alatima za rad sa geoprostornim podacima [2]. Razvijeni plugin dodatno proširuje interfejs, uvodeći nove opcije za vizualizaciju i analizu kretanja objekata.



Slika 2. *QGIS GUI*

3.2 Slojevi

U QGIS-u se podaci organizuju kroz **slojeve** (eng. *layers*) koji mogu biti vektorski (tačke, linije, poligoni) ili rasterski (satelitski snimci, digitalni modeli terena i sl.). Vizualizacija i analiza podataka zasnivaju se na kombinaciji slojeva.

3.3 Pluginovi

Pluginovi omogućavaju proširenje osnovnih funkcionalnosti QGIS-a. Mogu se instalirati iz zvaničnog repozitorijuma ili razvijati lokalno, najčešće u programskom jeziku Python korišćenjem PyQGIS API-ja. Upravljanje pluginovima obavlja se preko Plugin Manager-a, u kojem se oni instaliraju, aktiviraju i održavaju.

3.4 Plugin Manager

Plugin Manager je centralni alat za upravljanje pluginovima u QGIS-u. Pristupa mu se direktno putem glavnog menija *Plugins > Manage and Install Plugins*. Kroz njega korisnici mogu pretraživati i instalirati pluginove, dodavati privatne repozitorijume, pregledati metapodatke (npr. detalje o autoru, verziji, opisu i zavisnostima (eng. dependencies) svakog plugina, kao i uključivati/isključivati instalirane pluginove.

3.5. Python

Python je interpretirani, objektno-orijentisani programski jezik visokog nivoa sa dinamičkom tipizacijom. Njegova jednostavna i čitljiva sintaksa, zajedno sa moćnim ugrađenim strukturama podataka, čini ga pogodnim za brzi razvoj aplikacija, skriptovanje i povezivanje postojećih komponenti, uz smanjenje troškova održavanja koda [4].

3.5.1 Python i QGIS

QGIS poseduje integrisanu Python konzolu, koja omogućava direktno izvršavanje Python komandi u okviru QGIS okruženja. Ova konzola se otvara putem menija Plugins > Python Console. Python jezik se široko koristi za implementaciju pluginova u QGIS-u. Za razvoj pluginova koristi se PyQGIS API, putem kojeg QGIS pruža developerima programski interfejs za interakciju i modifikovanje osnovnih funkcija QGIS-a, ali i grafičkog interfejsa.

3.6. PostgreSQL

PostgreSQL, odnosno Postgres, predstavlja objektno-relacioni sistem za upravljanje bazama podataka (eng. Object-Relational Database Management System – ORDBMS) otvorenog koda (eng. open source) koji, za razliku od klasičnih relacionih baza podataka, poseduje mogućnost čuvanja objekata i vektora.

3.7. PostGIS ekstenzija

PostGIS je ekstenzija za PostgreSQL bazu podataka, koja omogućava skladištenje, upravljanje i analizu prostornih podataka direktno u bazi. PostGIS omogućava skladištenje različitih vrsta prostornih podataka, uključujući tačke, linije i poligone, u oba formata: dvodimenzionalnom (2D) i trodimenzionalnom (3D) [3].

3.8. Planiranje kretanja

Planiranje kretanja predstavlja proces određivanja optimalne putanje kretanja objekta kroz određeni prostor, uzimajući u obzir različite kriterijume kao što su minimalna dužina puta, vreme kretanja ili izbegavanje prepreka. Planiranje može biti realizovano od strane čoveka ili mašine koja ima mogućnost planiranja svog kretanja. Ukoliko je onaj koji planira mašina, tada govorimo o algoritmima planiranja kretanja [5].

3.8.1 Primene u geoinformacionim sistemima

Coverage Path Planning nalazi široku primenu u oblasti geoinformacionih sistema (GIS). Geoinformacioni sistemi omogućavaju prikupljanje i obradu prostornih podataka potrebnih za planiranje optimalnih putanja.

Upotrebom bespilotnih letelica, naročito dronova, proces prikupljanja prostornih podataka postao je znatno efikasniji, brži i ekonomičniji. Na ovaj način GIS industrija ostvaruje velike benefite jer čine da skupljanje prostornih podataka bude jednostavnije i pristupačnije. Tradicionalni način sakupljanja podataka podrazumevao je iznajmljivanje letelica i pilota, kao i postavljanje instrumenta za snimanje za letelicu, što je zahtevalo složenu pripremu i visoke troškove. Velika prednost upotrebe bespilotnih letelica je što mogu preleteti veliku površinu za nekoliko sati i prikupiti podatke visoke rezolucije. Pored toga moguće je snimiti teško dostupne i rizične lokacije bez ugrožavanja ljudi. Podaci prikupljeni dronovima lako se integrišu u GIS softvere, kao što je to QGIS.

3.9. Algoritmi

U radu su za planiranje putanje kretanja objekta u prostoru, sa ciljem pokrivanja cele površine od interesa, korišćeni algoritmi Nearest Neighbor (NN) i A* (A star), koji omogućavaju određivanje optimalnih ili približno optimalnih putanja u okviru definisanog prostora interesa. Nearest Neighbor (NN) algoritam je heuristički pristup kojim se iz trenutne pozicije bira najbliža neposećena tačka kao sledeća destinacija.

A* algoritam se primenjuje u svrhe određivanja najkraćeg puta između dva čvora u grafu. Predstavlja unapređenje Dijkstrinog algoritma i smanjuje broj čvorova koje je potrebno posetiti odnosno ispitati. Ovaj algoritam koristi heuristički pristup za procenjivanje cene pružanja do ciljnog čvora. Algoritam održava celo stablo i iterativno se pruža ka jednom čvoru u svakom koraku. Cena pružanja do sledećeg čvora se određuje na osnovu najmanje pređenog rastojanja, vremena putovanja i slično, u zavisnosti od zahteva praktične namene. Svakom iteracijom odnosno svakim korakom, algoritam određuje cenu putanje sledećom formulom:

$$f(n) = g(n) + h(n) \quad (1)$$

Gde je n sledeći čvor, $g(n)$ cena putanje od početnog čvora do n, i $h(n)$ heuristička funkcija koja određuje optimalan put od n ka ciljnom čvoru. $h(n)$ se smatra prihvatljivom cenom, što znači da nikada neće preceniti dostizanje cilja, odnosno uvek predstavlja najnižu granicu moguće cene kretanja [1].

4. KREIRANJE PLUGINA

Plugin u QGIS-u može se kreirati na dva načina: korišćenjem alata Plugin Builder, koji automatski generiše osnovnu strukturu projekta ili ručno definisanjem potrebnih fajlova i direktorijuma. Plugin Builder je veoma koristan jer ubrzava početnu fazu razvoja. Ručno kreiranje plugina zahteva poznavanje osnovne strukture i obaveznih komponenti koje svaki plugin mora da sadrži.

4.1. Struktura tipičnog plugina

Pluginovi se sastoje od seta fajlova koji moraju da prate standardnu strukturu. Plugin Manager upravlja pluginovima i oni se učitavaju kada pokrenemo QGIS program.

Plugin mora da sadrži sledeće fajlove:

- metadata.txt: tekstualni fajl koji sadrži informacije o pluginu. Ove informacije su vidljive u Plugin Manageru.
- __init__.py: koji se poziva od strane Plugin Managera i učitava glavni fajl. Ovaj fajl se učitava prvi i on sadrži funkciju classFactory() koja kreira instancu glavne plugin klase.
- main.py: glavni fajl definiše glavnu plugin klasu i očekivano je da ima minimum tri metode.
    - __init__() metoda koja daje pristup QGIS interfejsu.
    - initGui() metoda koja se poziva kada se plugin učita.
    - unload() metoda koja se poziva kada se plugin uklanja.

4.3. Proširenje korisničkog interfejsa

Korisnički interfejs QGIS-a može se dodatno proširiti instalacijom ili razvojem pluginova. Kada se plugin integriše, on ne utiče na već postojeći GUI programa, već ga dodatno proširuje kreiranjem novih elemenata i funkcionalnosti.

5. DIZAJN I IMPLEMENTACIJA PLUGINA

Cilj kreiranja plugina je omogućiti vizualizaciju i analizu kretanja objekta na mapi. U ovom primeru, realizovana je simulacija kretanja drona na osnovu koordinata koje definiše korisnik. Plugin omogućava praćenje kretanja objekta i analizu putanje u realnom vremenu na osnovu prethodno zadatih podataka.

Podaci o kretanju drona se čuvaju u PostgreSQL bazi. Plugin treba da omogući animaciju kretanja drona na mapi, upisivanje i čuvanje novih koordinata i podataka u bazu. Plugin ne obuhvata kontrolu stvarnog drona, već predstavlja simulaciju potencijalnog kretanja na osnovu odabranih koordinata.

5.1. Konfiguracija radnog okruženja

Za implementaciju plugina korišćene su sledeće komponente:

- QGIS verzija 3.40 LTR (kodnog imena Bratislava), korišćen za vizualizaciju i obradu prostornih podataka.
- Python, koji dolazi integrisan sa QGIS instalacijom na Windows operativnom sistemu i koristi se za razvoj.
- PostgreSQL sa PostGIS ekstenzijom, koji obezbeđuje čuvanje i prostornu obradu podataka, instaliran i konfigurisan unutar Docker kontejnera radi lakše portabilnosti i izolacije okruženja.

5.1.1 Docker

Docker je platforma za kontejnerizaciju koja omogućava pokretanje aplikacija u izolovanom okruženju, odnosno kontejneru (eng. container). U okviru ovog projekta, PostgreSQL baza podataka pokrenuta je unutar Docker kontejnera i koristi se specifična slika postgres_qgis. Ova slika je kreirana tako da u sebi sadrži zvaničnu PostgreSQL sliku (postgres:latest), čime omogućava jednostavno postavljanje baze sa svim osnovnim funkcionalnostima PostgreSQL-a.

5.1.2. Struktura baze podataka

U bazi se čuvaju podaci o poligonima i parametrima vezanim za kretanje, tabela sastoji se od sledećih kolona iz tabele 4.1.

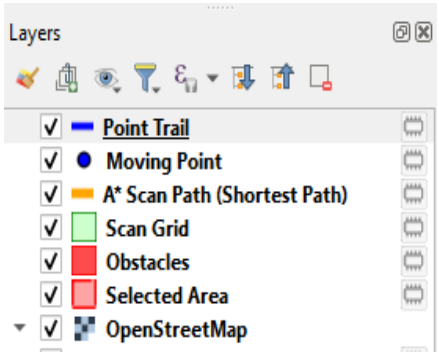
Tabela 4.1. *Opis podataka koji su čuvaju u tabeli polygon_data*

Kolona	Opis
id	Jedinstveni ID broj poligona.
geometry	Koordinate granica poligona.
area	Površina u kvadratnim metrima.
mov_duration	Vreme potrebno dronu da pokrije ceo poligon, u minutama.
path	Putanja drona kroz poligon, formatirana kao JSON.
flight_height	Visina leta drona u metrima.
camera_flow	Brzina snimanja kamere, izražena u fps.
image_overlap	Procenat preklapanja između slika (0–100%)
strategy_cost	Procena troška misije.
ground_sampling_distance	Rezolucija slike u cm po pikselu.
created_at	Datum i vreme kreiranja.

Polje geometry je tipa GEOMETRY(POLYGON, 4326) predstavlja definiciju kolone u prostornoj bazi podataka. Poligon označava tip geometrije, što znači da se u tom polju mogu čuvati isključivo poligoni.

5.2. Slojevi

U okviru projekta, slojevi služe kao osnovni način organizacije i prikaza prostornih podataka potrebnih za simulaciju kretanja drona. Svaki sloj nosi određeni tip informacija i omogućava korisniku da prikaže različite aspekte projekta unutar QGIS interfejsa. Pored osnovnih slojeva koji definišu poligon i putanju, moguće je dodati i slojeve sa preprekama, čime se omogućava simulacija realnog okruženja u procesu planiranja kretanja letelice.



Slika 3: *Prikaz aktivnih slojeva u projektu*

5.2.1 OpenStreetMap

U okviru projekta korišćena je OpenStreetMap (OSM), jer predstavlja besplatnu i otvorenu mapu sveta. OSM nudi detaljne geografske podatke o ulicama, zgradama, granicama, rekama i drugim objektima. Podaci su dostupni pod licencom koja omogućava slobodnu upotrebu, izmene i distribuciju.

5.2.2 Generisanje mreže

Pre nego što se putanja može odrediti, generiše se mreža koja obezbeđuje potpunu pokrivenost poligona. Svaka ćelija u mreži predstavlja pojedinačnu sliku koju bi bespilotna letelica fotografisala u toku kretanja po putanji. Funkcionalnost kreiranja mreže sastoji se od dva važna koraka: računanje dimenzija ćelija i generisanje mreže sa pripadajućom putanjom.

5.2.4 Generisanje putanje algoritmom

Pre nego što dron počne simulirano kretanje, putanja se izračunava korišćenjem algoritma za planiranje kretanja. Algoritam generiše redosled tačaka koje dron treba da poseti, a zatim se ove tačke povezuju linijama u vektorskom sloju, čime se dobija definisana putanja.

5.2.5 Simulacija kretanja

Vektorski slojevi za kretanje drona predstavljaju tačke i linije koje definišu putanju drona. Svaka tačka sadrži koordinate, a linije povezuju te tačke kako bi se vizualizovala simulirana putanja. Tačke se prikazuju u realnom vremenu, a dron se kreće od početne ka završnoj tački, prateći generisanu putanju.

5.3. Metapodaci

Plugin podržava definisanje metapodataka (naziv, opis funkcionalnosti, verzija, autor i slično) koji se unose u fajl *metadata.txt*. Na osnovu ovih podataka QGIS omogućava identifikaciju, pretragu i prikaz osnovnih informacija o pluginu u listi dodataka.

6. REZULATI

Razvijeni QGIS plugin omogućava simulaciju kretanja bespilotne letelice (UAV) unutar definisanog poligona od interesa. Rezultati pokazuju da plugin pruža jednostavnu i interaktivnu vizualizaciju kretanja objekata, uz fleksibilnost za dalje proširenje i primenu u različitim oblastima.

7. ZAKLJUČAK

Implementacijom ovog rešenja pokazano je da QGIS, kao fleksibilna platforma otvorenog koda, omogućava razvoj specijalizovanih alata koji mogu unaprediti proces analize prostornih podataka i planiranja kretanja. Pre svega, plugin je u potpunosti integrisan u QGIS okruženje, što korisnicima omogućava da koriste poznate GIS alate bez potrebe za dodatnim softverom. Iako je u radu fokus bio na simulaciji kretanja UAV-a, potencijalne primene razvijenog rešenja prevazilaze ovaj domen. Plugin se može koristiti u poljoprivredi (planiranje ruta za prskanje i mapiranje parcela), urbanizmu i prostornom planiranju (analiza kretanja vozila i pešaka), ekologiji (monitoring zaštićenih područja i praćenje kretanja životinja), kao i u bezbednosnim i spasilačkim misijama.

6. LITERATURA

[1] Hart, P.E., Nilsson, N.J., Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths.

[2] QGIS Documentation, https://qgis.org (pristupljeno u septembru 2025.)

[3] PostGIS Documentation, https://postgis.net/documentation/ (pristupljeno u septembru 2025.)

[4] Python Documentation, https://www.python.org/ (pristupljeno u septembru 2025.)

[5] Enric Galceran and Marc Carreras. A Survey on Coverage Path Planning for Robotics.

Kratka biografija:

Ivana Sekereš rođena je 01.10.1995. u Somboru. Završila je „Fakultet tehničkih nauka" u Novom Sadu, smer Računarstvo i automatika, i 2020. godine stekla zvanje diplomirani inženjer elektrotehnike i računarstva. Iste godine upisala je master akademske studije na smeru „Primenjene računarske nauke i informatika – Elektronsko poslovanje". kontakt: sankovicivana27@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE18Ninkovic

ORKES CONDUCTOR – POREĐENJE PERFORMANSI SA APACHE KAFKA

ORKES CONDUCTOR – PERFORMANCE COMPARISON WITH APACHE KAFKA

Jelena Ninković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu opisani su Orkes Conductor kao primer orkestracije događajima i Apache Kafka kao primer koreografije. Urađena je analiza i poređenje performansi ova dva alata. Rad uključuje i opis implementacije oba alata, napisane u programskom jeziku Java, kao i poređenje redova koji predstavljaju osnovnu strukturu podataka na koju se alati oslanjaju.*

Ključne reči: *Conductor, zadatak, razmena poruka, Apache Kafka, red*

Abstract – *This paper describes Orkes Conductor as an example of event orchestration and Apache Kafka as an example of choreography. A performance analysis and comparison of two tools is provided. This paper also includes description of implementation of both tools, written in Java programming language, as well as a comparison of queues, which represent the fundamental data structure these tools rely on.*

Keywords: *Conductor, task, messaging, Apache Kafka, queue*

1. UVOD

Mikroservisna arhitektura (MSA) predstavlja arhitekturalni dizajn šablon koji je uveden da reši probleme oko horizontalne skalabilnosti, dostupnosti, modularnosti i agilnosti arhitekture u tradicionalnim monolitnim sistemima. Aplikacija se razvija kao skup malih servisa [1], gde je svaki nezavisno razvijan, testiran, ažuriran, skaliran i deploy-ovan i komunicira sa ostatkom preko jednostavnih mehanizama, najčešće HTTP poziva. Servisi su tako definisani da svaki ima sopstvene entitete i bazu podataka, što čini da promene u bazi podataka jednog servisa nisi vidljive u bazi drugog servisa. Ukoliko dođe do rollback-a transakcije zbog greške u jednom od servisa povratak na pređašnje stanje nije moguć jer su u pitanju distribuirane transakcije.

Da bi se rešio problem uvodi se SAGA [2] šablon. U slučaju greške okida se sekvenca rollback događaja, od jednog servisa ka drugom, u obrnutom redosledu. SAGA šablon može biti implementiran korišćenjem koreografije događaja i orkestracionim tehnikama.

U slučaju koreografije događajima, svaki mikroservis radi zasebno i kada završi lokalnu transakciju emituje događaj, koji drugi servisi osluškuju sa ciljem da započnu svoje transakcije.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Popov, red. prof.

Proces se nastavlja sve dok poslednji servis ne emituje nijedan događaj, što predstavlja kraj transakcije.

Kod orkestracije postoji centralni orkestrator, koji se ponaša kao „roditeljski“ servis, sluša sve događaje koje emituju lokalne transakcije mikroservisa i na osnovu događaja okida sledeću lokalnu transakciju u drugom mikroservisu ili servisima.

2. ORKES CONDUCTOR

Orkes Conductor [3] je radni okvir koji je razvijen povrh Netflix Conductor-a. Pored osnovnih funkcionalnosti koje je nudila platforma dodate su funkcionalnosti i poboljšanja i cilju lakšeg deploy-ovanja i upravljanja tokovima u produkcionom okruženju.

2.1. Osnovni pojmovi

Proces orkestriranja korišćenjem Conductor-a obuhvata korišćenje tri osnovna koncepta: zadatke, radnike i tokove.

Zadatak predstavlja jedinicu posla ili korak u toku, poput kreiranja HTTP poziva, slanja mejla, procesiranja podataka ili izvršavanja poslovne logike. Predstavlja osnovnu gradivnu jedinicu toka i dodatno može biti podeljen na operatore, sistemske zadatke ili radnike za sopstveni kod.

Radnici su kod zadužen za izvršavanje zadataka. Za sistemske zadatke i operatore je zadužen Conductor server, dok zadacima koje je definisao korisnik upravljaju aplikacije koje ih implementiraju. Nakon što radnik kontaktira server da primi i izvrši zakazane zadatke Conductor će proslediti ulazne parametre zadatka radniku i preuzeti izlazne podatke nakon završetka.

Tok se definiše kao kolekcija zadataka i operatora, koji specificiraju redosled i izvršenje definisanih zadataka. Ova orkestracija se dešava u hibridnom ekosistemu koji objedinjuje serverless funkcije, mikroservise i monolitske aplikacije. Kako je Conductor jezički agnostičan orkestracija može biti izvršena u bilo kom programskom jeziku. Workflow scheduler omogućava tokovima da budu pokrenuti po određenom rasporedu. Ovo daje mogućnost da tokovi budu konfigurisani da se pokreću u željenoj učestalosti i da se prilikom kreiranja rasporeda bira verzija toka.

2.2. AI zadaci

Orkes Conductor omogućava kreiranje aplikacija koje koriste generativne AI modele i vektorske baze podataka. Generativni AI je tip veštačke inteligencije sposoban za kreiranje novog „čovekolikog“ sadržaja na osnovu pre-treniranih modela koji su bili izloženi velikim količinama

sličnog sadržaja. Ljudska interakcija je i dalje potrebna da se modeli usmere šta treba da bude generisano - usmeravaju se slanjem promptova (tekstualnih instrukcija). Odgovor Gen-AI modela je isto tekstualan, i ovi modeli se nazivaju LLM. LLM su deep learning algoritmi obučeni na velikoj količini podataka. Mogu da obavljaju razne NLP zadatke, poput generisanja, prevođenja, chatbot-ova, AI asistenata i sl.

2.3. Alert zadaci

Alert zadaci su poseban vid zadataka koji imaju ulogu da šalju notifikacije ili upozorenja na osnovu određenih uslova ili događaja u sistemu.

2.4. Rukovalac događajima

Rukovalac događajima procesira dolazeće poruke i izvršava akcije na osnovu njihovih detalja. Okidač može biti okinut od strane narednih akcija – complete task, terminate workflow, update variables, fail task i start workflow.

2.5. Kontrola pristupa

Orkes Conductor nudi kontrolu pristupa baziranu na ulogama (RBAC) korisnicima Orkes platformi, kao i aplikacijama koje koriste Conductor API. RBAC obezbeđuje pristup metapodacima tokova, zadataka, tajni, promenljivama okruženja, integracijama, promptovima, korisničkim formama, rukovaocima događaja, rasporedima, webhook-ovima i domenima.

Korisnik predstavlja čoveka koji interaguje sa Conductor-om preko Orkes platforme, i autentifikovan je korišćenjem SSO provajdera ili mejla/lozinke. Svaki korisnik može imati jednu ili više dodeljenih uloga.

Grupa predstavlja set korisnika, i predstavlja brz način da se dodele permisije većem broju korisnika. Svaka grupa može biti povezana sa jednom ili više uloga. Takođe može imati dodeljen set permisija, koje obezbeđuju pristup određenim resursima. Kada je korisnik dodat u grupu automatski nasleđuje sve uloge i permisije grupe, a kada je uklonjen iz nje gubi sve uloge i permisije koje je nasledio iz nje.

Aplikacija predstavlja aplikaciju koja interaguje sa Conductor serverom putem API-ja ili SDK-ja. Aplikaciji mogu biti dodeljene permisije, koje će obezbediti pristup određenim Conductor resursima. Svaka aplikacija može imati jedan ili više ključ/tajna parova, koji se koriste za dobijanje pristupa.

Tag predstavlja par ključ/vrednost koji može biti pridružen metapodacima resursa. Služi kao prečica za deljenje permisija brojnih resursa ili korisnika.

Uloga predstavlja set opštih podrazumevanih permisija resursima. Može biti dodeljena korisniku, grupi ili aplikaciji. Ako je dodeljeno više uloga biće dodeljene permisije za svaku od njih.

Pored permisija baziranih na ulogama moguće je dodeliti i granularne permisije grupama ili aplikacijama. Granularne permisije pružaju dodatni pristup povrh korisnikovih ili aplikacijskih permisija baziranih na ulozi.

Domen se koristi da se dodeli pristup svim zadacima u određenom domenu. Koristan je za masovno dodeljivanje prava radnik aplikaciji da izvrši sve zadatke, bez da se mora dodati svaki pojedinačni zadatak i da se definiše njegov domen.

2.6. Rutiranje zadataka

Za svaki konfigurisani tip zadatka Conductor server će održavati red i distribuiraće zadatke svim radnicima konektovanim na server. Postoji i opcija gde isti zadatak može biti rutiran različitom setu radnika na osnovu koncepta zvanog Task to Domain. Ova vrednost se prosleđuje toku kada je pokrenut, i ako je prisutna znači da se zadaci rutiraju drugačije od podrazumevanog načina.

2.7. Nadgledanje redova zadataka

Red zadataka sadrži zadatke koji čekaju da se izvrše. Nadgledanje ovih redova obezbeđuje optimalni performans i efikasnost u procesiranju zadataka, identifikovanje potencijalnih problema i održavanje pouzdanosti sistema.

2.8. Metrika i skaliranje radnika

Skaliranje i podešavanje performansi radnika zavisi od sledećih metrika: broja zadataka na čekanju, propusnosti pojedinačnog radnika i ukupnog broja pokrenutih radnika.

2.9. Rukovanje greškama

Conductor je napravljen da nudi najmanje jednu garanciju isporuke – sve poruke mogu da se čuvaju, trajne su i biće isporučene radnicima najmanje jednom. Ovakav model osigurava dve stvari: kada tok započne biće kompletiran ako su svi zadaci kompletirani i ako izvršavanje toka bude neuspešno zbog ponovnih pokušaja i sl., poruke će biti isporučene sledećem čvoru koji je živ i responsivan.

Timeout se može dogoditi ako nema radnika za zadati tip zadatka, radnik primi poruku ali prekine svoj rad pre nego što završi zadatak i zadatak nikada ne pređe u complete status ili je radnik završio procesiranje, ali ne može da komunicira sa Conductor serverom usled greške u mreži, ili srušenog Conductor servera.

3. APACHE KAFKA

Apache Kafka [4] je distribuirana platforma za striming podataka koja može da objavljuje, prima, čuva i procesira strimove u realnom vremenu. Dizajnirana je da rukuje velikim brojem real-time informacija i rutira ih mnogostrukim consumer-ima.

Osnovna jedinica unutar Kafke je poruka – niz bajtova. Poruke mogu imati opcione metapodatke – ključ. Ključ je takođe niz bajtova i koristi se kada se poruke upisuju u particije. Zbog bolje efikasnosti poruke se u Kafku upisuju kao batch – kolekcija poruka, gde se poruke kategorišu u teme, koje su dodatno razbijene na particije. Poruke se upisuju na kraj teme i čitaju sa početka. Ukoliko tema ima više particija ne može biti garantovano da će u celoj temi biti ispoštovan redosled kako su poruke stizale, samo u pojedinačnoj particiji. Particije su način na koji Kafka obezbeđuje skalabilnost, jer svaka particija

može biti na drugom serveru, što znači da jedna tema može biti horizontalno skalirana na više servera.

3.1. Producer

Producer-i [5] su aplikacije koje kreiraju poruke i šalju ih Kafka brokeru za dalje konzumiranje. Producer ne upisuje poruke u particije, već kreira zahteve za poruke i šalje ih vođi brokera. Kafka producer-i mogu biti sinhroni i asinhroni. Sinhroni producer-i nakon slanja poruke čekaju potvrdu od brokera, dok asinhroni nastavljaju sa daljim radom bez čekanja. Prednost sinhronih producer-a je pouzdanost, međutim čekanje na potvrdu može dovesti do zastoja u sistemu i ograničavanja broja poruka koje mogu biti poslate odjednom. Kod asinhronih producer-a ovi problemi su rešeni, ali su dosta komplikovaniji za implementiranje, naročito rukovanje greškama i ako nisu pažljivo implementirani može doći do gubitka podataka.

3.2. Consumer

Consumer-i [5] su aplikacije koje konzumiraju poruke. Consumer se pretplaćuje na jednu ili više tema i čita poruke u redosledu kojim su stigle. Consumer vodi računa koje poruke je već pročitao tako što vodi računa o offset-u poruka. Offset je metapodatak, integer brojač koji se stalno uvećava, na koji Kafka daje svaku poruku kako je objavljena. Čuvanjem offset-a poslednje konzumirane poruke za svaku particiju, u Zookeeper-u ili u samoj Kafki, consumer može biti zaustavljen i ponovo pokrenut bez gubljenja podataka. Consumer-i se nalaze unutar consumer grupe – jedan ili više consumer-a koji rade zajedno i konzumiraju temu. Grupa osigurava da je svaka particija konzumirana od samo jednog člana. Mapiranje consumer-a i particije se naziva vlasništvo particije od strane consumer-a. Na ovaj način consumer-i mogu biti horizontalno skalirani da konzumiraju teme sa velikim brojem poruka. Ukoliko jedan consumer ima grešku ostali članovi grupe će rebalansirati particije tako da preuzmu posao neuspešnog člana.

3.3. Brokeri i klasteri

Jedan Kafka server se naziva broker. Broker prima poruke od producer-a, dodeljuje im offset, i commit-uje poruke na disku. Takođe servisira consumer-e, odgovarajući na fetch zahteve porukama koje je prethodno commit-ovao. Kafka brokeri funkcionišu kao deo klastera. Unutar klastera, jedan broker služi kao kontroler, izabran automatski iz skupa živih članova klastera. Kontroler je zadužen za administrativne operacije, uključujući dodeljivanje particija brokerima i nadgledanje brokera u slučaju neuspeha. Particija je u vlasništvu samo jednog brokera unutar klastera, i taj broker se zove vođa particije. Međutim, particija može biti dodeljena više brokera, što će rezultovati repliciranjem particije.

Karakteristika Kafke je period zadržavanja. Brokeri su konfigurisani sa podrazumevanim periodom ili dok tema ne dostigne određenu veličinu. Pojedinačne teme mogu biti definisane sa sopstvenim periodom zadržavanja.

3.3. Zookeeper

Zookeeper je komponenta Kafke koja služi kao koordinator i služi za biranje kontrolera, čuvanje statusa brokera, čuvanje metapodataka tema, održavanje informacija o klijentskim normama i ACL tema.

4. POREĐENJE ORKES CONDUCTOR-A I APACHE KAFKE

Za poređenje orkestracije i koreografije uzet je primer onlajn prodavnice. Kreiran je Java projekat u kome su definisani sledeći mikroservisi: ordering servis, inventory servis, payment servis, notification servis i shipping servis.

4.1. Implementacija Conductor toka

Mikroservis ordering preuzima ulogu orkestratora i koordiniše komunikaciju između preostalih servisa. Tok je definisan u JSON formatu i dodat u Orkes server. U okviru ordering servisa definisan je REST endpoint koji pokreće tok korišćenjem WorkflowClient-a definisanog u zavisnosti io.orkes.conductor:orkes-conductor-client. Prilikom pokretanja toka zadaje se ime toka koji se pokreće, kao i imena i vrednosti ulaznih promenljivih.

Prilikom definisanja radnika potrebno ga je anotirati sa @WorkerTask. Anotacija od ulaznih parametara ima ime zadatka, broj niti koje se koriste za izvršavanje zadatka, kao i interval koliko često radnik šalje upite serveru.

4.2. Implementacija Kafke

Koristi se event-driven komunikacija. Kafka server (kafka i zookeeper) je podignut na portu 9092 u Doker kontejneru i korišćenjem Java biblioteke u mikrosrevisima se kreiraju KafkaTemplate i ConcurrentKafkaListenerContainerFactory (samo u servisima gde postoji consumer) na osnovu konfiguracije definisane u application.yml fajlu. U okviru ordering servisa definisan je endpoint koji pokreće slanje poruka – ovaj servis jedini nema consumer-a, budući da je servis od kojeg počinje slanje.

Prilikom definisanja consumer-a potrebno ga je anotirati sa @KafkaListener. Anotacija od ulaznih parametara ima identifikator consumer-a, temu koji osluškuje, identifikator grupe – u ovom slučaju servis u kojem se nalazi, kontejner fabrike – definisano Java kodom prilikom konfiguracije Kafke i rukovalac u slučaju greške.

4.3. Poređenje primena struktura podataka tipa red

Red kao strukturu podataka koriste i Conductor i Kafka. Kod Conductor-a zadaci koji čekaju da se izvrše se nalaze u redu, a kod Kafke se poruke smeštaju u redove.

Conductor koristi FIFO (First-in First-out) strukturu, realizovanu preko Redis liste kao skladišta podataka.

Kafka koristi log-baziranu distribuiranu append-only strukturu, što znači da su poruke redosledno upisane po particiji, ali redosled među različitim particijama nije zagarantovan i višestruki consumer-i mogu čitati poruku nezavisno (poruka ne nestaje nakon čitanja).

Tabela 1. *Poređenje Orkes Queues i Kafka redova*

	Orkes Queues	Kafka
Tip reda	FIFO lista	Commit log/append-only
Redosled	Globalni FIFO	Definisan particijom
Čuvanje poruka	Da	Da
Višestruku consumer-i	Ne	Da
Lokacija skladišta	U memoriji (Redis)	Disk
Performanse	Odlično za real-time zadatke	Odlično za bulk striming
Ponovno čitanje poruka	Ne	Da

4.4. Komparativna analiza performansi

Uzevši u obzir da performanse sistema mogu zavisiti od različitih parametara – poput konfiguracije, hardverskih resursa i memorije, teško ih je kvantifikovati.

Za potrebe ovog rada, projekat čita podatke o porudžbinama iz pet datoteka, sa različitim brojem porudžbina.

Tabela 2. *Rezultati izvršavanja*

Broj poruka	500	1000	2000	5000	10000
Conductor	10s	15s	26s	49s	95s
Kafka	0.41s	0.31s	0.43s	1.32s	7.24s

Razlika prilikom obrade malog broja podataka nije velika, ali može se videti da Kafka ima bolje performanse, i razlika u performansama se povećava sa brojem podataka.

Kako servisi za shipping i notification ne zavise jedan od drugog definisani su u paraleli. Ovaj paralelizam kod Conductora-a zahteva i dodatne FORK i JOIN zadatke, gde prilikom izvršavanja najviše vremena odlazi na JOIN zadatak. U verziji gde je Conductor tok definisan kao sekvencijalni (izvršavaju se shipping pa notification zadatak) dolazi do ubrzanja - u slučaju sa 10000 podataka vreme izvršavanja je palo sa 95s na 76s. Razlozi zašto je sekvencijalni tok ispao brži od paralelnog su da je paralelizam logički, a ne fizički i overhead orkestracije – svaki zadatak mora biti evidentiran u bazi, status zadatka se upisuje u skladište metapodataka, radnik ga mora preuzeti i vratiti rezultat, što znači da kod jednostavnih tokova overhead postaje veći nego korist paralelizacije.

Na primeru implementacije Kafke, vidi se da Kafka ne čeka da poruke i od notification i shipping servisa budu primljene da bi se smatralo da je proces uspešno završen. Za dobijanje informacija o tome uvodi se još jedan servis – delivery servis, koji čeka uspešnu obradu obe poruke i nakon što su obrađene, razmena poruka će biti uspešna. Sa dodatnim servisom dolazi do značajnog usporavanja obrade poruka korišćenjem Kafke, gde je za 10000 poruka sada potrebno 29.9s.

5. ZAKLJUČAK

Sam Conductor ne obrađuje podatke direktno, već deluje kao centralni koordinator koji ih delegira radnicima, koji zatim vraćaju rezultate po završetku. Ova arhitektura omogućava sistemu da ostane slabo spregnut, uz potpunu vidljivost i kontrolu nad dugotrajnim i složenim procesima. Za razliku od Kafke koja je optimizovana za brzo i pouzdano prosleđivanje poruka između producer-a i consumer-a, Conductor je usmeren na orkestraciju tokova i upravljanje stanjem kroz kompletan životni ciklus procesa.

U situacijama gde je potrebna jednostavna razmena poruka ili linearno procesiranje, Kafka je adekvatniji izbor. Međutim, u slučajevima sa kompleksnijom logikom, gde je neophodno izvršavanje u paraleli, ponovni pokušaji, rukovanje greškama, uslovno grananje ili su zadaci vezani za specifične korisnike Conductor pruža prednost u pogledu deklarativnog modeliranja, bolje preglednosti, naprednog upravljanja greškama i načina upravljanjem zavisnostima između zadataka – redosled izvršavanja zadataka zavisi samo od definicije toka, a ne od implementacije u mikroservisima, što omogućava dinamično ažuriranje poslovne logike, bez modifikacije pojedinačnih komponenti.

Sami Conductor i Kafka nisu međusobno isključivi – Kafka može funkcionisati kao centralna infrastruktura za rukovanje događajima, dok Conductor može da služi kao „mozak“ procesa, koji orkestrira kako će se ti događaji obrađivati, transformisati i povezivati u smislene rezultate. Ovakav hibridni pristup omogućava korišćenje prednosti obe platforme – Kafke za stabilno prosleđivanje poruka, a Conductor za strukturiranu orkestraciju tokova poslovanja, čime se postižu sistemi koji su istovremeno skalabilni i inteligentni.

6. LITERATURA

[1] N. Alshuqayran, N. Ali and R. Evans, “A Systematic Mapping Study in Microservice Architecture”, Proc. of the 9th International Conference on Service Oriented Computing and Applications, IEEE, 2016.

[2] R. H. Campbell and P. G. Richards, “SAGA: A system to automate the management of software roduction”, Proceedings of the May 4-7 1981. National Computer Conference on AFIPS, pp. 231-234, 1981.

[3] https://orkes.io/content (pristupljeno u septembru 2024.)

[4] https://en.wikipedia.org/wiki/Apache_Kafka (pristupljeno u novembru 2024.)

[5] N. Gard, “Apache Kafka”, PACKT Publishing, 2013.

Kratka biografija:

Jelena Ninković je rođena u Šapcu 1993. godine. Upisala je fakultet 2013. godine, smer Elektrotehnika i računarstvo. Diplomirala je 2019. godine sa temom „Implementacija korisničkog interfejsa podsistema bankarskog poslovanja korišćenjem AngularJS razvojnog okvira“.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE19Popov

PLATFORMA ZA TRGOVINU ENERGIJOM SA MIKROSERVISNOM ARHITEKTUROM I AI ANALIZOM TRŽIŠTA

ENERGY TRADING PLATFORM WITH A MICROSERVICES ARCHITECTURE AND AI MARKET ANALYSIS

Aleksa Popov, *Fakultet tehničkih nauka, Novi Sad*

Oblast – PRIMENJENO SOFTVERSKO INŽENJERSTVO

Kratak sadržaj – *U radu je prikazana realizacija aplikacije za automatizovano trgovanje električnom energijom, koja koristi treniran AI model za predikciju cena. Rešenje je implementirano upotrebom mikroservisne arhitekture, a celokupna aplikacija je bazirana na Docker kontejnerima.*

Ključne reči: *Mikroservis, AI model, Docker, trgovina energijom*

Abstract – *The paper presents the implementation of an application for automated electricity trading, which uses a trained AI model for price prediction. The solution is implemented using a microservices architecture, and the entire application is based on Docker containers.*

Keywords: *Microservice, AI model, Docker, energy trading*

1. UVOD

Tržište električne energije se danas menja velikom brzinom. Ključni pokretač ovih promena je razvoj pametnih energetskih mreža (Smart Grid) koje omogućavaju dvosmernu komunikaciju između proizvođača i potrošača. U skladu sa ovakvim sistemom, korisnici više nisu samo pasivni potrošači, već mogu i sami da proizvode i skladište energiju, na primer, pomoću solarnih panela i kućnih baterija. Ova nova dinamika dovodi do čestih i naglih promena u cenama energije, čime dolazi do sve veće potrebe za automatizovanim sistemima koji će nam olakšati praćenje promena.

Cilj ovog rada je osmišljen da zadovolji potrebe nastale novom dinamikom trgovine, kreiranjem softverske platforme koja automatizuje proces trgovine energijom. Platforma koristi veštačku inteligenciju da predvidi kretanje cena i na osnovu tih prekdikcija donosi odluke o kupovini ili prodaji, olakšavajući korisnicima da ostvare uštedu ili profit. Za osnovu sistema izabrana je mikroservisna arhitektura, koja omogućava da se kompleksan problem razloži na manje delove. Na taj način, stvoren je skalabilan temelj za buduća unapređenja I dodavanje novih funkcionalnosti.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Bošković, doc.

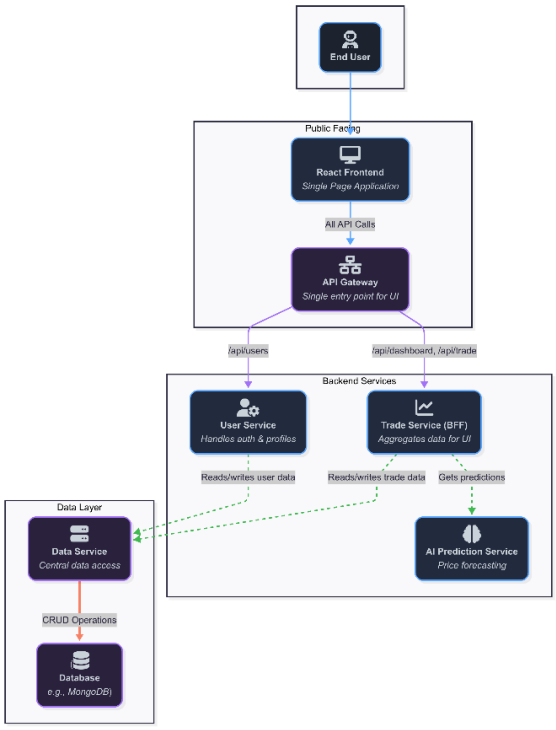
2. IZBOR ARHITEKTURE SISTEMA

Izbor prave arhitekture je ključan korak u izradi svakog softvera, koja će uticati na svaku dodatnu promenu na sistemu u budućnosti. Za ovaj projekat, postojale su dve glavne opcije pristupa: monolitnog i mikroservisnog.

Monolitna arhitektura podrazumeva da se sve komponente aplikacije nalaze unutar jedne, jedinstvene celine. Iako je ovakav pristup jednostavniji za početni razvoj, on sa sobom nosi veliki broj nedostataka kako sistem raste. Svaka pa i manja izmena zahteva ponovno testiranje i implementaciju celokupne aplikacije. Još jedna velika mana je da je ceo sistem vezan za jednu tehnologiju.

Zbog ovih nedostataka, za ovaj projekat je odabrana mikroservisna arhitektura [1]. Ona podrazumeva razbijanje sistema na skup manjih, nezavisnih servisa, gde je svaki servis odgovoran za jedan specifičan poslovni zadatak. Ovaj pristup je doneo par ključnih prednosti. Fleksibilnost tehnologije, koja je bilo od ključnog značaja za AI servis, nezavisnost i lakše održavanje što omogućava da se svaki servis razvija, testira i ažurira nezavisno i skalabilnost.

2.1. Način organizacije mikroservisa



Slika 1. *Arhitektura sistema sa prikazom komunikacije*

Unutar mikroservisne arhitekture, primenjeno je nekoliko ključnih obrazaca kako bi se sistem efikasno sagradio. On je podeljen na servise po njihovoj poslovnoj funkciji. Postoje četiri servisa koja izvršavaju glavne funkcije ovog sistema. Vizualan prikaz ove arhitekture, kao i način na koji servisi međusobno komuniciraju, dat je na Slici 1.

- User Service: upravlja korisničkim nalozima, procesima registracije i prijave, kao i izdavanjem sigurnosnih JWT tokena [4]
- Trade Service: sadrži ključnu poslovnu logiku, upravlja stanjem baterije i donosi odluke o trgovini.
- AI Service: funkcioniše kao specijalizovana komponenta koja na zahtev isporučuje predikcije cena.
- Data Service: centralna tačka za pristup bazi podataka, obezbeđujući potrebne podatke svim ostalim servisima.

"Baza podataka kao servis" (DaaS) [5]: Mikroservis-data simulira ulogu DaaS provajdera. Umesto da svaki servis direktno pristupa bazi podataka, svi zahtevi za čitanje ili pisanje podataka idu isključivo preko ovog servisa. Iako prestavlja odstupanje od "čistog" mikroservisnog pravila gde svaki servis ima svoju bazu, ovaj pristup je odabran kao savršeno rešenje za početnu fazu projekta. Postoje mnoge prednosti u korišćenju paterna Shared Database [6] za razvoj nove velike aplikacije koja uključuje mikroservise.

3. KORIŠĆENE TEHNOLOGIJE I ALATI

Za realizaciju ovog projekta korišćena je kombinacija modernih tehnologija, gde je svaka odabrana tako da na najbolji način odgovari na specifične zahteve.

Frontend – Korisnički interfejs je izgrađen pomoću React biblioteke, koja omogućava kreiranje dinamičkih I reaktivnih komponenti. Za upravljanje globalnim stanje aplikacije, korišćen je Redux.

Backend – Osnovu za User, Trade i Data servise čini Node.js, platforma izabrana zbog svoje efikasnosti u obradi mrežnih zahteva. Na njoj je korišćen Express.js framework za lako kreiranje REST API endpointa. Za AI servis odabran je Python zbog velike dostupnosti biblioteka, dok je Flask [3] poslužio kao web framework.

Baza podataka – Svi podaci se čuvaju unutar MongoDB, popularnoj noSQL bazi podataka. Za komunikaciju drugih servisa i baze korišćen je Mongoose, alat koji olakšava definisanje modela podataka i rad sa njima.

Veštačka inteligencija – Model za predikciju cena je implementiran pomoću Scikit-learn biblioteke.

Kontejnerizacija – Ceo sistem je orhanizovan pomoću Docker [2] platforme, gde je svaki mikroservis zapakovan u sopstveni izolovani kontejner. Za pokretanje i međusobno povezivanje svih kontejnera u lokalnom okruženju korišćen je Docker Compose, čime je postignuta laka prenosivost i konzistentnost sistema.

4. SPECIFIKACIJA SISTEMA

Ovo poglavlje detaljno opisuje funkcionalnosti sistema iz korisničkog i sistemskog ugla.

4.1. Korisničko iskustvo i funkcionalnost sistema

Korisnik po prvom ulasku na aplikaciju ima pristup početnoj strani, gde se nalazi cena energije kao i istorija cena, i takođe može da pristupi predikcijama i strategijama trgovine. Pre korišćenja glavne funcionalnosti, potrebno je prvo kreiranje profila. Nakon uspešne registracije, kao i prilikom svake sledece prijave, korisnik unosi svoje kredencijale. Ukoliko je autentifikacija uspešno prošla, korisniku se generiše JWT (JSON Web Token) [4]. Ovaj token služi kao digitalni ključ koji korisniku omogućava siguran pristup svim delovima platforme.

Glavna kontrolna tabla, koja je centralno mesto za sve aktivnosti, dizajnirana je da bude pregledna i intuitivna, a njen izgled je prikazan na slici 2. Na tabeli se ističe grafikon koji vizualizuje istorijke, ali i predviđene cene električne energije dobijene od AI servisa. Pored ovog grafa korisnik ima i opciju da pogleda predikcije za ceo sledeci dan na stranici Market.

Battery stranica korisniku omogućava da registruje svoje baterije i definise parametre za njih. Svaka baterija pored svojih vrednosti ima i strategiju trgovine kao i stanje u kom se nalazi. Pored pasivnog praćenja, korisnik ima mogućnost I da aktivno upravlja svojim resursima. Platforma nudi izbor predefinisanih strategija tgovanja koje korisnik može da dodeli svojim baterijama. Svaka strategija definiše različit pristup trgovini. I van strategija korisnik može da upravlja svojim baterijama manuelno.

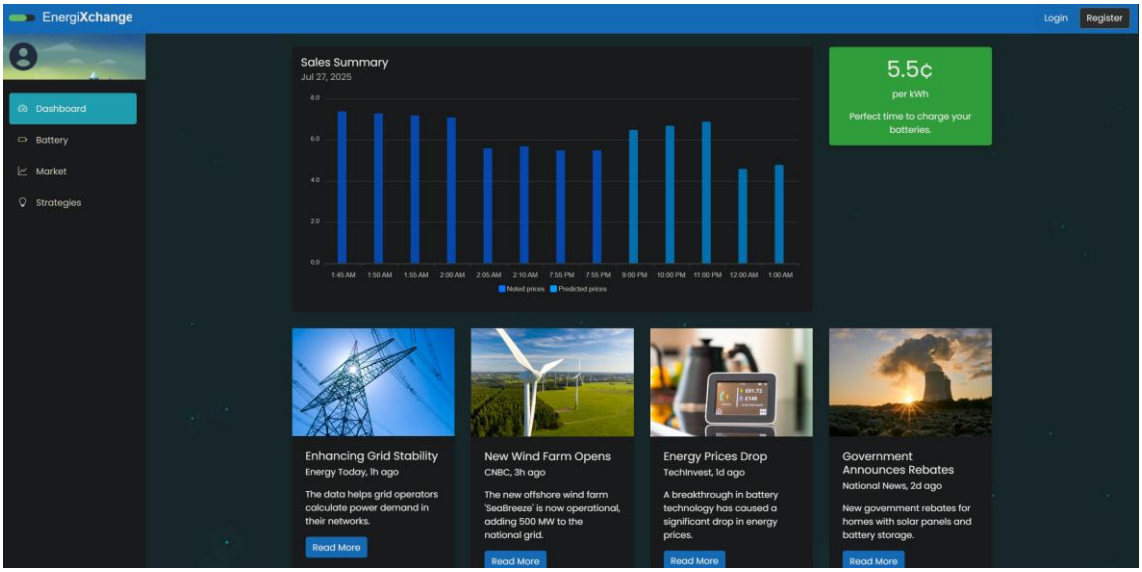
Ipak, ključna prednost platforme leži u njenom automatskom radu. Jednom kada korisnik odabere strategiju, sistem u pozadini preuzima upravljanje. Ovo je najvažniji sistemski proces, nevidljiv za korisnika, gde Trade Service neprestano analizira predikcije cena dobijene od servisa i u skladu sa aktivnom strategijom, on samostalno donosi odluke o najboljem trenutnom stanju baterije.

Sve spomenute automatske akcije se beleže kao transakcije i ažuriraju stanje korisnikovog virualnog novčanika. Svi ovi podaci su vidljivi na user stranici, i korisnik može uvek da doda sredstva na spomenut novcanik. Uplata je ovde obezbedjena Google reCaptcha-om v3 [8].

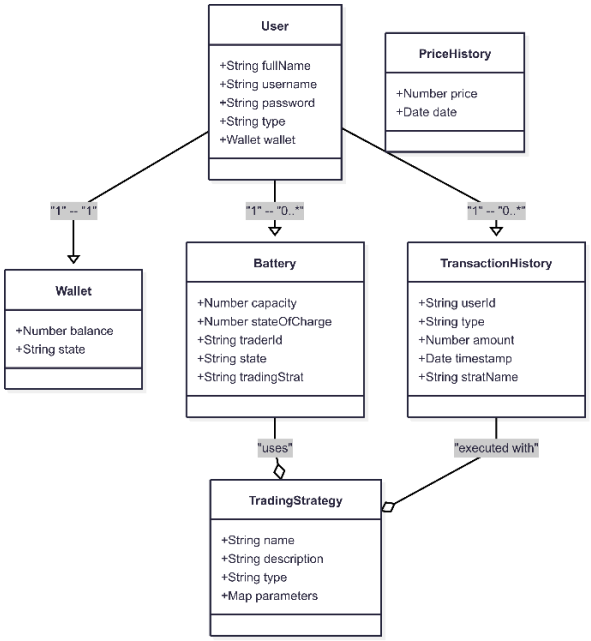
4.2. Dizajn model podataka

Osnovu sistema čine Mongoose modeli koji definišu ključne entitete I njihove međusobne veze. Statička struktura ovih modela, koja predstavlja temelj celog sistema, detaljno je prikazana na klasnom dijagramu na slici 3.

Kao što dijagram ilustruje, User model je centralni entitet koji sadrži osnovne informacije o korisniku I njegovom novčaniku. On je direktno povezan sa Battery modelom, omogućavajući da jedan korisnik poseduje više baterija, kao i sa TransactionHisotry modelom, koji čuva zapise o svim njegovim finansijkim aktivnostima. TradingStrategy model povezan je sa baterijom.



Slika 2. *Izgled glavne kontrolne table aplikacije*



Slika 3. *Klasni dijagram*

5. MODEL ZA PREDIKCIJU CENA

Za potrebe ovog projekta izabran je model Linearne Regresije [7] implementiran pomoću Scikit-learn biblioteke. Iako postoje kompleksniji modeli za vremenske serije, ovaj model je odabran iz nekoliko praktičnih razloga. On predstavlja odličnu polaznu osnovu I veoma je brz za treniranje. Takođe rezultati su lako razumljivi, što znači da se može jasno rezumeti kako svaka ulazna karakteristika utiče na krajnju predikciju. Za prototip sistema čiji je cilj dokazivanje koncepta, ove prednosti su bile presudne.

5.1. Proces treniranja

Proces rada modela se odvija u dve faze:

- **Faza treniranja**: U ovoj fazi, model se trenira na istorijskim podacima. Ulazni podaci su numeričke karakteristike, a ciljna vrednost koju model uči da predvidi je cena. Pozivom fit metode, model pronalazi matematičku formulu, odnosno linearnu vezu izmedju ulaznih karakteristika i cene. Jednom istreniran model, odnosno naučeni koeficijenti, čuvaju se u .pkl fajlu pomoću Joblib biblioteke.
- **Faza predikcije**: Kada Trade Service zatraži predikciju, učitava se sačuvani model iz .pkl fajla. Zatim se kreiraju buduće vremenske oznake I na njima se primenjuje isti proces inženjeringa karakteristika. Tako pripremljeni podaci se prosleđuju modelu, koji pomoću naučene formule izračunava i vraća predviđene cene za naredni dan.

6. ZAKLJUČAK

Na osnovu iznetih rezultata, može se zaključiti da primena mikroservisne arhitekture predstavlja efikasno rešenje za razvoj kompleksne platforme za trgovinu energijom. Pokazalo se da je ovakav pristup značajno fleksibilniji od monolitnog, naročito u domenu kombinovanja tehnologija. Korišćenje pragmatičnog obrazca kao što je Daas se pokazalo kao dobar kompromis koji je ubrzao početni razvoj kao i testiranje aplikacije. Potrebno je u buducnosti istražiti da li je ovakav model pouzdan i za globalnu upotrebu ili je “tradicionalni” pristup ipak bolji.

U daljem radu, primarni fokus bi se mogao staviti na unapređenje prediktivnog modela. Istraživanje upotrebe primene napredinijih modela, poput LSTM (Long Short-Term Memory) neuronskih mreža, koje su specijalizovane za analizu vremenskih serija. Ovo bi bilo od velikog značaja za preciznost same platforme, a samim tim i profitabilnosti iz automatske trgovine i pouzdanosti celog sistema.

Još jedan ocigledan korak jeste hostovanje platforme na

nekom od cloud servisa i detaljno testiranje performansi u realnom okruženju, što bi potvrdilo spremnost sistema za praktičnu upotrebu.

4. LITERATURA

[1] Sam Newma – Building Microservices: Designing Fine-Grained Systems
[2] Docker - https://www.techtarget.com/searchitoperations/definition/Docker
[3] Flask - https://flask.palletsprojects.com/en/stable/
[4] JWT Token - https://www.geeksforgeeks.org/web-tech/json-web-token-jwt/
[5] DaaS - https://www.mongodb.com/solutions/use-cases/data-as-a-service
[6] Pattern: Shared database - https://microservices.io/patterns/data/shared-database.html
[7] Linear Regression - https://www.geeksforgeeks.org/machine-learning/ml-linear-regression/
[8] reCAPTCHA - https://developers.google.com/recaptcha/docs/v3

Kratka biografija:

Aleksa Popov rođen je u Zrenjaninu 1998. god. Završio je osnovne akademske studije na Fakultetu tehničkih nauka 2021. godine, smer Primenjeno softversko inženjerstvo. Nakon toga upisao je iste godine master studije, smer Primenjeno softversko inženjerstvo.

Kontakt:popovaleksa123@gmail.com

Зборник радова Факултета техничких наука Нови Сад

УДК: 4.9
ДОИ: https://doi.org/10.24867/33BE20Kuzmanovic

КОМПАРАТИВНА АНАЛИЗА ОСНОВНИХ КАРАКТЕРИСТИКА И ПЕРФОРМАНСИ БРОКЕРА ПОРУКА *NATS*, *RABBITMQ* И *APACHE ROCKETMQ*

A COMPARATIVE ANALYSES OF THE MAIN CHARACTERISTICS AND PERFORMANCE OF NATS, RABBITMQ AND APACHE ROCKETMQ MESSAGE BROKERS

Огњен Кузмановић, *Факултет техничких наука, Нови Сад*

Област - ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У овом раду дато је поређење основних карактеристика и перформанси брокера порука NATS, RabbitMQ и Apache RocketMQ. Анализа перформанси урађена је кроз мерење кашњења порука и протока порука. За сваки брокер порука извршен је скуп тестова на платформи за рачунарство у облаку Azure. Конфигурација сваког теста се добила комбиновањем неколико различитих величина и броја порука. Зарад објективне компарације перформанси, осмишљен је и направљен посебан алат у програмског језику Python.*

Кључне речи: *Брокер порука, поређење перформанси, NATS, RabbitMQ, Apache RocketMQ.*

Abstract - *This paper compares the basic characteristics and performance of NATS, RabbitMQ and Apache RocketMQ message brokers. Performance is measured through message lattency and message throughput. Tests were conducted on the Azure cloud platform. The configuration of each test was obtained by combining several different message sizes and numbers. To ensure objective comparison, a custom Python-based tool was developed.*

Keywords: *Message broker, performance comparison, NATS, RabbitMQ, Apache RocketMQ.*

1. Увод

Временом, услед повећаног броја корисника интернета, расла је и количина података који су сервиси, који опслужују кориснике, морали да обраде. Стога, поједини дистрибуирани системи постојали су све комплекснији са све више повезаних компоненти које се прикључују и искључују из система са географски различитих локација, а све то ради опслуживања све више надолазећих корисника са различитих локација.

У таквим системима, уско повезани ентитети који међусобно комуницирају, у комбинацији са факторима попут непоузданости мреже и хетерегености различитих апликација могу да наштете трајности и поузданости система [1]. Из ових разлога, ексклузивно коришћење само синхроног модела комуникације „захтев/одговор“ не би пратило динамичну природу и потребе поменутих дистрибуираних система [2]. Како би се надоместиле мане синхроног модела комуникације, на значају је добио асинхрон модел комуникације који нуди већи ниво флексибилности, и то модел размене порука „објава/претплата“ са брокером порука као посредником у опслуживању порука јер не захтева да процеси који су учесници у размени порука буду континуирано активни.

Током претходног периода, настао је велики број различитих система за размену порука попут *Apache Kafka*, *ActiveMQ*, *Apache Pulsar*, *NSQ* итд. За предмет овог рада одлучено је да буду обрађене следеће имплементације: *NATS*, *RabbitMQ* и *Apache RocketMQ*. Поменути брокери порука истичу се тиме да су јавно доступни (енгл. *open source*), популарни су у инжењерско-академској заједници, нуде диверзитет различитих особина, а такође аутор није пронашао ниједан рад који је акценат ставио на баш њих.

Поред теоријске обраде поменутих брокера порука, циљ овог рада било је и мерење перформанси брокера порука на платформи за рачунарство у облаку *Azure*, као и крајње извођење закључака о томе у којим ситуацијама би требало користити који брокер порука.

2. *NATS*

Према [3], *NATS* представља инфраструктурну компоненту модерних дистрибуираних система која омогућава размену података између апликација и сервиса у виду порука. *NATS* услуге су омогућене од стране више *NATS* серверских процеса који су међусобно повезани и тако креирају *NATS* сервисну инфраструктуру. Кроз апликативни код дефинишу се клијенти који се повезују на *NATS* серверске процесе и њима објављују поруке или од њих примају поруке на које су претплаћени.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији је ментор био др Владимир Димитриески, ванредни професор.

Основни скуп функционалности које *NATS* нуди назива се *Core NATS*. На ову, базичну, *NATS* компоненту може се гледати као на једноставан систем за размену порука „објава/претплата“. Прималац порука ће поруке примати само ако су и он и генератор порука повезани на *NATS* у истом моменту. Основни скуп функционалности може се проширити кроз *JetStream* компоненту која нуди перзистентни дистрибуирани систем који омогућава олабављењију спрегу између генератора и конзумента порука, као и додатан скуп функционалности попут репликација порука, складишта кључ/вредност итд.

3. *RabbitMQ*

RabbitMQ је популаран брокер порука отвореног кода, написан у програмском језику *Erlang*, јавно доступан од 2007. године. Имплементира неколико различитх протокола међу којима су *AMQP 0-9-1* и *MQTT*.

Архитектура *RabbitMQ* брокера порука изгледа тако да се с једне стране налазе генератори порука, а са друге примаоци порука. Генератор поруке креира поруку којој придодаје кључ за трасирање (енгл. *routing key*) који шаље ка брокеру, а брокер прима поруку кроз компоненту за размену порука (енгл. *exchange*), односно размењивач порука. За размењивач порука потенцијално може бити везано много редова чекања уз помоћ различитих техника увезивања које зависе од конкретног типа размењивача поруке.

Табела 1. Табеларна компарација неперформатних категорија посматраних брокера порука

Особина	***NATS***	***RabbitMQ***	***RocketMQ***
Година настајања	2011	2007	2012
Језик	*Go*	*Erlang*	*Java*
Гаранција доставе	Најмање један/највише један/тачно један	Најмање један/ највише један	Најмање један
Гаранција редоследа	ФИФО редослед по генератору порука	Редослед у реду чекања, ФИФО редослед	Редослед у реду чекања, ФИФО редослед
Доступност	Висока	Висока	Висока
Трансакције	Не	Да	Да
Скалабилност	Висока	Слаба	Добра
Протоколи	*NATS* протокол	*HTTP*, *MQTT*, *AMQP*, *STOMP*	*MQTT*
Компаније [5] [6]	*Nutanix, MasterCard*	*Reddit*, *CircleCI*	*Alibaba*, *ByteDance*

Размењивач порука има задатак да дистрибуира копије порука ка редовима чекања у зависности његовог типа, од тога како су редови чекања увезани са њим, као и кључа за трасирање који је придодат порукама које се трасирају. С друге стране, примаоци порука претплаћени су на редове чекања и конзумирају поруке у њима.

4. *Apache RocketMQ*

RocketMQ је брокер порука креиран од стране компаније *Alibaba* како би решили проблеме перформанси које су имали са *ActiveMQ* брокером порука, а који су дошли до изражаја тек онда када је знатно порасла потражња за услугама ове компаније. Према [4], *RocketMQ* добио је на популарности услед његове једноставне архитектуре, великој лепези функционалности, као и изузетној скалабилности. Аутори овог брокера порука сматрају да је он постао индустријски стандард када је реч о порукама у којима се налазе финансијски подаци за које је неопходна изузетна доза сигурности преноса и обраде.

5. Поређење неперформантних особина

У табели 1, дато је поређење најзначајних неперформантних особина брокера порука које могу имати утицај на избор конкретног брокера порука за имплементацију у неки софтверски систем.

6. Тестирање перформанси брокера порука

За потребе компаративне анализе перформанси брокера порука упоређене су две перформантне категорије: кашњење порука (енгл. *latency*), као и проток порука (енгл. *throughput*). Како би се по ове две категорије добиле конкретне и упоредиве метрике, сваки брокер порука био је подвргнут серији тестова на пларформи *Azure*. Тестови су извршавани употребом алата посебно направљеног само за потребе овог рада.

6.1. Хардверско-софтверско окружење

За потребе тестирања перформанси брокера порука, коришћене су виртуелне машине *Standard D4ds v4* које нуди платформа *Azure* у склопу *Azure Virtual Machines* сервиса. Овај тип виртуелних машина нуди следећу спецификацију: 4 vCPUs / 16GiB RAM / 30 GiB HDD / 6400 IOPS / Ubuntu 22.04. Као физичка локација машина, изабран је регион северне Европе, у зони 3.

Зарад олакшаног покретања изолованих окружења у којима су се брокери порука извршавали, на машину су били инсталирани алати *Docker v27.3.1* и *Docker Compose v2.29.6*. *Docker* контејнери нису били условљени ограничењима ресурса.

Тестирање се вршило на једној виртуелној машини како би се обезбедило добијање фер резултата на које не утиче кашњење мреже.

6.2. Параметри тестирања

Као предмети тестирања, у обзир су се узели најзначајније перформантне метрике у свету брокера порука [7], а то су кашњење порука, као и проток порука. За сваку од поменутих категорија тестирања, тестирање се вршило за три могуће величине поруке како би се установило да ли и како величина порука утиче на перформансе брокера порука. Величине порука:

1. Мала порука – 1KB
2. Средња порука – 100KB
3. Велика порука – 1MB

Свака порука је садржала низ унапред насумично генерисаних карактера (*ASCII* карактери, цифре, као и знакови интерпункције).

Као додатан параметар тестирања употребљен је број порука како би се установило да ли постоје промене у перфомансама за различите редове величина броја порука. За сваку величину порука, три различите количине порука су биле тестиране:

1. Мали број порука - 1.000
2. Средњи број порука - 10.000
3. Велики број порука - 100.000

За сваку комбинацију поменутих параметера тестирања, извршено је најмање три теста. У случају да се међу извршена три теста нашао резултат који значајно одступа од осталих резултата, извршено је још три теста. Као крајњи резултат узимала се просечна вредност три најбоља резултата.

6.3. Спецификација брокера порука

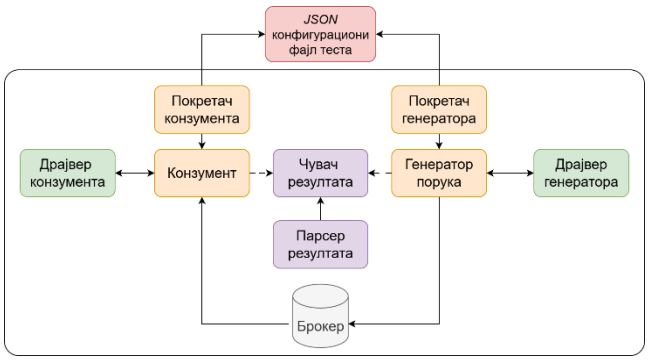
У наставку су дате тачне верзије коришћених брокера порука:

- *NATS* v2.10.14
- *RabbitMQ* v3.13.2
- *Apache RocketMQ* v5.2.0

Код свих сценарија тестирања генератор порука, брокер и прималац порука су били заједно на једној машини. Модел комуникације који је био коришћен је „објава/претплата“.

6.4. Начин тестирања и бележења резултата

За потребе свих описаних тестирања и зарад максимално објективних резултата, креиран је посебан алат у програмском језику *Python*. Први корак у раду са алатом јесте конфигурисање параметара теста кроз *JSON* конфигурациони фајл. Кроз овај фајл, корисник има опцију да подеси жељени брокер порука, величину и број порука. Корисник задаје команду за покретање конзумента порука који се повезује са брокером порука и креће да ослушкује поруке. У том тренутку, конзумент у терминалу исписује поруку да је спреман и корисник може приступити покретању генератора порука. На основу параметара из конфигурационог фајла који се односе на тип брокера порука, број и величину порука, генератор порука се инстацира и креће да генерише поруке. Конзумент порука их прима и обрађује. Оба процеса задужена су за вођење бриге о прикупљању метрика током извршавања теста, а потом и њихово чување. Након извршења теста, потребно је покренути парсер добијених метрика ради извлачења информација везаних за мерене категорије. Визуелни приказ архитектуре описаног алата дат је на слици 1. Код је доступан на платформи *GitHub* и може му се приступити путем [8].



Слика 1. Визуелни приказ архитектуре алата за извршавање тестова

7. Резултати

У наставку су дати резултати брокера порука у контексту кашњења и протока порука.

7.1. Кашњење

У табели 2, приказани су брокери порука сортирани по резултатима које су остварили у сваком тесту.

Табела 2. Табеларни приказ резултата кашњења порука брокера порука

	Мала порука			Средња порука			Велика порука		
Број порука	1к	10к	100к	1к	10к	100к	1к	10к	100к
RabbitMQ	1	1	1	2	3	3	3	3	2
NATS	3	3	2	3	2	2	2	2	1
RocketMQ	2	2	3	1	1	1	1	1	3

Оно што се може закључити евалуацијом резултата мерења кашњења порука јесте, да је целокупно гледано, *Apache RocketMQ* показао најбоље резултате са убедљиво најмањим кашњењем код средње и велике величине величине порука за све количине порука (уз недостатак резултата са велику величину порука и обим од 100.000 порука), док је за мале величине порука показао минимално лошије резултате од најбољег *RabbitMQ* брокера порука. Разлог за веома ниско кашњења се првенствено огледа у томе што су аутори *Apache RocketMQ* брокера уложили велике напоре да направе што више оптимизација у извршавању процеса. Према [9] [10], знатна унапређења су направљена на пољу оптимизације извршавање процеса унутар *Java* виртуелне машине, као и на пољу смањења кашњења у закључавању ресурса (енгл. *lock latency*) узевши у обзир да је закључавање ресурса једна од основних потреба у вишенитним окружењима. *RabbitMQ* је показао најбоље резултате код мале величине порука, док је код средње и велике величине порука показао веома лоше резултате. Разлог за његове лошије резултате може се тражити у томе да је *RabbitMQ* фокусиран на процесирање порука у групама (енгл. *batches*) како би се побољшао проток на уштрб кашњења. *NATS* брокер порука, иако није показао најбоље резултате ни у једном тесту, био је конзистентан у резултатима.

7.2. Проток порука

У табели 3, приказани су брокери порука сортирани по резултатима које су остварили у сваком тесту мерења протока слања и конзумирања порука. Резултати су код оба предмета мерења били идентични.

Табела 3. Табеларни приказ резултата протока слања и конзумирања порука

	Мала порука			Средња порука			Велика порука		
Број порука	1к	10к	100к	1к	10к	100к	1к	10к	100к
RabbitMQ	2	2	2	1	1	1	1	1	1
NATS	1	1	1	2	2	2	2	2	2
RocketMQ	3	3	3	3	3	3	3	3	3

Евалуацијом резултата мерења протока порука се може закључити да је *Apache RocketMQ* показао убедљиво најгоре резултате у оба посматрана случаја протока порука. Објашњење за овакво понашање би се могло пронаћи у томе што *RocketMQ* чува сваку поруку на диск, док то није предефинисано понашање код остала два брокера порука. *NATS* је приказао изузетно добре резулате са врло високим протоком порука код мале величине порука. Са друге стране, *RabbitMQ* приказао је конзистентне и најбоље резултате код свих мерења за средњу и велику величину порука што је и логично узевши у обзир његову усмереност на обраду у групама.

8. Закључак

У овом раду, извршено је поређење *NATS*, *RabbitMQ* и *Apache RocketMQ* брокера порука кроз њихове перформантне и неперформантне категорије. У оквиру перформатних категорија, акценат је стављен на кашњење и проток порука, као две најзначајније категорије код брокера порука. Како би се тестирало што више варијација реалних сценарија коришћења самих брокера, сваки брокер порука тестиран је кроз два параметра тестирања – величина и број порука. Зарад добијања што објективнијих резултати, креиран је посебан алат у програмском језику *Python* који омогућава инстанцирање примаоца и генератора порука на основу конфигурационих параметара теста дефинисаних кроз *JSON* конфигурациони фајл, бележење метрика, њихово чување, као и парсирању у смислене резултате који се могу записати и тумачити. Сви чиниоци алата били су покренути на виртуелној машини *Azure* платформе на којој су тестови и извршавани. Оно што се може закључити из крајње анализе резултата мерења јесте да ниједан од посматраних брокера порука није идеалан за све сценарије употребе. Приликом одабира коришћења једног од посматраних брокера порука, препорука аутора јесте да се првенствено сагледа перформантна категорија која је битнија за систем који се развија, као и очекивана величина порука. На основу ова два параметра може се направити следећа матрица одлука:

1. Битност – минимално кашњење поруке
    a. Мала величина порука – *RabbitMQ*
    b. Средња и велика величина порука – *Apache RocketMQ*
2. Битност – максималан проток порука
    a. Мала величина порука – *NATS*
    b. Средња и велика величина порука – *RabbitMQ*

У овом раду, акценат је стављен на извршавање на једној машини како би се искључио утицај кашњења мреже на перформансе. Стога би будући правац развоја могао бити мерење перформанси брокера порука у дистрибуираном окружењу.

9. Литература

[1] L. Magnoni, "Modern Messaging For Distributed Systems," 2015.

[2] K. S. E. Philippe Dobbelaere, "Kafka versus RabbitMQ," 2017.

[3] "Oficijalna veb stranica NATS dokumentacije," [Online]. Available: https://docs.nats.io/nats-concepts/overview.

[4] "Zvanična dokumentacija Apache RocketMQ brokera poruka," [Online]. Available: https://rocketmq.apache.org/docs/. [Accessed 03 2024].

[5] S. Raje, "Performance Comparison of Message Queue Methods," 2019.

[6] "HG Insights," [Online]. Available: https://discovery.hgdata.com/. [Accessed 05 2024].

[7] "Benchmarking Apache Pulsar, Kafka, and RabbitMQ," 21 08 2020. [Online]. Available: https://www.confluent.io/blog/kafka-fastest-messaging-system/. [Accessed 05 2024].

[8] O. Kuzmanovic, "Programski kod korišćen za izvšavanje testova".

[9] Y. Z. A. G. Y. Guo Fu, "A Fair Comparison of Message Queuing Systems," *IEEE Access,* 2020.

[10] A. Shi, "DZone," [Online]. Available: https://dzone.com/articles/apache-rocketmq-how-did-we-lowered-latency. [Accessed 10 2024].

Кратка биографија:

Огњен Кузмановић рођен је 1998. године у Новом Саду. Основну школу „Светозар Марковић Тоза” завршио је 2013. године као носилац дипломе „Вук Караџић”. Исте године уписује природно-математички смер у гимназији „Јован Јовановић Змај” коју завршава 2017. године. Потом, уписује Факултет техничких наука, одсек Рачунарство и аутоматика. Све предвиђене испите положио је са просечном оценом 9,68.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE21Kasikovic

KOMPARATIVNA ANALIZA AGENTSKIH *RAG* PRISTUPA U KONTEKSTU IZGRADNJE KONVERZACIONOG ASISTENTA

COMPARATIVE ANALYSIS OF AGENTIC *RAG* APPROACHES IN THE CONTEXT OF BUILDING A CONVERSATIONAL ASSISTANT

Ivana Kašiković, Aleksandar Kovačević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – SOFTVERSKO INŽENJERSTVO

Kratak sadržaj – *U okviru rada je predstavljeno istraživanje i implementacija dvije verzije konverzacionog asistenta. Fokus je na konceptu Retrieval Augmented Generation (RAG) koji pored „ugrađenog“ koristi i eksterno znanje za generisanje odgovora i interakciju sa korisnikom. Prva implementacija koristi osnovnu kombinaciju dobavljanja znanja i generisanja odgovora, a zatim je razvijen složeniji pristup sa agentima i specijalizovanim alatima za obradu upita. Cilj je detaljno opisati proces izrade sistema, uporediti arhitekture po kompleksnosti, proširivosti, performansama i relevantnosti odgovora. Evaluacija je prvo rađena ručno, a kasnije automatizovana pomoću Ragas okvira. Na kraju su istaknute prednosti i mane oba pristupa, primjeri njihove primjene te prijedlozi za unapređenje i dalja istraživanja.*

Ključne reči: *RAG, Vanilla RAG, agentski RAG, agenti, alati, konverzacioni asistent*

Abstract – *As part of this paper, we present research and the implementation of two versions of a conversational assistant. The focus was on exploring Retrieval-Augmented Generation (RAG), which uses external knowledge in addition to built-in knowledge to generate responses and interact with users. The implementation started with a simple version that combines retrieval and generation. Afterwards, a more advanced approach was developed using agents and specialized tools for query processing. The goal of the paper is to provide a detailed description of the system development process, compare the architectures in terms of complexity, scalability, performance, and answer relevance. Initially, system evaluation was performed manually, while later performance analysis was automated using the Ragas framework. Finally, the advantages and disadvantages of both approaches are highlighted, along with scenarios where each would be appropriate. The paper concludes with suggestions for improvements and directions for future research.*

Keywords: *RAG, Vanilla RAG, agentic RAG, agents, tools, chatbot*

NAPOMENA: Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Kovačević, red. prof.

1. UVOD

Arhitektura sistema konverzacionih asistenata, organizacija komponenti, veliki jezički model i još mnogo drugih stvari značajno utiču na ponašanje I rezultate pomenutog sistema. LLM—ovi [1] iako veoma moćni alati imaju ograničeno znanje, tj. znanje na kome su se obučavali što je dosta umanjivalo vrijednost konverzacionih asistenata koji su se bazirali isključivo na njima. Kao rješenje za taj problem uveden je RAG [2] koncept, pomoću kog se stvara mogućnost da se prilikom generisanja odgovora uključi i neko eksterno znanje koje do tada nije bilo poznato velikom jezičkom modelu.

Brzim razvojem ove oblasti stvorene su različite implementacije ovih sistema. Stoga, cilj ovog rada je da izvrši komparativnu analiza izmedju najjednostavnije verzije i kompleksnijeg pristupa implementiranog uz pomoć agenta i specializovanih alata, pri čemu smo prvo dali detaljan opis arhitekture oba rješenja. Kao rezultat imaćemo realnu sliku o tome koliko je implementacija kog sistema kompleksna, koliko pouzdane i relevantne odgovore možemo očekivati za tu količinu utrošenog vremena i na taj način ćemo moći procijeniti koja vrsta arhitekture i implementacije bi bila adekvatna za koji slučaj upotrebe.

U drugom poglavlju biće opisane teorijske osnove RAG sistema kao koncepta, ali i osnove jednostavnog i agentskog RAG-a. Naredno poglavlje baviće se detaljnim opisivanjem procesa implementacije i koraka koji su sprovedeni kako bismo došli do krajnjih verzija sistema. Nakon toga slijedi poglavlje gdje smo opisali proces evaluacije. U petom poglavlju diskutovali smo o rezultatima sistema i data su potencijalna objašnjenja za određene metrike i neka ponašanja. Posljednje poglavlje donosi zaključak sa sažetkom istraživanja.

2. TEORIJSKE OSNOVE

U ovom poglavlju biće date teorijske osnove RAG koncepta, kao i teorijske osnove oba pristupa za implementaciju koji će kasnije biti i realizovana.

2.1. Šta je RAG?

Retrieval-Augmented Generation (RAG) je arhitektonski obrazac koji kombinuje dvije bitne komponente, a то sy: **pretraga (retrieval)** relevantnih informacija iz izvora i **generacija (generation)** odgovora pomoću velikog jezičkog modela. Uvođenjem ovog koncepta riješili smo problem stagnirajućeg znanja, niske

objašnjivosti i otvorili smo mogućnost generisanja odgovora i na osnovu domenski specifičnog znanja.

2.2 Eksterno znanje i njegova integracija u RAG sistem

Eksterno znanje predstavlja osnovni benefit RAG koncepta. Može biti bilo koji struktuirani ili nestruktuirani tekst, a sama procedura indeksiranja je ista i svodi se na pretprocesiranje teksta i podjelu na manje segmente, zatim generisanje embedding reprezentacija od tih elemenata i čuvanje u odabranu vektorsku bazu. Nakon toga, proces pretrage zasniva se na konvertovanju korisničkog upita u embedding reprezentaciju i vrši se pretraga po sličnost gdje su semantički slični vektori nalaze blizu u prostoru [3].

2.3 Mehanizmi promptovanja

Prompt [4] je jedna od ključnih stavki koje utiču na kvalitet generisanog odgovora. To predstavlja niz instrukcija koje se prosljeđuju velikom jezičkom modelu tokom inferencije. Za što bolje rezultate dobra je praksa ispoštovati karakteristike efektivnog prompta, a to su da bude jasan i precizan, da u okviru prompta bude uključen relevantan kontekst, kao i da instukcije budu neutralne tj. da ne sugerišu neki odgovor i da ne nameću određeno mišljenje i stav.

Sa druge strane, postoje i različite tehnike i strategije za unapređenje prompta posebno u slučaju RAG-a od kojih možemo izdvojiti instruction based promptovanje, few shot promptovanje itd.

2.4 Vanilla RAG

Vanilla RAG predstavlja osnovni i najjednostavniji vid RAG koncepta. Ovu implementaciju odlikuje linearan tok obrade upita, prvo se pokreće proces dobavljanja relevantnog konteksta, a zatim se na osnovu toga i generiše odgovor. U nastavku ćemo istaći osnovne osobine ovog pristupa.

2.4.1 Ključne osobine

Karakterisične osobine značajne za ovaj pristup su jednostavnost implementacije i statička logika. To znači da se ovakvi sistemi poprilično brzo stavljaju u upotrebu i da se svi upiti obrađuju na isti način. Kao posljedica svega toga možemo zaključiti da bi debagovanje ovakvog sistema bilo veoma lako.

2.4.2 Nedostaci

Iako je Vanilla RAG veoma jednostavan za implementaciju, ovo rješenje imamo određenih mana, pogotovo kada imamo kompleksnije upite. Glavni problem je što nema mogućnost adaptacije logike na osnovu upita, tako da ukoliko postavimo neki složeni upit vrlo vjerovatno nećemo dobiti adekvatan i složen odgovor.

2.4.3 Praktična primjena

Kao posljedica pomenutih osobina možemo reći da bi ova implementacija najbolje funkcionisala u manjim okruženjima gdje upiti nisu toliko česti i kompleksni, a baza znanja je statična i nije podložna čestim promjenama.

2.5 Agentski RAG

Agentski RAG [5] predstavlja napredni oblik RAG arhitekture u kojoj ključnu ulogu ima inteligentni agent, odnosno entitet sposoban za donošenje odluka, upravljanje kontekstom i strategijsko izvršavanje zadataka u više koraka uz pomoć alata.

2.5.1 Agenti

Agenti [6] predstavljaju samostalne jedinice koje imaju sposobnost da rezonuju, donose odluke i izaberu adekvatne alate za izvršavanje određenih akcija. Postoje različite vrste agenata u zavisnosti od toga koji je njihov cilj. U okviru ovog rada koristiće se ReAct agent koji može da rezonuje i na osnovu međurezultata može da mijenja strategiju i prilagođava je.

2.5.2 Alati

Alati [7] su pomoćne strukture koje agent koristi tokom procesa odlučivanja. U suštini, to je interfejs za funkcije koje mogu da dobavljaju određeno znanje ili izvršavaju neku akciju adekvatnu za obradu upita. Bitna stavka prilikom specificiranja nekog alata je opis, na osnovu koga inteligenti koordinator, agent zna šta je uloga tog alata i u zavisnosti od toga ga pozove ili ne.

2.5.3 Ključne osobine

Ključne osobine karakteristične za agentski RAG su modularnost i fleksibilnost. U nastavku će biti prikazano da se cijela arhitektura ove implementacije sastoji od komponenti, tako da je veoma praktično proširiti neku funkcionalnost, zamijeniti neku komponentu. Pored toga, ima mogućnost orkestracije složenih zadataka i generisanje adekvatnog odgovora i za kompleksnije zadatke.

2.5.4 Nedostaci

S obzirom da raste kompleksnost sistema, to sa sobom povlači neke mane kao što je složenost implementacije. Za kreiranje ovakvog sistema potrebno je više vremena i javlja se povećana latencija pošto se izvršava veći broj koraka. Povezano sa tim, javlja se i problem većih troškova izvršavanja.

2.5.5 Praktična primjena

Upotreba agentskog RAG-a je opravdana u sistemima koji zahtjevaju složenije radnje i laku proširivost sistema dodatnim alatima. Npr. To bi bio slučaj u velikim kompanijama koje zahtjevaju pretragu dokumentacije, generisanje izvještaja slanje mejlova itd.

3. METODOLOGIJA

U ovom poglavlju opisane su metode i koraci koji su primijenjeni tokom rada na projektu, čiji je cilj bio istraživanje i implementacija RAG sistema. Projekat je obuhvatio specifikaciju dizajna i implementaciju obje vrste RAG sistema. U nastavku slijedi detaljniji opis.

3.1 Specifikacija zahtjeva

Projekat je realizovan korištenjem programskog jezika Typescript i Nodej.js radnog okvira, pri čemu je serverski dio organizovan upotrebom modularne monolitne arhitekture. Korisnički interfejs je razvijen u React-u sa ciljem da omogući jednostavnu interakciju sa sistemom.

Aplikacija je odrađena za imaginarnu kompaniju koja bi imala mogućnost da pretražuje kompanijsku dokumentaciju u jednostavnijoj verziji sistema, a u agentskoj verziji ta implementacija je proširena i pristupom internetu i mogućnošću riješavanja matematičkih izraza

preko aplikacije. Prvo je dat opis pripreme podataka za testiranje aplikacije, a onda ćemo opisati i konkretno implementaciju.

3.2 Indeksiranje dokumenata

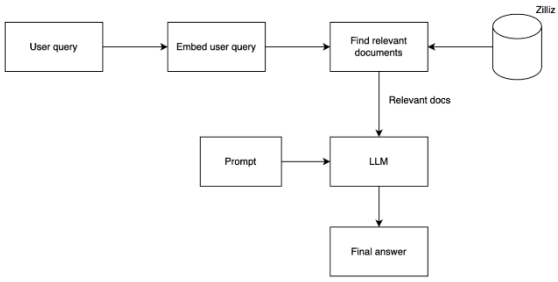
Prvi koraci u razvoju sistema su indeksiranje dokumenata i priprema eksternog znanja koje će se koristiti za testiranje ovih sistema. Za čuvanje generalnog znanja odlučili smo se za vektorsku bazu Zilliz [8]. Ovaj proces započinje raščlanjivanjem dokumenata na manje segmente, u našem slučaju vršena je semantička podjela teksta po pasusima.

Sljedeći korak je kreiranje vektorskih reprezentacija od tih pasusa upotrebom modela za embedovanje. U ovom slučaju koristili smo **OpenAIEmbeddings**, koji u pozadini podrazumjevano koristi napredni **model text-embedding-3-small**.

Posljednji korak pri indeksiranju dokumenata je čuvanje u vektorsku bazu. Odabrana vektorska baza nudi više načina ažuriranja baze, a u ovom radu iskorišten je inkrementalni način ažuriranja baze, pri čemu se tokom ažuriranja unose samo novi ili izmijenjeni segmenti, dok se zastarjeli automatski uklanjaju.

3.3 Implementacija Vanilla RAG-a

Vanilla RAG arhitektura implementirana je kao prva verzija konverzacionog asistenta i kombinuje proces za dobavljanje relevantnih dokumenata i proces generisanja odgovora. Proces dobavljanja relevantnih dokumenata oslanja se isključivo na veliki jezički model i nema mogućnost rezonovanja. Za implementaciju korišten je **Langchain** [9] radni okvir i **ChatGPT Turbo 3.5**. Slika čitave arhitekture prikazana je na slici 1.

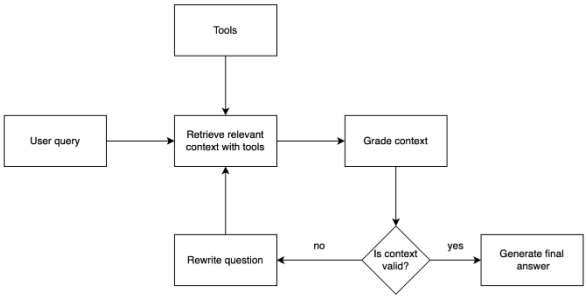


Slika 1. *Arhitektura Vanilla RAG-a*

3.4 Implementacija agentskog RAG-a

Agentski RAG predstavlja napredniju verziju RAG koncepta, u kom se umjesto velikog jezičkog modela kod jednostavne verzije koristi čitav agentski sistem sa različitim alatima, Uz pomoć agenta ovaj pristup dobija mogućnost rezonovanja I adaptacije logike za rješavanje kompleksnih korisničkih zahtjeva.

Za realizaciju ovog sistema korišten je **LangGraph** [10] gdje je svaki čvor predstavljao neki korak u obradi zahtjeva. Dok smo za definisanje alata koristili Langchain interfejs. Detaljniji prikaz arhitekture prikazan je na slici 2.



Slika 2 *Arhitektura agentskog RAG-A*

Na osnovu priloženog možemo zaključiti da prvi korak je dobavljanje relevantnog znanja pomoću agenta i alata. U ovom slučaju korišten je reaktivni agent koji planira dalji korak u skladu sa trenutnim stanjem.

Što se tiče alata za demonstraciju rada kreirali smo tri alata, a to su:

- Alat za dobavljanje znanja o kompaniji
- Alat za internet pretragu
- Alat za rješavanje matematičkih izraza

Zatim slijedi korak provjere da li je kontekst relevantan, ukoliko jeste ide se na kraj i generiše se odgovor, a u suprotnom pitanje se reformuliše i proces se ponavlja.

4. EVALUACIJA

Pod procesom evaluacije podrazumjeva se da se „izmjeri“ koliko tačne i relevantne odgovore sistem daje. Pošto se generalno proces sastoji od dvije faze dobavljanja znanja i generisanja odgovora, dobra praksa je evaluirati te dvije faze odvojeno.

U početku evaluacija je rađena ručno i empirijski na osnovu testnog skupa pitanja, a naknadno je taj proces automatizovan upotrebom Ragas radnog okvira.

5. REZULTATI

U ovom poglavlju biće dato finalno poređenje ove dvije implementacije na osnovu tehničkih metrika koje definišu tačnost odgovora, ali i poređenje po sekundarnim osobinama bitnim za razvoj ovih rješenja.

U prilogu je data tabela sa vrijednostima tehničkih metrika dobijenih preko Ragas [11] radnog okvira.

Tabela 1. *Evaluirane vrijednosti*

	Vanilla RAG	Agentski RAG
Relevantnost konteksta	0.6987	0.8903
Pridržavanje konteksta	0.2951	0.2698

Povraćaj konteksta	0.4773	0.4833
Relevantnost odgovora	0.7818	0.8196
Tačnost odgovora	0.6192	0.8684
Semantička sličnost	0.8434	0.9020

Na osnovu tabele, agentski pristup ima dosta bolje rezultate u relevantnosti konteksta, razlog za to su složeni upiti gdje drugi pristup dosta bolje rezonuje i pronalazi adekvatna dokumenta. Takođe, agentski pristup se neznatno bolje pokazao I u metrici povraćaj konteksta, do toga je došlo, jer agenti malo bolje rade u slučaju kadaa se kontekst vraća iz više dokumenata. Sa druge strane, posmatrajući pridržavanje konteksta jednostavniji model neočekivano daje bolji rezultat. Razlog za to je što agentski pristup daje dosta informacija da što bolje objasni koncept, ali u ovom slučaju to smanjuje vrijednost ove metrike.

Sa druge strane, što se tiče metrika vezanih za generisanje odgovora agentski pristup se dosta bolje pokazao kod tačnosti odgovora, jer ovo rješenje detaljnije vraća odgovor I detalje koji naizgled ne djeluju bitni. Kao posljedica svega toga možemo vidjeti da agentski prisup prednjači I u metrici semantičkoj sličnosti odgovora koji mjeri usklađenost generisanog i referentnog odgovora. Isto objašnjenje važi I za dosta bolju relevantnost odgovora kod agentskog rješenja.

Poredeći sekundarne osobine ovih sistema, očigledno je da je agentski pristup dosta kompleksniji, zahtjeva više vremena za razvoj I troškovi održavanja arhitekture su dosta veći. Ali glavna prednost ovog pristupa je što je lako proširiv i daje bolje, relevantnije I pouzdanije odgovore.

6. ZAKLJUČAK

U radu je predstavljen razvoj i implementacija jednostavne i agentske implementacije RAG sistema. Motivacija za razvoj sistema je uočiti prednost agenata u okviru ovog koncepta s obzirom na brzi razvoj tehnologije. Glavni cilj ovog rada je uporediti spomenute arhitekture, istaći prednosti i mane oba pristupa, kao i potencijalne prijedloge praktične primjene.

Na osnovu rezultata možemo zaključiti da jednostavnija implementacija RAG sistema se relativno brzo postavlja, nema složenu arhitekturu i ima linearan tok izvršavanja akcija. Dok sa druge strane agentski RAG daje detaljnije odgovore, ima sposobnost rezonovanja i značajno prednjači pri odgovaranju na složene upite. Takođe, arhitektura agentskog pristupa je lako proširiva, ali nosi veće troškove i kompleksnija je za razvoj. Stoga prilikom izbora načina implementacije RAG sistema bitno je naći balans između performansi, održavanja i budućeg razvoja.

Za buduća unapređenja, predlaže se optimizacija promptova uz pomoć predloženih alata. Takođe, korisno bi bilo da sistem ima mogućnost učenja tokom konverzacija i pamćenja dobrih praksi. Još jedno ograničenje je ograničen kontekst koji se šalje velikom jezičkom modelu, ukoliko bi se ovaj problem riješio model bi imao više informacija za odgovaranje što bi proizvelo relevantnije odgovore.

7. LITERATURA

[1] What are large language models (LLMs)? https://www.ibm.com/think/topics/large-language-models [датум приступа јун 2025]

[2] Newhauser, Mary; (2024). Introduction to Retrieval Augmented Generation (RAG) https://weaviate.io/blog/introduction-to-rag [датум приступа јун 2025]

[3] Vector Databases: Tutorial, Best Practices & Examples https://nexla.com/ai-infrastructure/vector-databases/ [датум приступа јун 2025]

[4] Prompt Engineering Guide https://www.promptingguide.ai [датум приступа јун 2025]

[5] What is agentic RAG? https://www.ibm.com/think/topics/agentic-rag [датум приступа јун 2025]

[6] Tahir; (2024). What are AI Agents? https://medium.com/@tahirbalarabe2/what-are-ai-agents-f06ef775e78f [датум приступа јун 2025]

[7] What are tools? https://huggingface.co/learn/agents-course/en/unit1/tools [датум приступа јун 2025]

[8] Zilliz званична документација https://docs.zilliz.com/ [датум приступа јул 2025]

[9] Званична документација Langchain-a https://python.langchain.com/docs/introduction/ [датум приступа јун 2025]

[10] Званична документација Langgraph-a https://langchain-ai.github.io/langgraph/concepts/why-langgraph/ [датум приступа јун 2025]

[11] Званична документација за Ragas радни оквир https://docs.ragas.io/en/stable/getstarted/ [датум приступа август 2025]

Kratka biografija:

Ivana Kašiković rođena je 03. oktobra 2000. godine u Trebinju. Kao nosilac diplome „Vuk Karadžić“ 2019. godine završava gimnaziju „Jovan Dučić“, nakon čega upisuje Fakultet tehničkih nauka, smer Softversko inženjerstvo i informacione tehnologije. Po završetku studija u roku, upisuje master akademske studije i iste završava 2025. godine.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.41
DOI: https://doi.org/10.24867/33BE22Stokic

BEZBEDNOST PODATAKA U KONTEKSTU BIG DATA

DATA SECURITY IN THE CONTEXT OF BIG DATA

Aleksa Stokić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu su istražene specifične pretnje i ranjivosti u okviru Big Data i analizirana primena savremenih tehnika i alata za bezbednost podataka. Posebna pažnja je posvećena principima kao što su autentifikacija, enkripcija i zaštita od brute force i SQL injection napada. Takođe, na primeru veb aplikacije je prikazan primer implementacije bezbednosnih mehanizama.*

Ključne reči: *Big Data, Bezbednost, 2FA, Jake lozinke, Enkripcija, Brute force, SQL injection*

Abstract – *In this paper, specific threats and vulnerabilities within Big Data were explored and the application of modern techniques and tools for data security wa analyzed. Special attention was given to principles such as authentication, encryption and protection against brute force and SQL injection attacks. Additionallz, the implementation of securit mechanisms was demonstrasted thtough the example of a web application.*

Keywords: *Big Data, Security, 2FA, Strong passwords, Encryption, Brute force, SQL injection*

1. UVOD

Big Data donosi značajne prednosti u analizi i donošenju odluka, ali istovremeno predstavlja ozbiljan izazov po pitanju bezbednosti i privatnosti korisnika. Ogromna količina i raznolikost podataka, koja često uključuje osetljive lične informacije, povećava rizik od zloupotrebe i otkrivanja identiteta. Cilj rada je da analizira pretnje i ranjovsti u okviru Big Data i prikaže primenu savremenih bezbedonosnih tehnika kao što su enkripcija, autentifikacija i zaštita od napada. Kao praktičan primer, razvija se veb aplikacija za turističku agenciju koja će demonstrirati implementaciju ovih mehanizama radi zaštite podataka korisnika.

2. BIG DATA

Big Data predstavlja jedan od najznačajnijih fenomena savremenog digitalnog doba. Rast obima, brzine i raznovrsnosti podataka zahteva nove pristupe njihovom skladištenju, obradi i analizi. Dok je tradicionalna obrada podataka ograničena kapacitetom računarskih sistema, Big Data omogućava upotrebu masivnih i razuđenih skupa podataka kako bi se iz njih izvukle vrednosti.

Ključne karakteristike Big Data opisuju se korz koncept "5V": obim, brzina, raznovrsnost, verodostojnost i vrednost. Upravo kombinacija ovih elemenata čini Big Data jedinim načinom da se odgovori na savremen potrebe u oblasti kao što su finansije, zdravstvo, industrija i trgovina.

Razvoj cloud infrastrukture omogućio je veću fleksibilnost i elastičnost u radu sa velikim podacima. Koncept Big Data as a Service (BDaaS) integriše prednosti cloud-a i analitičkih alata, pružajući organizacijama pristup računarskim resursima bez potrebe za sopstvenim skupim sistemima. Među najvažnijim tehnologijama ističu se Hadoop, MapReduce, Hive i Spark, koji omogućavaju paralelnu obradu i upravljanje velikim podacima.

Big Data danas predstavlja industriju koja menja tržište rada, stvara nove poslovne modele i podstiče razvoj digitalne ekonomije. Kompanije koje se oslanjaju na podatke beleže veću produktivnost, bržu reakciju na tržišne promene i bolje razumevanje korisnika. Istraživanja pokazuju da organizacije orjentisane na podatke imaju znatno veće stope rasta i inovacija u odnosu na konkurenciju.

Ipak, uz brojne prednosti javljaju se i izazovi, kao što su bezbednost i privatnsot. Pitanja regulacije, kao i etičke dileme u vezi sa korišćenjem podataka, ostaju otvorena i predstavljaju ključni pravac budućeg razvoja ove oblasti.

Zaključno, Big Data nije samo tehnološki već i ekonomski i društveni fenomen, koji značajno oblikuje poslovanje, nauku i svakodnevni život. Njegov dalji razvoj zavisiće od balansa između tehničkog napretka, regulatornih okvira i etičkih standarda u korišćenju podataka.

3. BEZBEDNOST PODATAK U KONTEKSTU BIG DATA

Bezbednost po dataka u Big Data okruženjima predstavlja kompleksnu kombinaciju metodologija, tehnologija i dobrih praksi usmerenih na zaštitu poverljivosti, integriteta i dostupnosti podataka. Veliki obim, raznolikost i dinamika podataka koji potiču iz IoT uređaja, senzora i društvenih mreža nameću nove izazove koje tradicionalni modeli zaštita ne mogu u potpunosti rešiti. Pored tehničkih rizika, značajan problem predstavlja i privatnost, jer savremene analitike omogućavaju deanomalizaciju pojedinaca, zbog čega regulative poput GDRP i HIPAA propisuju stroge standarde obrade i čuvanja podataka.

Istorijski gledano, zaštita podataka evoluirala je od fizičkih mera i osnovne enkripcije, preko razvoja mrežnih protokola (SSL/TLS), do sveobuhvatnih pravnih okvira

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio Prof. Dr Aleksandar Kupusinac

kao što su GDRP i CCPA. Pored njih, međunarodni standardi poput ISO/IEC 27001 i PCI DSS definišu kontrole i procedure za bezbedno upravljanje inforamcijama. Tehnološki napredak doneo je i nove mehanizme – homomorfnu enkripciju, blockchain za integritet podataka, kao i koncepte diferecijalne privatnosti i k-anonimnosti.

U oblasti metodologija, posebno se ističu Risk Managment Framework (RMF), Security by Design, Zero Trust Architecture, Data Lifecycle i procene uticaja na privatnost (PIA/DPIA). Ovi okviri omogućavaju sistematičan pristup zaštiti podataka, od inicijalnog projektovanja, preko kontinuiranog upravljanja rizikom, do procene usaglašenosti sa regulativom.

Tehnološka implementacija zaštite obuhvata enkripciju u mirovanju i tranzitu, autentifikaciju i autorizaciju, zaštitu od brute force i SQL injection napada, kao i sisteme za logovanje i praćenje anomalija. Savremene Big Data platforme zahtevaju skalabilna rešenja koja mogu raditi u realnom vremenu i u distribuiranim cloud okruženjima.

Praktične mere podrazumevaju primenu enkripcije, segemntaciju mreže, kontrolu pristupa zasnovanu na ulogama, analizu mrežnog saobraćaja, kontinuirano praćenje cloud okruženja, redovno pravljenje rezervnih kopija i planove za brz odgovor na incidente. Podjednako je važna i obuka zapslenih, jer ljudski faktor često predstavja najslabiju kariku. Uočavanje internih pretnji i stalno praćenje usaglašenosti sa regulativom dodatno jača bezbedonosnu kulturu organizacije.

Bezbednost podataka u Big Data kontekstu nije samo tehnički izazov, već i strateško pitanje koje zahteva integraciju metodologija, tehnologija, praksi i regulatornih okvira. Samo takav holistički pristup omogućava izgradnju pouzdanih i održivih sistema koji štite podatke, grade poverenje i omogućavaju odgovorno korišćenje Big Data analitike.

4. IMPLEMENTACIJA BEZBEDONOSNIH PRAKSI

Intenzivan razvoj tehnologije, zajedno sa globalnom povezanošću, nametnuo je potrebu za sistematičnom implementacijom bezbedonosnih praksi koje osiguravaju integrtiet, povreljivost i dostupnost informacija. Ovaj rad daje pregled najvažnijih tehnika i metoda koje predstavljaju temelj moderne sajber bezbednosti, uz poseban naglasak na primenu u Big Data i veb sistemima.

4.1. Jake lozinke i autentifikacija

Jedna os osnovnih mera zaštite predstavlja upotreba jakih i jedinstvenih lozinki, koje značajno umanjuju rizik od neovlašćenog pristupa. Istraživanja pokazuju da složene lozinke sa njmanje 12 karaktera, kombinacija slova, brojeva i simbola, mogu izdržati brute force napade u vremenskom okviru od više hiljada godina. Ipak, lozinke same po sebi nisu dovoljne. Zato je u praksi sve zastupljenija dvofaktorska autentifikacija (2FA), koja uvodi dodatni sloj sigurnosti kombinovanjem faktora znanja (lozinka), posedovanja (tokeni, mobilni uređaj) i inherencije (biometrijski podaci). Otvoreni standardi kao što su FIDO, Oauth2.0, OpenID Connect i SAML omogućavaju interoberabilnost i otpornost na phishing napade, dok budući pravci uključuju autentifikaciju bez lozinki i koncepte decentralizovanog identiteta.

Iako značajno povećava bezbednost, ona nije imuna na napade. Slabosti mogu nastati usled zloupotrebe porcesa oporavka naloga, kompromitacije tokena ili nesigurnosti SMS kanala. Nacionalni instituti (na primer NIST) već preporučuju izbegavanje SMS-a u sistemima visokog rizika. Ipak, prednosti 2FA – jednostavnost, dostupnost i visok nivo sigurnosti – čine je jednim od najpouzdanijih rešenja za zaštitu digitalnih servisa.

4.2. Zaštita od brute force napada

Brute force napadi, zasnovani na sistematičnom isprobavanju kombinacija lozinki i ključeva, ostaju jedan od najrasprostranjenijih metoda kompromitovanja naloga. Iako su jednostavni i teoretski nepogrešivi, njihova efikasnost zavisi od dužine i složenosti akreditiva. Savremeni sistemi primenjuju više mera zaštite: ograničenje broja prijava, CAPTCHA mehanizme, crne liste IP adresa, praćenje mrežnog saobraćaja kao i uvođenje višefaktorske autentifikacije. Korisnicima se preporučuje upotreba jedinstvenih lozinki, mendžera lozinki i izbegavanje nebezbednih sajtova. Na ovaj način brute force napada postaju sve manje delotvorni, ali i dalje predstavljaju ozbiljnu pretnju ako se sistemi ne održavaju redovno i bezbednosno ne ažuriraju.

4.3. Primena enkripcije

Enkripcija predstavlja jedan od najmoćnijih mehanizma zažtite podataka, jer osigurava njihovu povreljivost čak i u slučaju presretanja i krađe. Primena obuhvata i podatke u mirovanju i podatke u prenosu, uz poseban značaj u oblastima finansija, e-trgovina i e-uprave. U Srbiji je ova oblast pravno uređena Zakonom o elektronskom dokumentu, dok GDRP na nivou EU posebno prepoznaje enkripciju kao preporučenu meru.

Najznačajniji algoritmi uključuju simetrični AES iasimtrični RSA. Simetrična enkripcija je brza i efikasna za velike količine podataka, ali zahteva siguran prenos ključeva. RSA rešava ovaj problem upotrebom javnih i privatnih ključeva, iako je računarski zahtevniji. U praksi se često primenjuje hibridni pristup – AES za sadržaj, RSA za razmenu ključeva. Glavni izazovi odnose se na upravljanje kriptografskim ključevima, kao i na rizike od zloupotrebe enkripcije u ransomware napadima.

4.4. Zaštita od SQL injection napada

SQL injection je jedna od najopasnijih veb ranjivosti koja omogućava napadaču manipulaciju upitima ka bazi podataka. Posledice uključuju neovlašćen pristup, krađu ili brisanje podataka, pa čak i eskalaciju privilegija. Ključne mere prevencije obuhvataju validiranje unosa, upotrebu parametrizovanih upita i skladištenje porcedura, kao i primenu belih lista za kontrolu unosa. Redovno skeniranje aplikacija alatima kao što su Burp Scanner ili Acunetix omogućava rano otkrivanje ranjivosti. Primeri iz prakse pokazuju da dinamičko kreiranje SQL upita putem konkatenacije string-ova predstavlja najveći rizik, dok parametrizovani upiti i savremeni okviri pružaju pouzdanu zaštitu.

Implementacije bezbedonosnih praksi nije više opcija, već je nužnost u digitalnom dobu. Jake lozinke, 2FA, enkripcija, zaštita od brute force i SQL injection napada predstavljaju temeljnemehanizme za izgradnju otpornih sistema. Međutim, svki od ovih mehanizama nosi svoja ograničenja i zahteva kontinuirano unapređenje. Budućnost bezbednosti leži u integraciji više faktora – biometrijskih, bihejvioralnih i decentralizovanih – koji zajedno mogu osigurati robusnije i prilagodljivije sisteme. Samo sistematičnim i sveobuhvatnim pristupom moguće je izgraditi poverenje korisnika i očuvati integritet inforamcionih resursa u sulovima stalno rastućih sajber pretnji.

5. INTEGRISAN PRIMER IMPLEMENTACIJE BEZBEDONOSNIH MEHANIZAMA U VEB APLIKACIJI

U okviru ovog rada razvijena je aplikacija koja služi kao ilustrativni primer integracije ključnih bezbedonosnih mera. Iako funkcionalno zasnovana na modelu turističke agencije, aplikacija ima primarno edukativni i demonstrativni karakter, pokazujući kako se kroz jedan sistem mogu implementirati mehanizmi autentifikacije, zaštite podataka u prenosu i skladištenju, kao i prevencije zlonamernih aktivnosti.

5.1. Tehnološki stek i arhitektura

Aplikacija je izgrađena primenom savremenog tehnološkog steka. Frontend je realizovan u React.js-u, koji omogućava komponentni razvoj jednostraničnih aplikacija i komunikaciju sa serverom preko HTTP zahteva (Axios). Backend je izgrađen u Python-u korišćenjem Flask framework-a, koji omogućava izgradnju RESTful API servisa i modularni arhitekturu. Za upravljanje podacima korišćenja je MongoDB NoSQL baza, pogodna za dinamične strukture. Arhitektura je troslojna, sa jasnom podelom na kontrolere (obrada zahteva), servise (poslovna logika) i repozitorijume (rad sa bazom), što doprinosi lakšem održavanju i proširivosti.

5.2. Implementacija bezbedonosnih mehanizama

5.2.1. Provera jačine lozinki

Prilikom registracije vrši se dvostruka provera lozinki korišćenjem regulatornih izraza. Ovaj mehanizam osigurava da lozinke zadovoljavaju minimalne kriterijume složenosti. Čime se umanjuje rizik od brute force napada.

```
28  const isStrongPassword = (password) => {
29      const regex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{12,}$/;
30      return regex.test(password);
31  };
```

Slika 1. *Provera jačine lozinke na frontend delu aplikacije*

5.2.2. 2FA

Realizovana je integracija sa TOTP standardom putem PzOTP biblioteke i prikaza QR koda pomoću Qrcode modula. Nakon inicijalnog podešavanja, korisnik prilikom svkaog logovanja unosi jednokratni kod iz aplikacije kao što je Google Authenticator. Ovim se znatno povećava sigurnost prijave.

```
def generate_2fa_secret(email):
    secret = pyotp.random_base32()
    totp = pyotp.TOTP(secret)
    otp_uri = totp.provisioning_uri(name=email, issuer_name="MasterTuristickaAgencija")

    qr = qrcode.make(otp_uri)
    buffer = BytesIO()
    qr.save(buffer, format="PNG")
    qr_base64 = base64.b64encode(buffer.getvalue()).decode()

    return secret, qr_base64
```

Slika 2. *Generisanje QR koda u Python-u*

```
def verify_otp_code(email, otpCode):
    if not otpCode:
        return jsonify({'message': 'OTP code is required!'}), 400

    current_user = check_existing_user(email)
    secret = current_user['twoFASecret']
    result = verify_otp(secret, otpCode)

    if result:
        try:
            session.clear()
            session['passenger_id'] = str(current_user['_id'])
            session['email'] = str(current_user['email'])
            passenger_id = str(current_user['_id'])

            token_payload = {
                'passenger_id': passenger_id,
                'passenger_email': current_user['email'],
                'exp': datetime.utcnow() + timedelta(hours=1)
            }
            reset_failed_attempts(current_user['email'])
            token = encode(token_payload, os.environ.get('SECRET_KEY', 'MasterTuristickaAgencijaBack'), algorithm='HS256')

            return jsonify({'token': token}), 200

        except Exception as e:
            return jsonify({'error': str(e)}), 400
    else:
        return jsonify({'message': 'Invalid OTP code!'}), 400
```

Slika 3. *Servis za proveru OTP koda*

5.2.3. Zaštita od brute force napada

Implementirana je višeslojna strategija koja obuhvata globalni rate-limiting (Flask-Limiter), ograničenje broja zahteva po kritičnim rutama (/signin, /signup), kao i privremeno blokiranje naloga nakon više neuspelih pokušaja. Ovim se efikasno sprečavaju automatizovani napadi i kreiranje lašnih naloga.

```
12  @auth_signin_blueprint.route( rule: '/signin', methods=['POST'])
13  @limiter.limit("5 per minute")
14  def sign_in_controller():
```

Slika 4. *Ograničenje na maksimum 5 pristupa za rutu*

```
def sign_in_service(email, password):
    current_user = check_existing_user(email)
    if not current_user:
        return jsonify({'message': 'User with this email does not exist! Please check email or create account!'}), 400

    if int(current_user['lockUntil']) > int(time.time()):
        return jsonify({'message': 'Account temporarily locked. Try again later.'}), 400

    if not bcrypt.checkpw(password.encode('utf-8'), current_user['password']):
        if current_user['failedAttempts'] > 5:
            lock_time = int(time.time()) + 5 * 60
            set_lock(current_user['email'], lock_time)
            return jsonify({'message': 'Account locked. Try again later.'}), 400
        else:
            attemps = current_user['failedAttempts'] + 1
            increment_failed_attempts(current_user['email'], attemps)
            return jsonify({'message': 'Wrong password!'}), 400
```

Slika 5. *Logika za blokiranje korisničkog naloga na backend delu aplikacije*

5.2.4. Zaštita od NoSQL injection napada

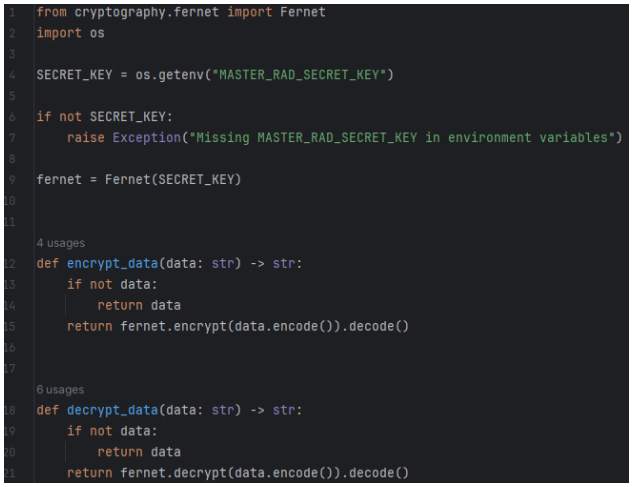
Kreiran je poseban validacioni servis koji proverava sve unose pre slanja u bazu. Pored regularnih izraza za verifikaciju formata, primenjuje se parametrizovaniupiti u pymongo biblioteci, čime se eliminiše rizik od direktne interpolacije korisničkog unosa u upit.

```python
def contains_sql_injection(value: str) -> bool:
    if not isinstance(value, str):
        return False

    patterns = [
        r"(?i)(\bor\b|\band\b)\s+\d+=\d+",
        r"(?i)drop\s+table",
        r"(?i)select\s.+\sfrom",
        r"(?i)insert\s+into",
        r"(?i)update\s+\w+\s+set",
        r"(?i)delete\s+from",
        r"['\";#`]",
        r"--",
    ]

    for pattern in patterns:
        if re.search(pattern, value):
            return True

    return False


10 usages
def validate_email(email: str) -> bool:
    pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
    return re.match(pattern, email) is not None


def validate_username(username: str) -> bool:
    return bool(re.match( pattern: r'^\w{1,30}$', username))
```

Slika 5. *Izgled validacionog servisa na backend delu aplikacije*

5.2.5. Implementacija enkripcije

Osetljivi podaci (JMBG, broj pasoša) čuvaju se u šifrovanojformi korišćenjem simterične enkripcije (Fernet iz crzptography biblioteke). Tajni ključ se čuva u .env fajlu, što obezbeđuje minimalnu zaštitu od kompromitacije koda.



```python
from cryptography.fernet import Fernet
import os

SECRET_KEY = os.getenv("MASTER_RAD_SECRET_KEY")

if not SECRET_KEY:
    raise Exception("Missing MASTER_RAD_SECRET_KEY in environment variables")

fernet = Fernet(SECRET_KEY)


4 usages
def encrypt_data(data: str) -> str:
    if not data:
        return data
    return fernet.encrypt(data.encode()).decode()


6 usages
def decrypt_data(data: str) -> str:
    if not data:
        return data
    return fernet.decrypt(data.encode()).decode()
```

Slika 6. *Enkripcioni servis*

```
_id: ObjectId('686be3d252825d4d84704d97')
email : "stokic.aleksa.01@gmail.com"
password : Binary.createFromBase64('JDJiJDEyJHkuWSS6VDBVbGJEbHdsRUI4QWRyNXVBLlZxdGR2cUx2M2R5by9VV3Q1OU5TWWlyYW9MOEIy', 0)
first_name : "Aleksa"
last_name : "Stokic"
birthday : "2001-01-19"
is2FASetUp : true
twoFASecret : "Z5QOVSPF7Y3VADJMAXTQYCERVTNIH3SR"
failedAttemps : 0
lockUntil : 0
JMBG : "gAAAAABoa-PSQ-0-Z_1bJKx1KMnjOIei1Yqo7O4K487CxvUGthXEZwXdXPdLrMPb4u8ZC9e…"
passportNumber : "gAAAAABoa-PS3vSH91n1u5QVbPScyWLpO1UhSfUzY3g1M9WykcOS_Vuz3DCnPJpX20Cr5q…"
failedAttempts : 0
```

Slika 7. *Izgled enkriptovanih podataka iz baze*

Implementacija prikazanih mehanizama pokazuje da je moguće izgraditi veb apliakciju koja ispunjava savremene bezbedonosne standarde uz minimalne resurse i upotrebu dostupnih biblioteka. Sistem je edukativno osmišljen kako bi demonstrirao najčešće načine odbrane. Ipak, naglašeno je da svaki mehanizam ima svja ograničenja, pa se preporučuje kombinovanje više mera i njihovo kontinuirano unapređenje.

Integracija bezbedonosnih mehanizama u veb aplikaciji predstavlja uslov za njihovu pouzdanu i bezbednu upotrebu. Primer aplikacije turističke agencije pokazuje da pravilnom primenom validacije lozinki, 2FA, zaštite od brute force i NoSQL injection napada, kao i enkripcije osetljivih podataka, moguće je značajno umanjiti rizik od kompromitacije sistema. Rad pokazuje da kombinacija dobrih praksi, adekvatnog tehnološkog izbora i edukacije korisnika predstavlja najbolji pristup izgradnji otpornih i pouzdanih veb aplikacija u savremenom digitalnom okruženju.

5. ZAKLJUČAK

U eri digitalne transformacije i eksponencijalnog rasta količine podataka, bezbednost u Big Data okruženjima postaje jedan od ključnih preduslova za očuvanje integriteta, poverljivosti i dostupnosti informacija. Veliki podaci pružaju izuzetne mogućnosti za naprednu analizu i donošenje strateških odluka, ali istovremeno uvode broje izazove u pogedu zaštite osetljivih informacija i otpornosti sistema na sajber napade. Karakteristike velikih podataka zahtevaju nove, fleksibilne i skalabilne pristupe u dizajnu i implementaciji bezbedonosnih mera.

U ovom radu istraženi su ključni bezbedonosni aspekti Big Data okruženja, pri čemu su analizirane najčešće pretnje kao što su krđa identiteta, neovlašćeni pristup, brute force i injection napadi, kao i izazovi uporavljanja autentifikacijom i privatnošću. Teorijski okvir upotpunjen je praktičnim delom u okviru kojeg je razvijena veb aplikacija zasnovana na arhitekturi Flask-React.js-MongoDB. U aplikaciji su implementirani mehanizmi kao što su dvofaktroska autentifikacija, kontrola jakih lozinki, validacija unosa, enkripcija osetljivih podataka i ograničenje zahteva, čime je pokazano kako se kroz slojevit pristup može značajno povećati bezbednost sistema.

Rezultati ukazuju da ne postoji univerzalno rešenje za sve pretnje i da bezbednost mora biti tretirana kao kontinuirani proces koji podrazumeva stalno praćenje, prilagođavanje i unapređeje mehanizama. Posebno je istaknuta potreba za integracijom sistema za detekciju anomalija u realnom vremenu, primenom zero-trust arhitekture i korišćenjem mašinskog učenja u otkrivanju zloupotreba.

Zaključno, rad potvrđuje da je bezbednost podataka neodvojiv deo Big Data sistema i da samo interdisciplinarni pristup – koji kombinuje tehničke, organizacione i edukativne mere – može obezbediti pouzdanost i otpornost u savremenom digitalnom okruženju.

6. LITERATURA

[1] https://cloud.google.com/learn/what-is-big-data

[2]https://www.forbes.com/sites/gartnergroup/2013/03/27/gartners-big-data-definition-consists-of-three-parts-not-to-be-confused-with-three-vs/

[3] https://www.turing.com/resources/big-data-security#what-is-big-data-security?

[4]https://www.keepersecurity.com/blog/2023/08/31/what-makes-a-strong-password/

[5]https://www.techtarget.com/searchsecurity/definition/two-factor-authentication

[6]https://www.fortinet.com/resources/cyberglossary/brute-force-attack

[7]https://www.techtarget.com/searchsecurity/definition/encryption

[8]https://www.acunetix.com/websitesecurity/sql-injection/

Kratka biografija:

Aleksa Stokić je rođen 2001. godine u Požarevcu, Srbija. Završio je Požarevačku gimnaziju u Požarevcu, 2019. godine. Fakultet Tehničkih Nauka u Novom Sadu je upisao 2019. godine. Diplomirao je u septembru 2023. godine, smer Primenjeno softversko inženjerstvo. Uspešno je ispunio sve akademske obaveze i položio sve ispite predviđene master studijskim programom Primenjeno softversko inženjerstvo.

kontakt: stokic.aleksa.01@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

UDK: 621.31
DOI: https://doi.org/10.24867/33BE23Maksimovic

ПРОЈЕКТОВАЊЕ АУТОМАТИКЕ КЛИМА КОМОРЕ ПОСЛОВНЕ ЗГРАДЕ И ИНТЕГРАЦИЈА СА BMS-ОМ (BUILDING MANAGEMENT SYSTEM)

DESIGN OF AIR HANDLING UNIT (AHU) AUTOMATION FOR A COMMERCIAL BUILDING AND INTEGRATION WITH THE BUILDING MANAGEMENT SYSTEM

Момчило Максимовић, Дарко Марчетић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У овом раду приказано је решење управљања клима комором путем BMS система, са посебним освртом на избор и повезивање компонената, радну логику и сигурносне функције. Детаљно су анализирани елементи клима коморе попут вентилатора, грејача, рекуператора, хладњака, овлаживача и релевантних сензора. Описани су принципи рада и повезивања компонената кроз електричне шеме, као и коришћење ЕС мотора. Обрађена је интеграција са BMS системом уз коришћење Schneider Electric опреме. Обухваћени су критеријуми за безбедан рад, као и логичке секвенце реализоване релејном техником, које прате нормалан и алармни режим рада климатске коморе.*

Кључне речи: *Клима комора, BMS, Релејна техника*

Abstract – *This paper presents a solution for managing an air handling unit (AHU) through a Building Management System (BMS), with a special focus on component selection and interconnection, control logic, and safety functions. Key elements of the AHU such as fans, heaters, heat recovery units, coolers, humidifiers, and relevant sensors are analyzed in detail. The operating principles and wiring of components are described through electrical schematics, including the use of EC motors. The integration with the BMS system using Schneider Electric equipment is elaborated. Safety operation criteria are addressed, along with logic sequences implemented via relay technology that govern both normal and alarm operating modes of the AHU.*

Keywords: *Air kandling unit, BMS, Relay logic*

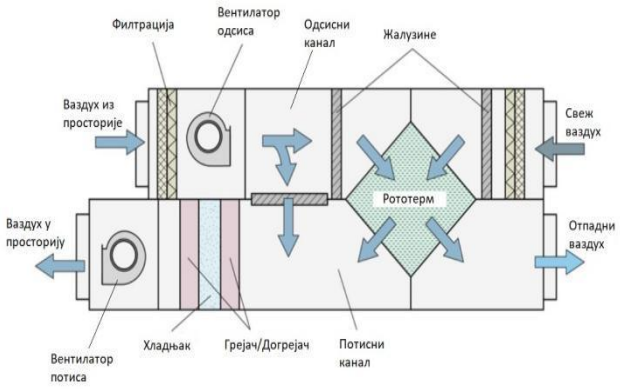
1. УВОД

Клима коморе представљају централни део система за обраду ваздуха у савременим објектима, а захтеви који се пред њих постављају у погледу комфора, енергетске ефикасности и безбедности све су строжи [1]. Циљ овог рада је приказ избора компонената: вентилатора, грејача, рекуператора, овлаживача, хладњака и припадајућих сензора. Обрађено је њихово електрично повезивање и дефинисање логике рада клима коморе у оквиру BMS система, уз посебан акценат на безбедносне функције и флексибилност коју пружа релејна техника у комбинацији са савременим сензорима и актуаторима. Анализирана је улога ЕС мотора, као и интеграција опреме произвођача Schneider Electric, са описом примене њихових PLC уређаја, I/O модула и комуникационих протокола. Оваквим приступом обухваћени су како аспекти практичне имплементације, тако и критеријуми безбедног и поузданог рада у реалним условима експлоатације.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији је ментор био др Дарко Марчетић, ред. проф.

2. ОСНОВНА ФУНКЦИЈА И ПРИНЦИПИ РАДА КЛИМА КОМОРЕ

Принцип рада укључује неколико основних фаза, спољашњи ваздух се усисава кроз улазне каналзе где се припрема за даљу обраду. Обрада ваздуха се врши кроз различите секције унутар клима коморе што се може видети на блок шеми клима моморе слика 1. Након обраде третирани ваздух се дистрибуира кроз систем или директно у просторије. Детаљно су описане секције за мешање ваздуха, филтерске јединице, грејачи и хладњаци, рекуператори топлоте (плочасти и ротациони), овлаживачи, вентилатори (центрифугални и аксијални), као и пратећи сензори и контролни елементи.



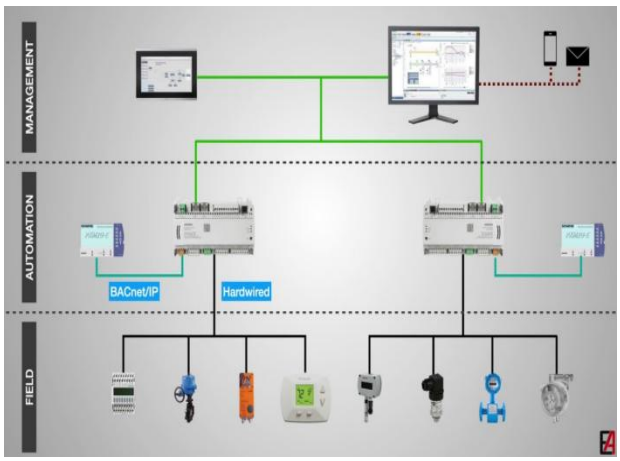
Слика 1. *Блок шема клима коморе са својим елементима*

Дефинисани су принципи управљања компонентама као што су: управљање вентилаторима, функција мразостата и противпожарног термостата, као и значај диференцијалних сензора притиска на филтерима. Рад

такође обухвата и логичке основе управљања системом у нормалном и алармном режиму рада.

3. BUILDING MANAGEMENT SYSTEM (BMS)

BMS представља централизовани систем аутоматизације који омогућава надзор и контролу над стањем система, оптимизацију енергетске ефикасности, интегрисано управљање сигурносним и алармним режимима, аналитику података и праћење утицаја објекта на животну средину. Систем је представљен као слојевита структура која обухвата део који служи за праћење и контролу параметара система (SCADA), аутоматски ниво (контролери и I/O модули) и ниво опреме у пољу (сензори, актуатори и извршни елементи). На слици 2 представљени су сви нивои и начин комуникације између њих. Дакле, веза менаџмент дела и аутоматике представљена је зеленом линијом тако означавајући да је реч о одређеном комуникационом протоколу, док црна линија између слоја аутоматике и опреме у пољу означава да је реч о сигналном ожичењу (конвенцијални сигнали).



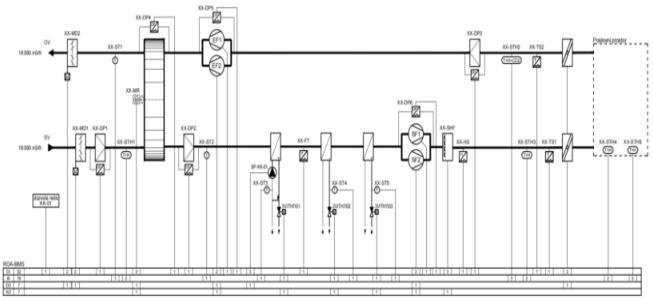
Слика 2. *Нивои комуникације BMS система*

У раду је примењена Schneider Electric платформа, са следећом опремом: PLC контролери (SXWASPXXX10001), дигитални и аналогни улазно/излазни модули (SXWDI16XX10001, SXWUI16XX10001, SXWDOA12X10001, SXWAOV8XX10001), као и напајања за сензоре и контролере (SXWPS24VX10001). Повезивање је могуће реализовати комуникационим протоколима Modbus и BACnet, који омогућавају интеграцију са осталим подсистемима зграде. Функција и намена сваког од примењених модула је детаљно објашњена с акцентом на њихову функцију у прикупљању, обради и преносу сигнала, што омогућава ефикасно и прецизно управљање климатским и техничким условима у објекту кроз BMS систем. Управљачка логика обухвата контролу температуре, влажности, квалитета ваздуха и радне сигнализације. Имплементиране су заштитне функције путем релејне технике и дигиталне логике, укључујући надзор алармних стања, сигурносне сигнале и прецизно секвенцирање рада компонената.

Модул за напајање (SXWPS24VX10001) пружа стабилизовани излазни напон од 24 VDC, опремељен заштитиним функцијама које аутоматски искључују напајње у случају преоптерећења или кратког споја. Пружа подршку за широк опсег улазног напона што је значајно за примене у променљивим енергетским условима. Максимална снага напајања је 30W. Контролер (SXWASPXXX10001) има напредне алгоритме за обраду сигнала и омогућава процесима као што су регулација температуре, притиска, влажности и других параметара у систему климатизације. Подржава стандардне комуникационе протоколе Modbus, BACnet, LON што омогућава интеграцију са различитим системима унутар BMS-а. Поседује више комуникационих портова као што су 2XEthernet, LonWorks, USB host, USB device, 2XRS-485. Проширивост и скалабилност омогућавају додавању до 30 модула, а потрошња контролера износи 10W. Дигитални улазно модул (SXWDI16XX10001) поседује 16 дигиталних канала (безнапонски контакти напонског нивоа 24V DC и 2.4mA). Потрошња модула је 1.6W. Модул универзалних улаза (SXWUI16XX10001) има 16 канала који подржавају аналогне синале у распону од 0-10 VDC , 0-20mA, дигиталне сигнале 24VDC, 2,4mA, отпорничке сигнале (10Ω до 10кΩ или 10кΩ до 60кΩ), температурне улазе (-50°C до 50°C). Потрошња модула је 1.8W. Модул дигиталних излаза (SXWDOA12X10001) поседује 12 канала чији контакти могу радити на напонском нивоу 250VAC или 30 VDC. Максимална струја по каналу 2A, што омогућава директно управљање мањим оптерећењима. Потрошња модула је 1.8W. Аналогно излазни модул (SXWAOV8XX10001) има 8 излаза којима се може генерисати напонски (0-10 VDC) или струјни (1mA do 2mA) сигнали, потрошње 0,7W. Сви наведени модули се монтирају на DIN шину преко одговарајући подножија а комуникацију са контролером преко RS485.

4. ОПЕРАТИВНА ЛОГИКА РАЗМАТРАНЕ КЛИМА КОМОРЕ КК-01

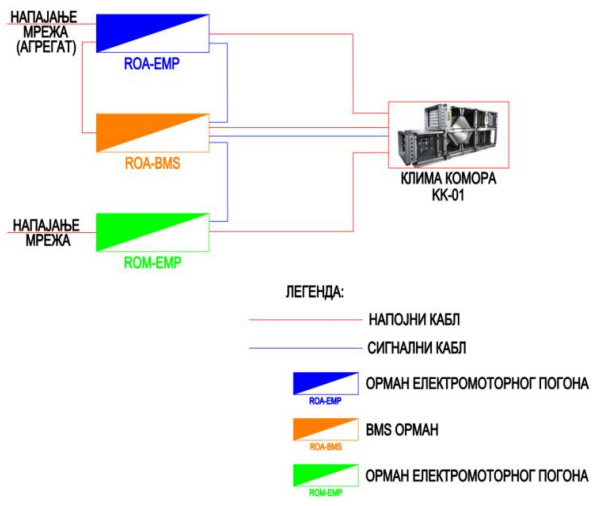
На слици 3 приказана је апликативна шема клима коморе КК-01 чији рад се заснива на контроли кључних параметара као што су температура, влажност, притисак и квалитет ваздуха. Оперативна логика система заснива се на прикупљеним сигналима приказаним у апликативној шеми, а циљ је обезбеђивање комфорних и енергетски ефикасних услова у простору.



Слика 3. *Апликативна шема клима коморе КК-01*

Приоритет система је одржавање квалитета ваздуха у простору, што се постиже уношењем већих количина свежег ваздуха у случајевима повећане запрљаности. Пошто комора КК-01 нема бајпас који раздваја рециркулацију од рекуперације, уколико је извучени ваздух превише загађен, систем смањује или зауставља рад рекуператора. Температура ваздуха контролише се мерењем на издувној решетки или у самом простору, а регулацију врши PI регулатор који управља вентилима измењивача (топле или хладне воде). Притисак или проток ваздуха у каналима одржава се мерењем у потисном и одсисном делу и управљањем брзином обртаја вентилатора путем PID алгоритма преко фреквентних регулатора или директно EC моторима који су примењени у разматраној клима комори КК-01. Важан део система чине и парни овлаживачи, који служе за довођење влажности на задате вредности након иницијалног хлађења ваздуха. Целокупан процес прати се сензорима влажности, уз хидростате који сигнализирају критичне вредности у каналу.

Ради бољег разумевања функционалне повезаности свих компонената клима коморе, потребно је схватити начин на који су уређаји међусобно повезани и како се обезбеђује сигурност система путем хардверских релеја. Сви елементи су груписани у више ормана са засебним напајањем и комуникационим линијама, а њихова узајамна повезаност приказана је кроз блок шему напајања (Слика 4).



Слика 4. *Блок шема напајања*

1. ROA-EMP – Орман се напаја из мреже са могућношћу пребацивања на агрегатско напајање. Агрегатско напајање је неопходно обезбедити за циркулациону пумпу и вентил грејача (измењивач топлоте) како би се спречило смрзавање система при нестанку струје.

2. ROM-EMP – Орман се напаја из мреже. Напаја све сензоре, актуаторе и моторе. Због мање критичности ових елемената није потребно обезбеђивати агрегатско напајање.

3. ROA-BMS – Орман у коме се налази програмабилни контролер и његови модули. Напаја се из ROA-EMP ормана и има обезбеђено резервно агрегатско напајање ради континуираног рада контролера и безбедности система.

Веза ормана електро-моторних погона и ормана ROA-BMS у коме се налазе програмабилни контролери остварује се релејном техником, системом улаза-излаза. Детаљне шеме повезивања и интеграција BMS и EMP ормана приказани су у петом поглављу.

5. ИНТЕГРАЦИЈА КОМПОНЕНАТА РАЗМАТРАНЕ КЛИМА КОМОРЕ СА BMS-ом

Уместо комуникационих протокола, размена сигнала између компонената коморе и контролера у BMS орману реализује се путем релејне технике, системом улаза/излаза. Овај приступ обезбеђује поузданост, електричну изолацију, флексибилност при пројектовању логике управљања и лаку интеграцију са различитим BMS решењима без потребе за додатним адаптацијама. Релејна техника је класичан начин повезивања у индустријској аутоматици [2] где се логички сигнали обрађују преко релеја ради сигурног и поузданог управљања.

5.1. Релеји као кључни елементи интеграције

Релеји су електромеханички уређаји који преклапањем контаката омогућавају или прекидају проток струје ка одређеним потрошачима [3], при чему се у пројекту користе релеји са 1CO и 4CO контактима. Примена обухвата прикупљање сигнала са сензора, управљање актуаторима и контролу безбедносних уређаја. Захваљујући својој структури, релејна техника омогућава флексибилно формирање логике рада и електричну заштиту, јер сваки сигнал може бити обрађен пре уласка у PLC, чиме се повећава поузданост и безбедност целог система.

5.2. Интегрисана опрема у клима комори КК-01

MD20SR-24TS је Schneider Electric актуатор са моментом од 20 Nm и повратном опругом (Failsafe функција), што обезбеђује враћање жалузине у задати положај при нестанку напајања. Поседује сигналне контакте (TS) за повратну информацију о положају. Тип управљања је двоположајни ON/OFF.

Жалузина се напаја преко трансформатора 230/24VAC из ормана ROA-BMS, а отварање се врши задавањем дигиталне командом са контролера. У случају прекида напајања, захваљујући повратној опрузи, жалузина се затвара. Стање отворености и затворености прати се преко повратних сигнала са клема S1–S2 (затворено) и S4–S6 (отворено), који се доводе на дигиталне улазе контролера. Исти принцип важи за обе жалузине — свежег и отпадног ваздуха.

Сензори диференцијалног притиска прате пад притиска на филтерима, вентилаторима и рототерму ради детекције алармних стања и потребе за одржавањем. На филтерима се користе сензори SDP910-300, директно повезани на дигиталне улазе контролера без потребе за додатним напајањем. Сензори на вентилаторима (SDP910-1000) сигнализирају рад мотора преко релеја у орману ROM-EMP. Сензор на рототерму (SDP910-500) алармира у случају зачепљености или промене протока. Сви сензори доприносе ефикасној контроли

и заштити клима коморе путем повратних сигнала у BMS.

У систему се користе четири типа температурних сензора:

1. Цевни сензор (STP100-100) – мери температуру течности у грејачу, хладњаку и догрејачу; поставља се у чауру у цевоводу.

2. Каналски сензор температуре и влаге (SHD100-T) – прати температуру и влажност у вентилационим каналима; потребно му је напајање.

3. Каналски сензор температуре, влаге и CO_2 (SCD110-H) – омогућава контролу квалитета повратног ваздуха ради уштеде енергије.

4. Просторни сензор температуре и влаге (SLAXX2) – кључан за прецизну регулацију у просторији, јер мери стварне услове у простору и омогућава BMS-у да оптимизује рад клима коморе.

Сви сензори се директно везују на BMS и омогућавају ефикасну и прецизну контролу климатизације.

Пожарни термостати RAK-TW1000.HB се постављају на потисни и одсисни канал клима коморе и служе за детекцију прегревања, након чега преко релеја искључују вентилаторе и активирају аларм. Повезани су са BMS-ом и могу активирати жалузине, аларме, или проследити сигнал систему за дојаву пожара. Поред тога, релеји KA3 и KA4 повезани на противпожарну централу прекидају напајање у случају пожара, затварају клапне и онемогућавају ширење ватре кроз вентилацију.

У системима грејања клима комора, циркулациона пумпа, вентил грејача и мразостат заједно обезбеђују ефикасну контролу температуре и заштиту од смрзавања. Циркулациона пумпа обезбеђује сталан проток топле воде кроз измењивач, док вентил грејача регулише количину воде и омогућава аналогно управљање. Мразостат прати температуру у близини измењивача и уколико она падне испод критичне вредности (нпр. 5°C), активира релеј који искључује вентилаторе и шаље сигнал BMS-у или директно активира циркулациону пумпу и отвара вентил грејача. На тај начин се спречава смрзавање воде у грејачу и могућа оштећења инсталације. Узајамна координација ових уређаја обезбеђује поуздан и безбедан рад HVAC система чак и при екстремно ниским температурама.

У овом систему, притисак у просторији се регулише путем одсисних и потисних вентилатора (по два од сваке врсте) са ECBlue моторима, који омогућавају енергетски ефикасан, тих и поуздан рад. ECBlue мотори имају интегрисану електронику, подршку за управљање преко 0–10V или Modbus сигнала, софт-старт функцију и дуг век трајања. Команда за рад мотора може се дати ручно или преко BMS-а, док се повратне информације о раду и кварy добијају преко релевантних клема. Брзина се регулише слањем аналогног сигнала (0–10V), а паралелно повезани вентилатори омогућавају равномерну расподелу оптерећења и повећану поузданост система.

Рототерм (KK01-MR) је електромотор снаге 0,5 kW који обезбеђује континуирану ротацију ротационог рекуператора у клима комори ради ефикасног преноса топлоте (и влаге) са издувног на улазни ваздух. Управљање се може вршити ручно или преко BMS-а, уз могућност аналогне регулације брзине (0–10 V), док се сигнали рада и квара преносе преко релеја на синоптику и BMS. Напајање рототерма обезбеђено је из ормана ROM-EMP.

6. **ЗАКЉУЧАК**

Клима коморе су кључна компонента система централне климатизације, јер омогућавају контролисано довођење, обраду и дистрибуцију ваздуха у складу са захтевима простора и комфора корисника. Оне убацују свеж, филтриран и обрађен ваздух у простор, док истовремено извлаче топао или загађен ваздух, чиме се одржава оптималан квалитет унутрашње средине.

Избор и конфигурација опреме у клима комори КК-01 и интеграција хардверских компонената са BMS системом од кључног су значаја за квалитетан, ефикасан и сигуран рад ових система.

Применом BMS-а омогућено је континуирано праћење и регулисање параметара, што води ка значајним уштедама енергије и продужетку века опреме. Комбинација хардверских ограничења преко релеја и софтверске контроле кључна је за постизање високе енергетске ефикасности и поузданости система.

7. **ЛИТЕРАТУРА**

[1] Велимир Чонградац, *Аутоматика у паметним стамбено-пословним објектима*, 21000 Нови Сад, Трг Доситеја Обрадовића 6.

[2] Дарко Марчетић, Марко Гецић, Борис Марчетић, *Програмабилни логички контролери у електроенергетици*, Факултет техничких наука, 21000 Нови Сад, Трг Доситеја Обрадовића 6.

[3] Страхил Ј. Гушавац, *Основни принципи пројектовања у мрежама средњег и ниског напона*, Факултет техничких наука, 21000 Нови Сад, Трг Доситеја Обрадовића 6.

Кратка биографија:

Момчило Максимовић рођен је у Шапцу 1998. год. Дипломски рад на Факултету техничких наука из области Електротехнике и рачунарства – Електроенергетски системи одбранио је 2022.год.
контакт:momcilo.maks7@gmail.com

Дарко Марчетић рођен је 1968. године у Новом Саду. И даље се успешно бави научним истраживањима.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE25Jankovic

SAVREMENI CRM SISTEMI: SALESFORCE PLATFORMA U UNAPREĐENJU POSLOVNIH PROCESA

MODERN CRM SYSTEMS: THE SALESFORCE PLATFORM IN ENHANCING BUSINESS PROCESSES

Marija Janković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Rad istražuje značaj i ulogu CRM sistema u savremenom poslovnom okruženju, detaljan pregled platforme Salesforce u kontekstu digitalne transformacije, prilagođavanja, mogućnosti integracije i primene veštačke inteligencije u cilju unapređenja poslovnih procesa.*

Ključne reči: *CRM sistemi, platforma Salesforce, Integracija veštačke inteligencije, Salesforce AI, Salesforce Einstein, Agentforce*

Abstract – *The paper explores the significance and role of CRM systems in the modern business environment, providing a comprehensive overview of the Salesforce platform's functionalities, with a focus on digital transformation, customization, integration capabilities and the use of artificial intelligence to enhance business processes.*

Keywords: *CRM Systems, Salesforce Platform, Artificial Intelligence Integration, Salesforce AI, Salesforce Einstein, Agentforce*

1. UVOD

Savremena poslovna okruženja, dinamična i složena, izlažu preduzeća svih tipova i obima brojnim izazovima.

Različiti sistemi, neusklađeni timovi, ručni unos i nedostatak automatizacije procesa, otežavaju obradu i integraciju podataka, povećavaju rizik od pojave grešaka i gubitka informacija. Nejasnoće u praćenju poslovnih procesa, učinka, ciljeva, ograničeni analitički uvidi, predviđanja, propusti u personalizaciji usluga i upravljanju povratnim informacijama, otežavaju donošenje odluka, smanjuju konkurentnost, ograničavaju unapređenje strategija i negativno utiču na kvalitet usluge, zadovoljstvo i lojalnost klijenata.

Platforma *Salesforce*, sa naprednim mogućnostima prilagođavanja, integracije i primene veštačke inteligencije, predstavlja sveobuhvatno rešenje za navedene rizike. Svojom fleksibilnošću i širokim ekosistemom, omogućava preduzećima transformaciju i unapređenje poslovanja, uvodeći inovacije u mnogim industrijama.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Dunja Vrbaški, docent

2. PREGLED SISTEMA ZA UPRAVLJANJE ODNOSIMA SA KLIJENTIMA

Prepoznavanje potreba, želja, navika, sklonosti i očekivanja korisnika proizvoda ili usluga u mnoštvu sakupljenih informacija, postiže se upotrebom *CRM* sistema (eng. *Customer Relationship Management*), odnosno sistema za upravljanje odnosima sa trenutnim i potencijalnim klijentima.

Prema Batlu i Maklanu (2015), *CRM* sistemi su osnovna poslovna strategija koja integriše interne procese i funkcije, kao i spoljne mreže, kako bi stvorila i isporučila vrednost uz ostvarivanje profita. Zasnivaju se na visokokvalitetnim podacima prikupljenih informacionim tehnologijama [1,2].

Lehtinen ističe (2007) da je svrha *CRM* sistema upoznavanje strateških klijenata i ostvarivanje dugoročnih odnosa sa njima, umesto usmerenja na povećanje kratkoročnih prihoda [2].

Prema Pejnu (2007), *CRM* sistemi se definišu kao strateški procesi koji nastoje razvijanju odnosa, donoseći veću vrednost za zainteresovane strane (eng. *stakeholders*) [2].

Hlebovski navodi (2005) da su *CRM* sistemi interaktivni procesi koji teže da postignu optimalnu ravnotežu između kompanijskog ulaganja i zadovoljstva klijenata, što zavisi od profita obe strane [2].

Kotler i Armstrong tumače (2004) *CRM* sisteme kao specifične softverske programe i ujedno analitičku tehniku koja služi za integraciju i korišćenje velikih baza podataka o pojedinačnim klijentima [2].

Prema Čenu i Popoviču (2003) *CRM* sistemi su poslovne strategije koje integrišu ljude, procese i tehnologiju kako bi se ostvarilo razumevanje klijenata. Navode da će kompanije koje uspešno implementiraju *CRM* uživati u nagradama kroz lojalnost i dugoročnu profitabilnost. Iako tehnologija predstavlja ključni element u implementaciji, gledanje *CRM* sistema isključivo kao tehnološko rešenje nije dovoljno. Neophodno je povezati tehnologiju sa odgovarajućim poslovnim procesima i angažmanom ljudi [3, 4].

Objedinjenjem raznih departmana, kao što su marketing, prodaja, usluge, društveni mediji i njihovom međusobnom razmenom podataka, stiče se jedinstveni, jasni i detaljni uvid, kao i bolji odnos i spoznaja korisničkog iskustva. Centralizacija podataka u *CRM* sistemima doprinosi unapređenju organizacije kompanije i smanjenju administrativnih poslova.

Integracijom sa alatima veštačke inteligencije (eng. *Artificial Intelligence, AI*), moguća je personalizacija i predikcija interakcija klijenata, automatizacija procesa i pružanje uvida koji pomažu u donošenju procena. Takođe, postoji mogućnost integracije sa raznim poslovnim alatima, kao što su alati za potpisivanje dokumenata, računovodstvo, fakturisanje, ankete i sticanje sveobuhvatnog pogleda na klijente i upotpunjenje poslovanja.

Današnji *CRM* sistemi implementirani su na platformama zasnovanim na računarstvu u oblaku (eng. *cloud*), što omogućuje čuvanje i pristupanje podacima o klijentima bilo kada i sa bilo koje lokacije. Podrazumeva laku implementaciju, bez neophodnog hardvera ili problema oko praćenja verzija i ažuriranja. Pored obezbeđene sigurnosti, plaćanje se izvršava po korišćenim funkcionalnostima, sa prilikom za proširivanje u zavisnosti od potreba [5].

Na tržištu postoji veliki broj različitih *CRM* sistema, kao što su *Salesforce, Microsoft Dynamics 365*, *SAP, Monday.com, Zoho, HubSpot, Pipedrive* i drugi.

Koriste se u različite svrhe i poseduju određene prednosti i mane, te je razumevanje njihovih karakteristika ključno za odabir pogodnog *CRM* rešenja i uspešnost poslovanja.

3. PLATFORMA SALESFORCE

Na vrhu globalnog tržišta *CRM* sistema nalazi se platforma *Salesforce*, menjajući način poslovanja kompanija i upravljanja odnosima. Često se pogrešno povezuje isključivo sa prodajom i kompanijama koje se bave trgovinom.

Platforma *Salesforce* prevazilazi tradicionalna rešenja, nudeći čitav ekosistem aplikacija i servisa dostupnih na *cloud* okruženju, koji vode, optimizuju i unapređuju poslovne procese kroz domene prodaje, usluga, marketinga i mnogih drugih oblasti [6].

Koristi širok spektar tehnologija za razvoj, od specifičnih programskih jezika i alata do naprednih *cloud* tehnologija i veštačke inteligencije.

Objektno orijentisani programski jezik, sličan *Java* programskom jeziku, a specifičan za *Salesforce* platformu, naziva se *Apex.* Koristi se za pisanje kontrolera, okidača (eng. *triggers*), kao i za kreiranje kompleksnih poslovnih logika na server strani [7].

Okvir (eng. *framework*) *Lightning Components* koristi se za izgradnju dinamičkih veb aplikacija unutar *Salesforce* platforme. Dok je *Aura Components* stariji okvir za razvoj korisničkog interfejsa, *Lightning Web Components* (*LWC*) je moderniji, baziran na veb standardima kao što su *JavaScript* i *HTML* [8].

Platforma koristi internu bazu podataka zasnovanu na relacionom modelu, izgrađenu na osnovu *Oracle* baze podataka. Iako je *Oracle* osnova za čuvanje podataka, korisnici platforme nemaju direktan pristup bazi. Umesto toga, *Salesforce* nudi *API* pozive i prilagođene jezike poput *SOQL* (eng. *Salesforce Object Query Language*) i *SOSL* (eng. *Salesforce Object Search Language*) za upite i upravljanje podacima. Platforma koristi *multi-tenant* pristup, gde se podaci više klijenata nalaze u istom fizičkom okruženju, ali su virtuelno odvojeni. Na taj način podaci se čuvaju efikasno, dok svaki korisnik stiče utisak da raspolaže sopstvenom bazom podataka. Podaci se čuvaju u tabelama koje se nazivaju objekti. Standardni objekti (kao što su *Account, Contact, Opportunity*) dolaze uz platformu, dok korisnici mogu kreirati i prilagođene objekte (eng. *custom objects*) za specifične poslovne potrebe [9].

Vodeći proizvodi *Salesforce* platforme su *Sales Cloud, Service Cloud, Marketing Cloud, Data Cloud, Commerce Cloud, Experience Cloud, Slack, Tableau, MuleSoft, Heroku, Einstein AI, Small Business, Net Zero, Partners* i *Success* [10].

Odabir pogodnog *Salesforce Cloud* rešenja predstavlja složen proces i izazov, s obzirom na širok spektar dostupnih opcija. Različiti faktori utiču na donošenje odluke, uključujući poslovne potrebe, specifičnosti industrije, veličinu i složenost organizacije, kao i tehničke zahteve za prilagođavanjem i integracijom. Najpre je neophodno identifikovati osnovne potrebe poslovanja. Na primer, kompanije koje teže poboljšanju prodaje, korisničke podrške ili automatizaciji marketinga, mogu se opredeliti za rešenja koja odgovaraju tim prioritetima. Uz to, specifičnost industrije dodatno usmerava izbor i precizniju podršku poslovnim procesima. Takođe, veličina organizacije i složenost njenih operacija imaju uticaja. Manjim preduzećima često je potrebna jednostavnija platforma, dok veće kompanije mogu zahtevati kompleksnija rešenja sa naprednim funkcionalnostima. Potreba za prilagođavanjem i integracijom sa postojećim sistemima predstavlja dodatni faktor koji mora biti uzet u obzir. Ovim pristupom dolazi se do efikasnog izbora rešenja, uzimajući u obzir specifične potrebe i dugoročne ciljeve organizacije [11]. Na slici 1 predstavljeni su vodeći proizvodi, kao i čitav *Salesforce* ekosistem.

Slika 1. *Prikaz ekosistema Salesforce platforme [12]*

Skup specijalizovanih rešenja unutar *Salesforce* platforme, prilagođenih specifičnim industrijama, nazivaju se *Salesforce Industries.* Ova rešenja razvijena su kako bi se zadovoljile potrebe različitih sektora, nudeći unapred konfigurisane funkcionalnosti koje omogućavaju bržu implementaciju, optimizaciju poslovnih procesa i veću efikasnost. Izdvajaju se *Automotive, Communications, Education, Financial Services, Healthcare & Life Sciences, Retail, Nonprofit, Manufacturing, Government, Media,* ali i mnoga druga [13].

4. INTEGRACIJA VEŠTAČKE INTELIGENCIJE U SALESFORCE PLATFORMU

Pouzdana i proširiva veštačka inteligencija, doboko je integrisana u strukturu *Salesforce* platforme kroz pojam *Salesforce AI.*

4.1. Einstein AI

Proizvod, odnosno set ugrađenih *AI* funkcija, koji omogućava personalizovana, prediktivna iskustva i generisanje sadržaja, naziva se *Salesforce Einstein.* Primenjuje konverzacioni korisnički interfejs u okviru svake aplikacije ili radnog procesa i izgrađen je na *Einstein Trust Layer* sloju, sa značajnim zaštitnim mehanizmima. Ističu se *Einstein Prediction Builder,* koji omogućava kreiranje prediktivnih modela bez potrebe za programiranjem, *Einstein Bots* koji koriste veštačku inteligenciju za automatizaciju komunikacije sa korisnicima putem *chatbot* alata, dok *Einstein Discovery* analizira velike skupove podataka i pruža relevantne uvide. Koristeći generativnu veštačku inteligenciju, *Einstein GPT,* kreira personalizovane odgovore i upite u realnom vremenu, unapređujući interakciju sa klijentima i poslovnim korisnicima [14].

Tehnologija *Einstein AI* se može posmatrati kao platforma, jer je njen osnovni deo integrisan u svaki *Salesforce* proizvod. Takođe, prilikom korišćenja *Salesforce* platforme, brojne *Einstein* aplikacije se mogu koristiti kao dodaci koje je potrebno povezati ili prilagoditi [15].

Dakle, *Salesforce Einstein* pruža razne prednosti za organizacije, uključujući povećanje operativne efikasnosti kroz automatizaciju zadataka, optimizaciju procesa i brže donošenje odluka. Omogućava prilagođavanje korisničkog iskustva, unapređenje saradnje među timovima, bolju segmentaciju kupaca i doprinosi njihovom zadržavanju. Kontinuirano uči iz novih podataka, predstavljajući ključni alat za povećanje produktivnosti i konkurentnosti u poslovnom okruženju [16, 17].

4.2. Agentforce i AI agenti

Napredna kolekcija autonomnih *AI* agenata i alata, *Agentforce* (ranije poznato kao *Einstein Copilot*), u potpunosti su integrisani u *Salesforce* platformu i namenjeni za složene interakcije sa klijentima [18].

Dolaze sa unapred pripremljenim šablonima, na osnovu kojih je moguće brzo razvijanje i prilagođavanje, pružajući podršku zaposlenima i korisnicima, unapređujući radne procese za bilo koju ulogu, industriju i poslovni izazov.

Podešavanjem parametara, definisanjem tema, akcija i specifičnih uputa, optimizuje se rad agenata u poslovnim procesima. Inteligentni i proaktivni, u potpunosti su sposobni da razumeju i odgovore na upite korisnika bez ljudske intervencije. Osnovani su na mašinskom učenju i obradi prirodnog jezika za obavljanje širokog spektra zadataka, od odgovaranja na jednostavna pitanja do rešavanja složenih problema ili istovremeno obavljanje više zaduženja.

Izvršavanje *AI* agenata se odvija kroz četiri faze. Prva faza predstavlja prikupljanje podataka iz različitih izvora u realnom vremenu. Druga faza je donošenje odluka koristeći napredne modele mašinskog učenja. Treća faza je izvršavanje radnje, što se može odnositi na odgovaranje na korisnički upit, obradu zahteva ili prosleđivanje složenog problema ljudskom agentu. Poslednja, četvrta faza je učenje i prilagođavanje svakom interakcijom, usavršavajući algoritme radi poboljšanja tačnosti i efikasnosti. Sposobnost kontinuiranog učenja omogućava *AI* agentima da ostanu efikasni i relevantni, čak i kada se očekivanja korisnika i poslovna okruženja menjaju. Istovremeno mogu obrađivati više korisničkih interakcija, smanjujući vreme odgovora i povećavajući efikasnost korisničke podrške. Dostupni su u bilo koje vreme, bez uticaja na vremensku zonu ili radno vreme, nude mogućnost skalabilnosti, konzistentnu i pouzdanu podršku, kao i personalizovane interakcije.

Sa *Agentforce* platformom, unapred definisani agenti se mogu brzo implementirati u brojnim oblastima, kao što su prodaja, marketing, usluge i slično.

Agent *Service Agent* zamenjuje tradicionalne *chatbot* alate rešavajući širok spektar pitanja u vezi sa domenom korisničke podrške, bez potrebe za unapred programiranim scenarijima. Agent *Sales Development Representative* omogućava kontinuiranu interakciju sa potencijalnim kupcima, odgovara na njihova pitanja, upravlja prigovorima i zakazuje sastanke koristeći podatke iz dostupnih izvora, pružajući prodajnim timovima priliku da se posvete građenju dubljih odnosa sa klijentima. Agent *Sales Coach* nudi personalizovane sesije za vežbanje prodajnih veština prodajnim timovima, uz prilagođene govore i odgovore, koristeći *Salesforce* podatke i generativnu veštačku inteligenciju. Agent *Merchandiser* pomaže menadžerima u oblasti e-trgovine u postavljanju sajta, definisanju ciljeva, promocijama, kreiranju opisa proizvoda i pružanju uvida zasnovanih na analizi podataka. Agent *Buyer Agent* unapređuje B2B kupovine tako što pomaže kupcima da pronađu proizvode, obave kupovinu i prate narudžbine putem poruka ili prodajnih portala. Agent *Personal Shopper* funkcioniše kao digitalni savetnik na sajtovima e-trgovine ili aplikacijama za razmenu poruka, nudeći specifične preporuke proizvoda i pomoć prilikom pretrage. Agent *Campaign Optimizer* automatizuje ceo životni ciklus kampanja koristeći veštačku inteligenciju za analizu, generisanje, prilagođavanje i optimizaciju marketinških kampanja prema poslovnim ciljevima.

Platforma za izgradnju i prilagođavanje *AI* agenata naziva se *Agent Builder.* Olakšava konfiguraciju definisanih agenata ili izgradnju novih za bilo koju ulogu, industriju ili poslovni slučaj, koristeći alate kao što su *Flows, Prompts, Apex* i *MuleSoft API* [19, 20, 21, 22].

Kroz studije slučaja u radu, analizirani su izazovi i problemi sa kojima su se susreli luksuzno odmaralište *Turtle Bay Resort* i svetski šampionat *Formula 1,* postignuti rezultati i unapređenje korisničkog iskustva korišćenjem *Salesforce AI* tehnologije [23, 24].

5. ZAKLJUČAK

U ovoj studiji istraživan je širok spektar mogućnosti, primene i uticaja platforme *Salesforce,* sa posebnim naglaskom na integraciju veštačke inteligencije, koja predstavlja značajni iskorak u unapređenju savremenog poslovanja i transformaciji poslovnih procesa.

Korišćenjem navedenih tehnologija unutar *Salesforce* platforme, postiže se automatizacija i optimizacija zadataka, što doprinosi povećanju efikasnosti i profitabilnosti. Omogućava se brže i preciznije donošenje odluka, uočavanje obrazaca i pravljenje predikcija.

Praćenjem prethodnih interakcija, postiže se personalizacija, unapređujući korisničko zadovoljstvo i produktivnost. Takođe, generisanje sadržaja u realnom vremenu poboljšava celokupno iskustvo. U skladu sa priznatim industrijskim standardima, može značajno uticati na proces razvoja softverskih rešenja, podižući nivo ekspertize programera i kvalitet koda. Smanjuje se rizik od pojave grešaka, povećava efikasnost rada, otkrivaju bezbednosne ranjivosti u ranoj fazi razvoja, kao i temeljnije razumevanje složenih delova koda, rezultirajući bržim i pouzdanim napredovanjem. Neprekidnim učenjem iz novih podataka postiže se kontinuirano poboljšanje i prilagođavanje, čime se održava relevantnost i konkurentnost na tržištu.

S obzirom na brz napredak veštačke inteligencije, pred *Salesforce* platformom stoji ogroman potencijal za dalji razvoj i primenu, koji će oblikovati budućnost poslovnih procesa i strateških odluka i podići ih na potpuno novi nivo.

6. LITERATURA

[1] F. Buttle i S. Maklan, "Customer Relationship Management: Concepts and Technologies", ResearchGate, 2015, researchgate.net/publication/290447911_Customer_Relationship_Management_Concepts_and_Technologies, pristupano: decembar 2024.

[2] A. Chromčáková i H. Starzyczná, "Customer Relationship Management in Small and Medium-Sized Enterprises of the Moravian-Silesian Region: Qualitative Research", 2019, aak.slu.cz/pdfs/aak/2019/02/04.pdf, pristupano: decembar 2024.

[3] I. Chen i K. Popovich, "Understanding Customer Relationship Management (CRM): People, Process and Technology", Emerald Insight, 2003, emerald.com/insight/content/doi/10.1108/14637150310496758, pristupano: decembar 2024.

[4] H. Gil-Gomez, V. Guerola-Navarro, R. Oltra-Badenes i J. A. Lozano-Quilis, "Customer Relationship Management: Digital Transformation and Sustainable Business Model Innovation", Tandfonline, 2020, tandfonline.com/doi/full/10.1080/1331677X.2019.1676283, pristupano: decembar 2024.

[5] Salesforce, "CRM: What Is CRM (Customer Relationship Management)?", salesforce.com/eu/crm/what-is-crm, pristupano: septembar 2024.

[6] P. Pathak, "Research Paper on Salesforce Technology", 2024, ijarsct.co.in/Paper15215.pdf, pristupano: decembar 2024.

[7] Salesforce Developers, "Developers: Apex Developer Guide", developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode, pristupano: oktobar 2024.

[8] Salesforce Developers, "Developers: Get Started with Lightning Web Components", developer.salesforce.com/docs/platform/lwc/guide, pristupano: oktobar 2024.

[9] Integrate.io, "The Salesforce Database Explained", 2023, integrate.io/blog/the-salesforce-database-explained, pristupano: oktobar 2024.

[10] Salesforce, "All Salesforce products. One integrated platform.", salesforce.com/products, pristupano: oktobar 2024.

[11] Synebo, "How to Choose the Right Salesforce Cloud?", synebo.io/blog/15-types-of-salesforce-clouds, pristupano: oktobar 2024.

[12] Salesforce, "Salesforce Platform: Explore the Salesforce Platform", salesforce.com/platform, pristupano: septembar 2024.

[13] Salesforce, "Industries", salesforce.com/industries, pristupano: septembar 2024.

[14] Salesforce, "Arificial Intelligence: Salesforce Artificial Intelligence", salesforce.com/eu/artificial-intelligence, pristupano: oktobar 2024.

[15] N. Saini i H. Sharma, "Salesforce Einstein: Artificial Intelligence for Customer Success Platform", 2020, ijsret.com/wp-content/uploads/2020/06/IJSRET_V6_issue3_402.pdf, pristupano: januar 2025.

[16] Salesforce, "Artificial Intelligence: Salesforce Artificial Intelligence", salesforce.com/artificial-intelligence, pristupano: oktobar 2024.

[17] Atrium, "What is Salesforce Einstein? Your 2024 Guide to Einstein AI Products and Capabilities", atrium.ai/resources/what-is-salesforce-einstein-your-2024-guide-to-einstein-ai-products-and-capabilities, pristupano: oktobar 2024.

[18] Salesforce, "Artificial Intelligence: Copilot is now Agentforce", salesforce.com/artificial-intelligence/einstein-ai-assistant, pristupano: oktobar 2024.

[19] Salesforce, "Agentforce: What Are AI Agents? Benefits, Examples, Types", salesforce.com/agentforce/what-are-ai-agents, pristupano: oktobar 2024.

[20] Salesforce. "Frequently asked questions", salesforce.com/products, pristupano: oktobar 2024.

[21] Salesforce, "News & Insights: Salesforce Unveils Agentforce - What AI Was Meant to Be", salesforce.com/news/press-releases/2024/09/12/agentforce-announcement, pristupano: oktobar 2024.

[22] Salesforce, "Agentforce", salesforce.com/agentforce, pristupano: oktobar 2024.

[23] Salesforce, "Turtle Bay Resort elevates hospitality with AI-driven personalization", salesforce.com/customer-stories/turtle-bay-delivers-better-experiences-AI, pristupano: avgust 2025.

[24] Salesforce, "Agentforce will help Formula 1 speed up service response by 80%", salesforce.com/customer-stories/formula-one, pristupano: avgust 2025.

Kratka biografija:

Marija Janković rođena je u Novom Sadu 1999. godine. Osnovne akademske studije završila je 2023. godine na Fakultetu tehničkih nauka u Novom Sadu. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Elektronsko poslovanje odbranila je 2025. godine. kontakt: jankovicmarija15@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

УДК: 4.3
ДОИ: https://doi.org/10.24867/33BE26Zubovic

КОРИШЋЕЊЕ ТЕСТИРАЊА НА ЦРНОЈ КУТИЈИ ЗА ПОКРЕТАЊЕ АУТОМАТСКИХ ТЕСТОВА НА ТВ ПРИЈЕМНИКУ

USING BLACK BOX TESTING TO RUN AUTOMATED TESTS ON A TV BOX

Јован Зубовић, *Факултет техничких наука, Нови Сад*

Област: РАЧУНАРСКЕ ТЕХНИКЕ И РАЧУНАРСКА КОМУНИКАЦИЈА

Кратак садржај – Аутоматизација процеса тестирања који би иначе био одрађен ручно може да уштеди много времена и радних сати, посебно ако се аутоматски тестови могу покретати преко ноћи. Ако се омогући генерисање извештаја, на пример Ексел табеле, тест инжењер треба само да их провери ујутру и проследи програмерима и вођама тимова. Посао постаје још бржи и ефикаснији ако постоји радни оквир (енг. framework) који пруже универзалне функције за руковање најчешће тестираних функционалности, као што су обрада слике и звука или комуникациони протоколи. Тај радни оквир се може прилагодити за различите уређаје и апликације, које су развијали потпуно различити тимови и компаније. Овај рад садржи опис прилагођавања постојећег радног оквира за аутоматизацију процеса тестирања на тестираном ТВ пријемнику, што укључује тестирање стабилности и функционалности постојећих апликација и самог уређаја.

Кључне речи: Системско тестирање, Аутоматско тестирање, Потрошачка електроника, тест извештај

Abstract - *Automating the process of testing which would otherwise be done manually can save great amounts of time and working hours, especially given that automated tests can be run during night hours. If generating files that summarize test reports is included, for example, an Excel table, the test engineer only needs to check them out in the morning and forward them to developers and team leads. This job becomes even faster and more efficient if there is a a framework that provides universal functions for handling the most tested functionalities, such as sound and image processing, or communication protocols. That framework can be adjusted for various applications and devices, developed by completely different teams and companies. This paper contains a description of adapting the existing framework for automation of the test process on the tested TV box, which includes testing of stability and functionality of existing apps and the device itself.*

Keywords: *System testing, Automatic testing, Consumer electronics, test report*

1. УВОД

Свака апликација мора бити детаљно истестирана пре него што се пошаље муштерији. Овај процес се може одрадити ручно, где ће одабран тим људи узети даљински, тастатуру, палицу за игру (енг. joystick) или неки други контролер за ту специфицичну сврху и прећи разне тестне сценарије један по један. То мора да се уради више пута да би се осигурало да је свака могућност функционална и стабилна.

Нажалост, овај процес је далеко од оптималног [1]. Он захтева доста људи којима треба пуно радних сати. Не само то, него тестер након тога мора да сакупи све резултате и направи извештај, што одузима додатно време [2].

Због тога, аутоматизација тестног процеса је веома ефикасна при смањивању људског фактора, а уз то се процес сада може обавити преко ноћи. Такође, многи програмски језици нуде разне библиотеке уз које се могу генерисати сви потребни извештаји, на пример у Мајкрософт Ексел табели. Могу се додати и датотеке са детаљнијим праћењем параметара као што су искоришћеност RAM меморије, CPU-а, динамичке меморије (енг. heap) и величине виртуелне меморије.

Даље унапређење би била имплементација радног оквира у ком би биле сакупљене функције за проверу најчешће тестираних функционалности, као што су присутност и квалитет звука и видеа, комуникације са циљним уређајем, генерисањем извештаја. У овом раду фокус ће бити на прилагођењу постојећег радног оквира за аутоматизацију тестног процеса на ТВ пријемнику под тестирањем, што укључује тестирање системских и корисничких апликација, како и понашања самог уређаја.

Остатак рада је структуиран на следећи начин: У поглављу 2 су описани изазови са којим се суочавамо при прилагођавању постојећег радног оквира за за специфичну примену. У поглављу 3 ће бити описано предложено решење које ће садржати све аспекте апликације или уређаја које треба покрити. У поглављу 4 ће бити приказани резултати и начин на који треба да се тумаче. Коначно, у поглављу 5 ће бити кратак закључак целог рада.

2. ИЗАЗОВИ ПРИ ПРИЛАГОЂАВАЊУ ПОСТОЈЕЋЕГ РАДНОГ ОКВИРА СПЕЦИФИЧНОМ ПРОЈЕКТУ

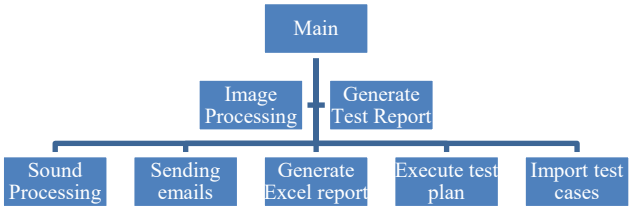
Постојећи радни оквир је развијен на РТ-РК Институту и успешно примењен на многим

НАПОМЕНА:
Овај рад проистекао је из мастер рада чио ментор је био др Илија Башичевић, ред. проф.

пројектима. Иако радни оквир пружа универзалне функције за за често тестиране функционалности, сваки пројекат захтева прилагођење параметара функција за специфичну употребу у том пројекту [3]. Унутар пројекта, параметри се неће мењати сваки пут. Зато би било корисно написати посебан модел у ком ће се налазити модификоване функције из радног оквира са параметрима специфичним за тај пројекат. Тај модул ће бити укључен у сваки тест и тако ће прегледност и будуће одржавање кода бити много лакше.

У случају ТВ пријемника под тестирањем, требаће нам функције за проверу присутности звука и видеа, читање тренутне позиције и укупне дужине садржаја са екрана, обрада слике за препознавање прогрес бара, података о догађајима, контрола на екрану и водича.

Што се тиче самих тестова, итерација не треба да се настави ако падне провера било којег од тестираних аспеката. У супротном, време ће се непотребно потрошити на тестирања неких других аспеката за итерацију која је свеједно пала. Присуство звука и видеа треба да буде проверено прво јер нема смисла тестирати ТВ апликације или пријемника ако нема слике или звука. Добра идеја би била да се то тестира два пута по итерацији, због сигурности.



Слика 1. *Дијаграм који приказује различите модуле присутне у радном оквиру*

3. ПРЕДЛОЖЕНО РЕШЕЊЕ

У овом решењу, све функције су груписане у базном модулу за уређај под тестирањем.

Први корак је био сакупљање базе снимака екрана на којима се виде сви подаци које ћемо тражити. Координате ових података се даље користе као референтне у даљим тестовима. Тестови који садрже проверу звука немају базу података јер звук варира превише од снимка до снимка. Највероватније је да ће се тражити прогрес бар, почетно и крајње време неког догађаја, име догађаја, број канала, тренутну позицију и неке посебне команде које се приказују на екрану. Када су одређене координате свих елемената који ће се претраживати, написали смо функције за њихово препознавање.

Други корак је проучавање лог датотеке да бисмо нашли који текст треба да претражујемо да би се добавиле тренутна позиција, број канала, имена активности или било који други аспект који нас занима. Када смо одредили шта тачно тражимо, написали смо функције које ће претраживати тај текст у тестовима.

Такође су нам требале функције за претраживање одређених активности, детектовање притиснутих дугмића на даљинском управљачу и потврде да је апликација успешно покренута.

Након тога смо додали неколико функција за праћење понашања апликације и уређаја, јер желимо да зауставимо тестирање и поново покренемо уређај ако нешто критично крене по злу. Ове функције ће бити корисне и за генерисање коначног тест извештаја. Оне укључују праћење попуњености RAM меморије, CPU-а, динамичке меморије и величине виртуелне меморије. Радни оквир већ садржи функције за генерисање извештаја, па није потребно додавати нове. Такође садржи опцију за пуштање веће количине тестова одједном, који се могу пустити и преко ноћи. На тест инжењеру је да процени колико тестова се може извршити преко ноћи. Није проблем ако се пусти превише тестова јер резултати готових тестова могу да се прегледају док се чекају преостали. Важније је да се не пусти премало тестова, јер се онда непотребно губи време. Док се прегледају резултати извршених тестова, нова група тестова може бити пуштена у позадини, ако су сви већ пуштени тестови завршени.

Још једно корисно унапређење већ присутно у радном оквиру је могућност слања електронске поште (енг. E-mail) са свим потребним датотекама свима које би та електронска пошта интересовала. Менаџмент углавном жели само табелу са коначним резултатима, док програмери вероватно желе детаљнији извештај, па се мејл треба прилагодити на основу тога. Узевши све горе наведено у обзир, Пајтон је одабран као најбољи језик за имплементацију јер већ садржи велики број библиотека и документације. Даље, радни оквир је идеалан за тестирање на црној кутији-тестирање у ком није потребно знати тачно шта апликација ради, већ само шта да се очекује од ње, што је идеално за тестирање пројеката који нису отвореног кода.

У овом радном оквиру практично нема корисничког интерфејса, већ се све ради преко терминала. Тестови и радни оквир треба да буду на истој локацији, и када се радни оквир покрене, опције ће се приказати у терминалу. Може се бирати између више група тестова, које се могу прилагодити по жељи, а онда се може бирати да ли ће бити пуштен један или сви тестови из групе. За пуштање преко ноћи, треба пустити целу групу, а за испитивање појединачног теста, само он треба да буде пуштен.

4. РЕЗУЛТАТИ

Резултати су сви сакупљени у једном фолдеру и садрже извештаје о успешности теста, праћеним параметрима, лог датотеке, као и аудио и видео снимке из тренутака када се десило нешто што није очекивано. Датотека која садржи сумаран преглед свега праћеног тесту је такође присутан у виду ексел табеле.

Тест инжењер треба да буде обучен да чита лог датотеке за пројекат или уређај у питању, јер је углавном програмер тај који ставља логове зарад испитивања [4]. Ово је од велике помоћи јер

драстично скраћује време које програмеру треба да пронађе шта се десило у случају грешке.

```
1. Tests_Functional
2. Tests_Functional_NO_STREAM
3. Tests_LongRun_ONE_HOUR
4. Tests_Stability
5. Tests_Stability_NO_STREAM

Select type of tests:
>>> 3
Tests:
" 1. LongRun_DASH_Live_without_subtitles"
" 2. LongRun_DASH_Live_with_subtitles"
" 3. LongRun_DASH_VoD_without_subtitles"
" 4. LongRun_DASH_VoD_with_subtitles"
" 5. LongRun_encrypted_DASH_VoD_without_subtitles"
" 6. LongRun_encrypted_MSS_VoD_without_subtitles"
" 7. LongRun_HLS_Live_without_subtitles"
" 8. LongRun_HLS_VoD_without_subtitles"
" 9. LongRun_HLS_VoD_with_subtitles"
" 10. LongRun_MSS_VoD_without_subtitles"
11. All tests
0. Exit
Choose test:
>>> 1
```

Слика 2. *Терминал у ком тест инжењер може да бира између различитих група тестова, и да ли ће пустити један или све тестове из одабране групе*

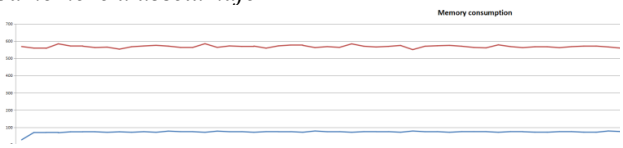
У следећем низу снимака екрана (слике 3-6) се не виде све генерисане датотеке, али се види довољно да може да се схвати колико је извештај детаљан.

```
Stop looping
Number of successful iterations/Total number of iterations: 10/10
************************************
Test result: PASS success rate: 1.0
Time of test execution: 0:02:10.316106
```

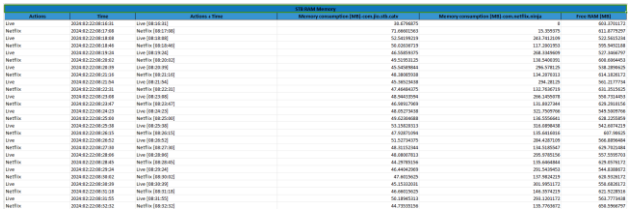
Слика 3. *Приказ тестних резултата*

Pictures	20-Feb-24 11:22	File folder	
Videos	20-Feb-24 11:25	File folder	
all_pids.txt	20-Feb-24 11:24	TXT File	5 KB
avc_denied.txt	20-Feb-24 11:25	TXT File	188 KB
bugreport-UHDHybridSTB-10.0.9-2024-0...	20-Feb-24 11:29	zip Archive	12,825 KB
com_jio_stb_catv_dumpsys_data.txt	20-Feb-24 11:24	TXT File	243 KB
CPU_IDLE_MEMORY.txt	20-Feb-24 11:29	TXT File	4 KB
CPU_LOAD_MEMORY.txt	20-Feb-24 11:29	TXT File	2 KB
cpu_temp.txt	20-Feb-24 10:11	TXT File	0 KB
DALVIK_HEAP_com_jio_stb_catv.txt	20-Feb-24 11:29	TXT File	4 KB
dumpsys_meminfo_oom.txt	20-Feb-24 11:25	TXT File	607 KB
extended_pid_info.txt	20-Feb-24 10:19	TXT File	1 KB
iwaptf_communication.log	20-Feb-24 11:25	Text Document	413 KB
JAVA_HEAP_com_jio_stb_catv.txt	20-Feb-24 11:29	TXT File	4 KB
measure_memory.xlsx	20-Feb-24 11:29	XLSX Worksheet	43 KB
Memory_consumption.txt	20-Feb-24 11:29	TXT File	4 KB
MEMORY_Free_RAM.txt	20-Feb-24 11:29	TXT File	4 KB
NATIVE_HEAP_com_jio_stb_catv.txt	20-Feb-24 11:29	TXT File	4 KB
pid_info.txt	20-Feb-24 11:29	TXT File	1 KB
Stability_Navigate_within_app.log	20-Feb-24 11:25	Text Document	80,990 KB
Stability_Navigate_within_app_execution...	20-Feb-24 11:25	TXT File	45 KB
top_data.txt	20-Feb-24 11:24	TXT File	1,886 KB
VM_SIZE.txt	20-Feb-24 11:29	TXT File	4 KB

Слика 4. *Фолдер који садржи све генерисане датотеке и извештаје*



Слика 5. *График који показује искоришћеност RAM меморије током времена*



Слиак 6. *Табела која приказује искоришћеност RAM меморије током итерација уз временске ознаке*

5. ЗАКЉУЧАК

У овом раду су приказане различите адаптације постојећег Пајтон радног оквира који помаже при аутоматизацији целокупног тестног процеса за ТВ пријемник на тестирању. Са овим решењем, процес ће бити готов преко ноћи, а тест инжењера ће извештај чекати спреман ујутру, а електронска пошта са потребним датотекама ће бити послати. Сада програмери и тест инжењери могу да прочуе те датотеке и оптимизује пројекат много брже и ефикасније. Ако је процес успешно аутоматизован, ручно тестирање се може свести на истраживачко играње са апликацијом или уређајем, да би се одређени ивични случајеви такође покрили.

6. ЛИТЕРАТУРА

[1] D. Marijan, V. Zlokolica, N. Teslic, V. Pekovic and T. Tekcan, "Automatic functional TV set failure detection system," in ***IEEE Transactions on Consumer Electronics***, vol. 56, no. 1, pp. 125-133, February 2010, doi: 10.1109/TCE.2010.5439135

[2] M. Katona, I. Kastelan, V. Pekovic, N. Teslic and T. Tekcan, "Automatic black box testing of television systems on the final production line," in ***IEEE Transactions on Consumer Electronics***, vol. 57, no. 1, pp. 224-231, February 2011, doi: 10.1109/TCE.2011.5735506

[3] Divya Kumar, K.K. Mishra, The Impacts of Test Automation on Software's Cost, Quality and Time to Market, Procedia Computer Science, Volume 79, 2016, Pages 8-15, doi: 10.1016/j.procs.2016.03.003

[4] Nagabushanam, Durga & Dharinya, Sree & Vijayasree, Dasari & Sai Roopa, Nadendla & Arun, Anugu. (2022). A Review on the Process of Automated Software Testing. 10.48550/arXiv.2209.03069.

Кратка биографија:

Јован Зубовић рођен је у Сомбору 2001. године. Основне студије на Факултету техничких наука из области Рачунарство и аутоматика – Рачунарска техника и рачунарске комуникације завршио је 2023. године.
Контакт: jovanzubovic2001@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/33BE27Janosevic

JEZIK SPECIFIČAN ZA DOMEN MAKROA ZA TASTATURE

KEYBOARD MACRO DOMAIN SPECIFIC LANGUAGE

Dušan Janošević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je predstavljena implementacija jezika specifičnog za domen (DSL) makroa za tastature, kao i analiza domena makroa za tastature. Opisan je alat koji koristi ovaj domen i implementiran koristeći programski jezik Python i paket textX za parsiranje gramatike jezika specifičnog za domen, kao i za samo čitanje skripti napisanih datim jezikom specifičnim za domen. Glavni cilj projekta jeste napraviti jednostavni alat za pisanje makroa za tastature koji je intuitivan i zahteva niži nivo programerskog znanja.*

Ključne reči: *DSL, textX, Python, alat, makro*

Abstract – *The paper presents a domain-specific language (DSL) implementation of keyboard macros, as well as an analysis of the keyboard macros domain. A tool using this domain is described, implemented using the Python programming language and the textX package for parsing the grammar of a domain-specific language. The main goal of the project is to create a simple tool for writing macros for keyboards that is intuitive and requires a lower level of programming knowledge*

Keywords: *DSL, textX, Python, tool, macro*

1. UVOD

Makroi za tastaturu predstavljaju unapred definisane nizove komandi koji omogućavaju automatsko izvršavanje više radnji na računaru jednim pokretom ili prečicom. Iako su svi makroi prečice, nisu sve prečice makroi, razlikuju se po složenosti. Prečice obično aktiviraju jednu funkciju dok su makroi kompleksniji i aktiviraju više radnji [3].

U savremenom računarstvu makroi su postali neizostavan deo svakodnevnog rada na računaru, naročito kod naprednijih korisnika koji žele da optimizuju svoj rad i povećaju svoju produktivnost. Pored makroa na tastaturi često se koriste i makroi vezani za miš [3].

Bitan su aspect kod korisnika računara koji imaju ponavljajuće radnje a vremenom su se proširili i na video igre.

Uvođenjem kompleksnijih softverskih sistema raste i potreba za kompleksnijim makroima, posebno onima koje korisnici sami mogu definisati. Istraživanje sprovedeno na uzorku od 82 korisnika pokazalo je da postoji snažna veza između stepena svakodnevnog korišćenja računara i upotrebe makroa [3].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Igor Dejanović, red. prof.

2. ISTORIJSKI RAZVOJ

2.1. Formatiranje teksta

Prečice su nastale pre pojave računara, još na mehaničkim pisaćim mašinama. Prva značajna implementacija prečica bila je na Remingtonovoj pisaćoj mašini koja je koristila taster *Shift* za promenu veličine slova. U digitalnom obliku poznate su još i prečice za kopiranje, lepljenje i sečenje teksta [1].

Do 1980-ih prečice su postale standardna funkcionalnost. Uvedena je prečica *Ctrl+Alt+Del* 1981. Godine dok su *Alt+Tab* i *Undo* pojavili u kasnijim verzijama *Windows* operativnog Sistema

2.2. Makroi

Pojam makroa u računarskom smislu potiče iz 1960-ih kada su korišćeni u asemblerskim jezicima kao makro instrukcije koje smanjuju količinu izvorne programske logike. Tokom 1980-ih makroi se pojavljuju u alatima poput *SmartKey* i *ProKey*.

2.2.1. Period 1980-ih

Pojavljuju se prvi makroi u programima za tabelarne proračune, kao što je *lotus 1-2-3. Microsoft office* uvodi jednostavne makroe kroz alat *WordBasic*. Prvi program za snimanje pokreta miša i tastature zove se *AutoIt.*

2.2.2. Period 1990-ih

Microsoft pravi *VBA* alat za automatizaciju u *Office* programima. Makroi se počinju koristiti u video igrama gde jedno dugme može pokrenuti više radnji. Program *Macro Express* omogućava pravljenje makroa bez znanja programiranja.

2.2.3. Period 2000-ih

Alati kao što su *AutoHotKey* i *RPA* proširuju svoju primenu makroa na poslovne zadatke. Automatizacija polako prelazi iz licne upotrebe u poslovne tokove rada.

2.2.4. Period 2010-ih

Makroi se sele u *cloud* okruženja. *Selenium* omogućava automatizaciju *web* zadataka. Počinje integracija veštačke inteligencije za prepoznavanje elemenata i pokretanje makroa.

2.2.5 Period 2020-ih

Microsoft uvodi *Power Automate* u *Windows*. Makroi se široko primenjuju u industriji i uslugama, AI alati kao *ChatGPT* i *Copilot* pojednostavljuju kreiranje skripti. *VBA* i dalje ima snažnu ulogu a sigurnosni problemi su velikim delom rešeni [2, 4].

3. POSTOJEĆA REŠENJA

3.1. Macro Recorder

Komercijalni alat za snimanje i automatsko izvršavanje akcija mišem ili tastaturom. Razvila ga je kompanija *Bartels Media*. Ima vizuelni interfejs i podršku za *OpenAI* odnosno *ChatGPT*. Nije potrebno programersko predznanje [5].

3.2. Macro Express

Softverski alat još iz 1990-ih. Omogućava kreiranje složenih makroa bez programiranja. Imao je više verzija gde su glavne promene po verzijama bile: Verzija 3 je uvela uslovno izvršavanje, petlje i okidače makroa. Verzija 4 dobila je bolju podršku za *Windows* operativni sistem i lakše uređivanje makroa. Verzija 5 dobila je novi interfejs i 64-bitnu podršku, pored toga dodatno je optimizovana. Najnovija verzija, pod brojem 6, dobila je integraciju sa AI alatima i *cloud* servisima, poboljšanu stabilnost i plaćala se samo jednom. Poseduje probni period i portabilnu verziju alata [6].

3.3. AutoHotKey

Besplatan i *open-source* jezik za automatizaciju na *Windows* operativnom sistemu. Podržava kompleksne skripte za tastaturu, miš i rad sa prozorima i fajlovima. Jednostavan je za napredne korisnike ali može biti zahtevan novim korisnicima [7].

3.4. Pulover's Macro Creator

Besplatan alat baziran na *AutoHotKey* alatu ali sa vizuelnim interfejsom. Omogućava lako kreiranje makroa bez pisanja koda. Ograničen je na *Windows* platformu [8].

3.5. Windows Power Automate

Windows-ov alat za naprednu automatizaciju poslovni procesa. Radi i u *cloudu* i na *deskop*-u. Integrisan je u Windows operativni sistem, besplatan je i podržava RPA funkcionalnosti. Veća moć, ali i kompleksnost u odnosu na druge alate [9].

4. KORIŠĆENI ALATI

Opisani su alati korišćeni za izradu projekta I opisani su razlozi izbora. Projekat je rađen uz pomoć *Python* programskog jezika verzije 3.12, biblioteke koje su korišćene su *textX* za definisanje jezika i parsiranje koda, *pynput* kao alat za čitanje i aktivaciju tastera i *pyautogui* kao alat za dobijanje podataka sa ekrana radi aktivacije automatskih makroa.

4.1. Python

Predstavlja interpretirani, jednostavni i svestran jezik pogodan za automatizaciju, analizu podataka, mašinsko učenje i *web* razvoj. Ima veliku zajednicu, bogate biblioteke i pogodan je za brz razvoj softvera. Mana mu je što je sporiji od ostalih programskih jezika [10].

4.2. textX

Python biblioteka koja služi za kreiranje domenskih jezika i parsiranje teksta na osnovu definisane gramatike. Omogućava lako generisanje *Python* klasa iz gramatike i nudi jednostavan način za obradu korisnički definisanih jezika. Zasnovan je na PEG parseru što osigurava nedvosmislenost prilikom čitanja ulaza [11].

4.3. Pynput

Python biblioteka koja omogućava kontrolu i slušanje ulaza sa tastature i miša. Podržava sve glavne operativne sisteme i pogodna je za makroe jer omogućava simulaciju i presretanje tastera. Dobro radi u *multi-threading* okruženju ali ne podržava čitanje sadržaja sa ekrana i zahteva ručno mapiranje specijalnih tastera [12].

4.4 Pyautogui

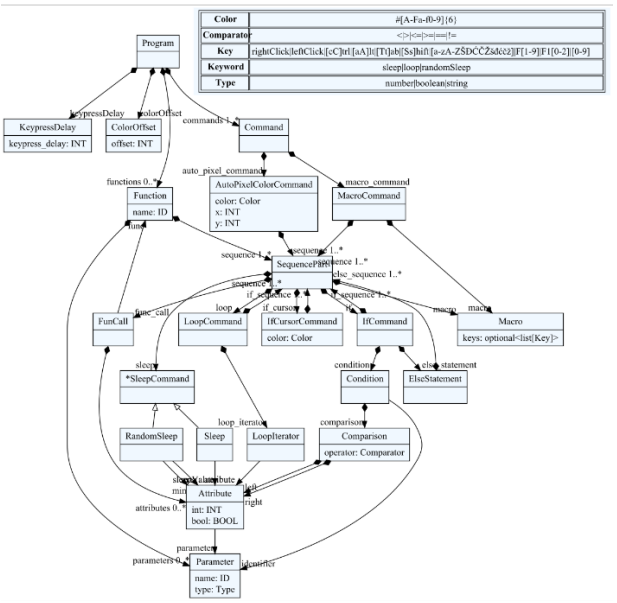
Biblioteka koja služi sa automatizaciju miša i tastature kao i za čitanje sadržaja sa ekrana. Podržava glavne operativne sisteme, ali može čitati sliku samo sa glavnog ekrana i ne može slušati ulaz sa tastature. Pogodna je za jednostavne automatizacije i simulacije unosa [13].

5. META MODEL MAKROA ZA TASTATURE

U ovom poglavlju opisan je domen makroa za tastature, sa naglaskom na pojmove koji čine osnovu za definisanje jezika specifičnog za domen. Centralni elementi su događaji koji pokreću makroe i akcije koje se iz tih događaja izvršavaju. Akcije mogu biti jednostavne ili kompleksne, uključujući petlje, uslovno izvršavanje, pauze i funkcije radi ponovne upotrebe koda. Svi ti elementi objedinjuju meta-model koji definiše koncepte, njihove veze i ograničenja.

Cilj je omogućiti korisniku da fleksibilno i apstraktno kreira makroe bez nepotrebnog ponavljanja.

5.1 Vizuelizacija meta-modela



Slika 1. *Meta-model jezika za makroe za tastature*

6. Interpretiranje

Najbitniji deo samog alata. Opisuje samo izvršavanje skripte. Pre nego što otpočne izvršavanje skripte, textX model se svodi na Python klase koje su ručno napisane za određenu upotrebu, odnosno ne koriste se Python klase koje generiše textX, samim tim je i gramatika određenog oblika.

Deo pokretanja pre samog interpretiranja se svodi na učitavanje meta-modela, zatim učitavanje određenih skripti, odabir skripte i svođenje te skripte na Python model. Nakon toga otpočinje interpretiranje skripte.

6.1 main.py

Glavni fajl projekta, sadrži *main()* funkciju od koje otpočinje izvršavanje. U njoj se prvo učitavaju sve makro skripte iz projektnog direktorijuma funkcijom *load_macros()*. Nakon čega se vrši provera cikličnosti pomoću klase *Checker*. Zatim se izabrana skripta šalje na izvršavanje kroz funkciju *interpret()*. Sam meta-model se učitava pre početka rada funkcija.

6.2. Definicija cli alata

Definiše drugi način pokretanja skripti preko komandne linije umesto pokretanja skripte kao Python projekta. Ulazna tačka je *cli.py* odakle se poziva definicija skripte iz fajla *type_macro.py*, koja učitava meta-model i definiše funkcije za učitavanje i izvršavanje makro skripte. Korisnik samo navodi putanju do željene skripte kao argument komandnog alata.

6.3. Checker klasa za proveru cikličnosti

Veoma bitna klasa u projektu. Služi za proveru cikličnosti poziva makroa, odnosno da li možemo ući u beskonačnu petlju poziva makroa. Ova pojava je potencijalno opasna jer može dovesti do gubitka kontrole nad računarom. Zato je u sam alat ugrađen bezbednosni makro za zaustavljanje svih akcija (*Alt+F12*).

Glavna funkcija je *detect_cycle()* koja primenjuje algoritam dubinske pretrage (DFS) nad stablom poziva. Za svaki makro proverava se da li aktivira drugi makro, i ako je tako taj odnos se prati rekurzivno. Ako tokom pretrage naiđemo na makro koji je već bio u putanji pretrage, otkrivena je cikličnost i algoritam vraća upozorenje.

Ako nijedan put ne odvede do takve situacije, makro se beleži kao bezbedan i isključuje se iz narednih provera. Ovakva provera obezbeđuje stabilnost sistema i sprečava slučajne beskonačne petlje u automatizaciji.

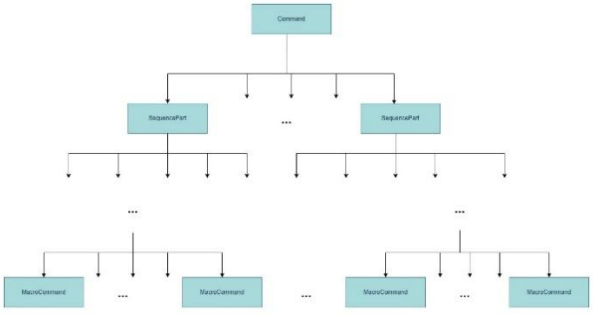
6.4. Python klase modela

Selekcija u kojoj se posebno opisuju sve klase domena u Python obliku, opisuju se njihovi atributi kao i funkcije koje poseduju date klase.

Nakon što se sam model učita preko klase *MacroGroup* koja suštinski poseduje celokupnu skriptu sa makroima, celokupna stabla koja predstavljaju makroe se svode na klasu *FullMacro* koja strukturu stabla svo ina listu akcija.

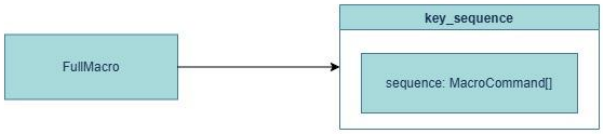
6.5. Raspetljavanje

Raspetljavanje je proces svođenja stabla akcija jednog makroa na listu akcija. Na početku modela, svaki pojedinačni makro poseduje stablo akcija koje obuhvata izvršavanje makroa. Ovaj oblik nije baš pogodan za izvršavanje i može se raspetljati zbog jednostavnosti. Na slici 2. prikazan je primer jednog stabla makroa.



Slika 2. *Primer stabla akcija jednog makroa*

Nakon što se izvrši raspetljavanje, dobiće se instanca klase *FullMacro* koja će u sebi sadržati raspetljanu listu akcija koje su se nalazile u stablu (Slika 3.).



Slika 3. *Primer raspetljane sekvence akcija preko klase FullMacro*

6.6. Interpreter

Sve počinje od funkcije *interpret()* koja pokreće slušanje sa tastature ili slušanje specifičnog piksela na ekranu u zavisnosti od tipa makroa, svako slušanje makroa poseduje svoju zasebnu nit. Ako je makro zasnovan na tasterima, koristi se funkcija *listen_to_macro()*, a ako reaguje na boju piksela poziva se funkcija *listen_to_pixel()*, niti koje su zasnovane ovim funkcijama zovemo nitima slušačima.

Dok se slušaju događaji, odvojena nit iz paketa *pynput* prati kada je taster pritisnut ili otpušten i u zavisnosti od toga puno odnosno prazni set pritisnutih tastera. Kada se ispune uslovi za pokretanje makroa, nit slušač će pokrenuti novu nit koja će izvršavati pritiske tastera definisane makroom koji je aktiviran preko funkcije *execute_macro_sequence()*.

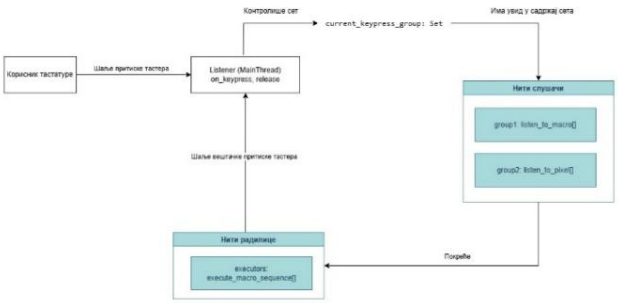
6.6.1. Multi-threading pojašnjeno

Do sada su spomenute niti koje su prisutne kod interpretera, ovde su definisani odnosu između njih kao i detaljniji opis tih niti.

Glavna nit prati pritiske tastera, ova nit potiče iz paketa *pynput*. Služi da ubacuje odnosno izbacuje tastere iz seta pritisnutih tastera.

Niti slušači su niti koje se pokreću u isto vreme kada i glavna nit. Ove niti prate sadržaj seta pritisnutih tastera i proveravaju da li je ispunjen uslov za pokretanje makro sekvence akcija.

Niti radilice su niti koje se instanciraju u nitima slušačima. Kada je uslov za aktivaciju makroa ispunjen, instancira se nova nit radilica koja će izvršavati akcije datog makroa, za to vreme, nit slušač nastavlja da sluša. Ovo znači da jedna nit slušač može da instancira više istih niti radilica.



Slika 4. *Primer modela funkcionisanja niti interpretera*

7. PRIMERI SKRIPTI DEFINISANI JEZIKOM

U ovom poglavlju opisani su konkretni primeri korišćenja skripti napisanih za korišćenje uz pomoć alata opisanog u ovom radu.

7.1. Otvaranje programa i puštanje muzike

Ova skripta sadrži više funkcija koje pomažu pri apstrakciji određenih sekvenci akcija i pomažu pri jednostavnije pisanju. Skripta poseduje makroe za otvaranje *Web* pretraživača Opera, makro za otvaranje programa za video igre *Steam* [14] kao i dve varijante prečica za puštanje muzike, jedna varijanta preko browsera, druga preko aplikacije za muziku *Spotify*.

Slika 5. *Pomoćne funkcije za skriptu otvaranja programa i pokretanja muzike*

```
Main {
    alt + o :sleep(300), command, sleep(100), openOpera();
    alt + s :sleep(300), command, sleep(100), openSteam();
    alt + l: sleep(300), command, sleep(100), playLoFi(false);
    alt + k: sleep(300), command, sleep(100), playLoFi(true);
};
```

Slika 6. *Makroi za pokretanje muzike i specifičnih programa uz pozivanje pomoćnih funkcija*

7.2. Skripta za video igru *Path of Exile*

Pokazuje pravu moć makroa z video igrama, sadrži dva makroa koja slušaju specifične pozicije piksela na ekranu. Ukoliko količina životnih poena ili količina magije heroja u igri padne ispod određenog nivoa, odnosno promeni se boja piksela bara, aktiviraće se specifičan makroa da taj bar napuni.

Pored toga sadrži i jednostavan makro za aktivaciju više magija preko prečice.

```
heal(): 1;
mana(): 5;

Main {
    onColor(#3e2627, 750, 1350,150) : heal();
    onColor(#16161d, 2690, 1350,150) : mana();
    alt + d : ctrl + q, ctrl + w, ctrl + e, ctrl + r;
};
```

Slika 7. *Primer skripte za slušanje barova za igru Path of Exile[16] za aktivaciju moći*

8. ZAKLJUČAK

Makroi predstavljaju unapred definisane sekvence akcija komandi koje značajno ubrzavaju rad na računaru. Njihov razvoj je prošao dug put. Od mehaničkih prečica do modernih alata koji mogu koristiti veštačku inteligenciju. Glavna prednost makroa je ušteda vremena, pritiskom nekoliko tastera mogu se pokrenuti složene radnje, što je posebno korisno u profesionalnom okruženju.

U radu je prikazano kako su alati za makroe evoluirali i kako se danas koriste u alatima poput *AutoHotKey*-a и *Macro Express*-a. Cilj projekta bio je da se napravi jednostavan, ali moćan alat prilagođen početnicima ali dovoljno fleksibilan za napredne korisnike. Fokus je stavljen na lakoću upotrebe, mada to ograničava rad na složenijim skriptama.

Projekat takođe uključuje proveru postojanja rekurzije među makroima kao i *multi-threading* mehanizam za paralelno izvršavanje, što omogućava bolje performanse. Primeri upotrebe pokazuju primenu u svakodnevnom radu i igrama.

Unapređenja ovog alata obuhvataju proširenje sa grafičkim interfejsom, kontrolu miša, kao i prebacivanje alata na neki drugi programski jezik zbog performansi. Takođe interesantno proširenje bila bi podrška za više računara, odnosno kontrola više računara sa jednog glavnog.

9. LITERATURA

[1] https://www.taskade.com/blog/history-keyboard-shortcuts-productivity, приступ 26. Фебруар 2025.

[2] https://en.wikipedia.org/wiki/Macro_(computer_science)

[3] Peres, S. C., Tamborello, F. P., Fleetwood, M. D., Chung, P., & Paige-Smith, D. L. (2004). *Keyboard Shortcut Usage: The Roles of Social Factors and Computer Experience. Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 48(5), 803–807.* https://journals.sagepub.com/doi/abs/10.1177/154193120404800513

[4] https://www.digit.in/features/general/digit-mag-the-origin-of-software-macros-53206.html, приступ 1. Март 2025.

[5] https://www.macrorecorder.com , приступ 1. Март 2025.

[6] https://www.macros.com , приступ 1. Март 2025.

[7] https://www.autohotkey.com , приступ 1. Март 2025 .

[8] https://www.macrocreator.com , приступ 8. Март 2025 .

[9] https://learn.microsoft.com/en-us/power-automate , приступ 8. Март 2025 .

[10] https://www.Python.org/about/ , приступ 18. Април 2025.

[11] https://textx.github.io/textX/ , приступ 18. Април 2025.

[12] https://pynput.readthedocs.io/en/latest/ , приступ 17. Maj 2025.

[13] https://pyautogui.readthedocs.io/en/latest/, приступ 18. Maj 2025.

[14] https://steam.fandom.com/wiki/Steam , приступ 29. Јун 2025.

[15] https://en.wikipedia.org/wiki/Spotify, приступ 29. Јун 2025.

[16] https://www.pathofexile.com, приступ 29. Јун 2025.

Kratka biografija:

Dušan Janošević rođen je u Majdanpeku 2001. godine. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Softversko inženjerstvo odbranio 2025. godine.

Kontakt: dusan.janosevic123@gmail.com

Зборник радова Факултета техничких наука Нови Сад

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/33BE29Bjelica

ИЗРАДА КОМПАЈЛЕРА УПОТРЕБОМ РАСТЕМО БИБЛИОТЕКЕ

COMPILER DEVELOPMENT USING THE RUSTEMO LIBRARY

Марко Бјелица, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У овом раду је уз ослонац на библиотеку Rustemo, изграђен компајлер Расти, назван тако јер је имплементиран у програмском језику Rust. У раду је дата теоријска основа за све фазе компајлирања које су реализоване у Растију, а то су лексичка, синтаксна и семантичка анализа, са највећим фокусом на синтаксну анализу. Расти се састоји од лексичког анализатора (имплементираног ручно и изгенерисаног помоћу Rustemo библиотеке), синтаксног анализатора (имплементираног ручно и изгенерисаног Rustemo библитеком), семантичког анализатора и евалуатора.*

Кључне речи: *Компајлер, Rustemo, Rust, лексички анализатор, синтаксни анализатор, семантички анализатор, евалуатор.*

Abstract – The paper presents a compiler, named Rasti, developed witth the *Rustemo library. Compiler name Rasti was chosen, because it is implemented in the Rust programming language. This work provides the theoretical foundation for all phases of compilation realized in Rasti, namely lexical, syntax and semantic analysis, with the greatest focus on syntax analysis. Rasti consists of a lexical analyzer (implemented manually and generated with Rustemo), a syntax analyzer (also implemented manually and generated with Rustemo), a semantic analyzer and an evaluator.*

Keywords: *Compiler, Rustemo, Rust, lexical analyzer, syntax analyzer, semantic analyzer, evaluator.*

1. УВОД

У свијету рачунарства, компајлер је програм који преводи изворни код написан на високом нивоу у циљни код погодан за извршавање на рачунару или виртуелној машини. Изворни код може бити написан у програмским језицима попут *C*, *C++*, *Rust*, *Java*, *Python* или чак у асемблеру. Циљни код може бити неки други програмски језик (најчешће нижег нивоа у односу на изворни), машински код, асемблер, бајткод или нека међуформа.

Процес компајлирања се састоји од неколико узастопних фаза, при чему свака фаза има одређен задатак, а то су:

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Игор Дејановић, ред. проф.

лексичка анализа, синтаксна анализа, семантичка анализа, генерисање међукода, оптимизација и на крају генерисање циљног кода.

У оквиру овог рада представљен је Расти, компајлер изграђен у програмском језику *Rust*, уз помоћ библиотеке *Rustemo* [1]. Фокус је на теоријској основи компајлирања и практичној имплементацији лексичке, синтаксне и семантичке анализе. У пројекту лексичка анализа је реализована на два начина, а то су ручном имплементацијом и генерисањем помоћу *Rustemo* библиотеке. Слично томе синтаксна анализа је реализована ручном имплементацијом парсера са рекурзивним спустом и аутоматски изгенерисаним LR парсером на основу прослијеђене граматике *Rustemo* библиотеци. Затим семантичка анализа обрађује синтаксно стабло добијено од парсера, провјерава логичку исправност и гради ново стабло. Након тога, евалуатор пролази кроз стабло настало семантичком анализом и извршава наредбе.

Расти подржава рад са изразима који садрже нумеричке и булове вриједности, као и комбиновање истих са аритметичким, логичким, релационим и унарним операторима. Такође, подржане су следеће наредбе: декларација промјењивих, додјела вриједности, петље и условне наредбе.

Рад је структуриран у четири дијела. Први дио представља наведени увод у којем је укратко описан концепт компајлера и које фазе има, такође дат је кратак осврт на Расти. У другом дијелу је теоријски преглед основа компајлера и опис фаза лексичке, синтаксне и семантичке анализе. Затим у трећем дијелу је описан Расти, кроз његове функционалности, архитектуру и разлике које изгенерисани и ручно имплементирани парсер носе. И на крају у четвртом дијелу дат је закључак у којем су разматране могуће примјене и будући правци развоја.

2. ТЕОРИЈСКЕ ОСНОВЕ

2.1. Процес компајлирања

За успјешно рјешавање проблема из пословног домена, неопходно је прецизно дефинисање проблема уз коришћење одговарајућег језика. Док за комуникацију међу људима користи се природни језик, рачунари нису у могућности да га интерпретирају на начин, на који то људи раде. Због тога је потребно користити језик који рачунари могу да разумију и формализовати проблем у складу са правилима тог језика.

Због сложености машинског језика за људе, развијени су програмски језици који омогућавају једноставнију комуникацију са рачунарима. Изворни код написан у неком програмском језику се не може директно извршавати на рачунару, јер рачунар разумије само инструкције у машинском коду. Због тога потребно је превести изворни код у облик који рачунар може разумијети и извршити. Тај превод обавља компајлер, који као улаз прихвата изворни програм, а као излаз генерише циљни код, и тај процес се зове компајлирање.

Сваком програмском језику потребан је преводилац, али неки језици нису погодни за све задатке, па се у неким доменима јавља потреба за креирањем нових, прилагођених језика. То су језици специфични за домен (енг. *Domain-specific languages*), осмишљени за рјешавање проблема у конкретном домену и за чију изградњу је потребно разумијевање процеса компајлирања.

2.2. Компајлери и интерпретери

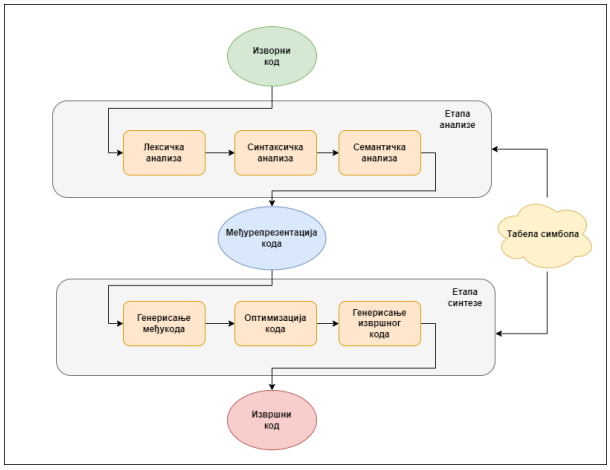
Први корак у рјешавању проблема из пословног домена, формулисаног терминологијом програмског језика, јесте превођење изворног у извршни код. Начин превођења може да се разликује у зависности да ли се користе компајлери или интерпретери [2].

Компајлери у потпуности преводе изворни код у извршни код, који је затим спреман за покретање. Овај приступ омогућава оптимизацију кода прије извршавања, што резултује бржим радом програма.

Интерпретери обрађују код инструкцију по инструкцију, без стварања извршне датотеке. Спорији приступ, али омогућава бољу дијагностику проблема, управо зато што се програм прати током извршавања.

2.3. Етапе компајлирања

Компајлирање је сложен процес који се састоји из више узастопних фаза. Те фазе се дијеле на двије главне етапе, а то су анализа и синтеза, као што је приказано на слици 1.



Слика 1. *Графички приказ компајлирања*

Етапа анализе или предњи дио (енг. *frontend*) компајлера обрађује изворни код и претвара га у међурепрезентацију, као што је апстрактно синтаксно стабло. Ова етапа укључује три фазе: лексичка, синтаксна и семантичка анализа, које провјеравају исправност кода према правилима језика.

Етапа анализе или задњи дио (енг. *backend*) компајлера користи међурепрезентацију за генерисање извршног кода.

Ове етапе нису строго раздвојене, већ међусобно сарађују путем табеле симбола, која чува податке о идентификаторима и игра кључну улогу у компајлирању.

2.4. Лексичка анализа

Лексичка анализа је прва фаза компајлирања, чији је задатак да чита изворни код карактер по карактер и претвара га у низ токена. Овај процес обавља лексички анализатор или лексер који:

- Препознаје лексеме и класификује их по типу градећи токене од њих.
- Прескаче непотребне елементе попут коментара и празног простора (ово је опционо, јер постоје језици гдје увлачења имају значење).
- Памти позицију токена у коду, ради лакшег лоцирања грешке уколико се деси.
- Гради табелу симбола са подацима о индентификаторима.
- Пријављује грешку, уколико не може да препозна улазни симбол.

Лексичка анализа је једина фаза која ради директно са карактерима, све наредне фазе раде са токенима.

2.5. Синтаксна анализа

Синтаксна анализа је друга фаза компајлирања која слиједи након лексичке анализе. Слично као што синтакса природног језика проучава правила која одређују како се ријечи комбинују у реченице, тако граматика програмског језика представља скуп правила који дефинише, како се токени могу комбиновати да би се добио исправно структуриран код.

Током ове фазе, синтаксни анализатор или парсер на основу токена, добијених из лексичке анализе, провјерава да ли њихов редослијед у складу са граматиком језика, притом градећи синтаксно стабло [3].

Синтаксно стабло или стабло парсирања (енг. *parse tree*) јесте хијерархијска структура која одражава организацију кода у складу са граматиком програмског језика. Формира се током синтаксне анализе и служи као основа за наредну фазу компајлирања, што је семантичка анализа.

Елементи синтаксног стабла:

- Чворови стабла могу бити терминали или нетерминали.
- Коријен стабла је почетни симбол граматике.
- Листови стабла су увијек терминали, јер се не могу даље расчланити.

У зависности од начина изградње синтаксног стабла, односно правца у којем парсер се креће приликом препознавања структуре програма, постоје два типа синтаксне анализе, а то су синтаксна анализа наниже (енг. *top-down parsing*) и синтаксна анализа навише (енг. *bottom-up parsing*).

2.5.1. Синтаксна анализа наниже

Синтаксна анализа наниже је приступ парсирању који започиње од почетног нетерминала граматике и покушава да конструише синтаксно стабло према доњим листовима, односно терминалима. Анализатор покушава да усклади улазне токене са продукцијама граматике, примјењујући правила тако да генерише улазни низ. У сваком кораку, одлука о примјени правила доноси се на основу предувидног симбола (енг. *lookahead*) [4].

Најзаступљенији парсери који користе ову методу су:

- Парсер са рекурзивним спустом (ручно имплементиран парсер у Растију).
- LL(1) парсер.

Овај приступ није отпоран на појаву лијеве рекурзије у граматици.

2.5.1.1. Рекурзивни спуст

Техника рекурзивног спуста представља приступ синтаксној анализи наниже, при којем се сваки нетерминални симбол граматике реализује као засебна рекурзивна функција. На основу предувидног симбола, ове функције селективно примјењују одговарајућа продукцијска правила. Терминални симболи се директно пореде са тренутним улазом, а у случају непоклапања јавља се синтаксна грешка.

2.5.2. Синтаксна анализа навише

Анализа навише представља процес конструисања синтаксног стабла за дати улазни низ токена, полазећи од његових листова и поступно сводећи га ка коријену, односно ка почетном симболу граматике. Током овог процеса, у сваком кораку анализе, неки подниз улаза који одговара десној страни правила граматике замјењује се његовом лијевом страном. Ако су ови поднизови изабрани на правилан начин, процес парсирања одговара обрнутом току крајњег десног извођења, гдје се продукције примењују од листова ка коријену стабла

2.5.2.1. Анализа навише пребацивањем и свођењем

Анализа навише пребацивањем и свођењем (енг. *shift-reduce parsing*) је општи облик синтаксне анализе навише која гради синтаксно стабло од листова ка коријену []. Овај приступ користи стек за праћење парцијалних резултата обраде и има двије методе:

- Пребацивање (енг. *shift*), метода која улазне симболе пребацује на стек.
- Свођење (енг. *reduce*), метода која секвенцу симбола замјењује нетерминалом, примјеном неког продукционог правила.

Циљ је на крају да стек садржи само почетни симбол граматике, чиме се потврђује синтаксна исправност улаза. Ову технику користе сви LR парсери.

2.5.2.2. LR парсер

LR парсери представљају класу парсера за анализу навише који користе технику пребацивања и свођења. Назив LR означава:

- **L** слово означава читање улаза са лијева на десно (енг. ***L****eft-to-right*).
- **R** слово означава конструисање обрнуто крајње десно извођење стабла (енг. ***R****ightmost derivation in reverse*).

LR парсери доносе одлуке током анализе користећи стек и табелу анализе, на основу којих утврђују да ли треба наставити читање улаза, свести препознате симболе, прихватити улаз или пријавити синтаксну грешку.

2.6. Семантичка анализа

Семантичка анализа је завршна фаза етапе анализе у процесу компајлирања. Док синтаксна анализа провјерава структуру кода према граматици, семантичка анализа провјерава логичку исправност програма. Њен циљ је да осигура да су сви идентификатори исправно дефинисани, да се користе у складу са својим типовима, и да се поштују правила као што су видљивост и права приступа.

Неке од најчешћих семантичких грешака:

- Неслагање типова (енг. *type mismatch*).
- Коришћење недекларисаних промјенљивих
- Вишеструка декларација у истом опсегу.
- Приступ промјенљивој ван њеног опсега.
- Неслагање између формалних и стварних параматера функције.

Семантичка анализа се може изводити током или након синтаксне анализе. Једнопролазни компајлери врше синтаксну и семантичку анализу у једном пролазу, што је ефикасно, док вишепролазни компајлери одвајају ове кораке ради веће флексибилности и боље подршке за сложене структуре програма, овај принцип је имплементиран у Растију.

3. СПЕЦИФИКАЦИЈА СИСТЕМА

3.1. Функционалности система

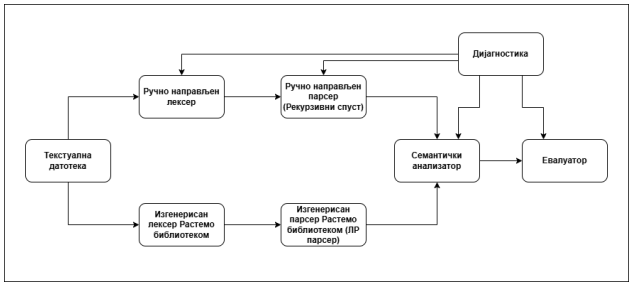
Пројекат Расти представља имплементацију компајлера написаног од нуле у програмском језику Раст. Подржава парсирање и евалуацију израза са нумеричким и буловим вриједностима, аритметичким, логичким, релационим и унарним операторима, као и рад са заградама. Поред тога, обухвата основне програмске конструкције као што су додјела, константе, условне наредбе, петље и угњеждени блокови кода.

3.2. Архитектура

Пројекат Расти представља имплементацију компајлера за императиван програмски језик. Иако не прати у потпуности класичне фазе компајлирања, састоји се од више компоненти, које су приказане на слици 2.

Архитектура приказана на слици обухвата:

- Ручно имплементиран лексер који учитава текст у радну меморију и генерише токене обрађујући га карактер по карактер.
- Ручно имплементиран парсер који гради синтаксно стабло методом рекурзивног спуста.
- Лексер изгенерисан Растемо библиотеком, који учитава токен по потреби и ради ефикасније од ручно имплементираног, јер не држи цијели код у радној меморији.
- LR парсер изгенерисан Растемо библиотеком.
- Семантички анализатор пролази кроз стабло парсирања и везује га семантичким правилама, при чему гради ново стабло.
- Након успјешне семантичке анализе, евалуатор обилази ново стабло, извршава наредбе, евалуира изразе и крајњи резултат приказује кориснику.
- Дијагностика која прикупља логове и грешке уколико се десе у свим фазама компајлирања.



Слика 2. *Архитектура пројекта*

3.3. Разлике између изгенерисаног и ручно имплементираног парсера

Највећа разлика између LR парсера и парсера са рекурзивним спустом лежи у приступу парсирању. LR парсер врши синтаксну анализу навише и обично се аутоматски генерише, док парсер са рекурзивним спустом обавља анализу наниже и пише се ручно. У овом поглављу биће разматране конкретне разлике између ових парсера имплементираних у пројекту.

Ручна имплементација парсера са рекурзивним спустом је једноставна и ефикасна за једноставне граматике, али мање проширива и осјетљива на граматичке проблеме као што је лијева рекурзија. Са друге стране, изгенерисани LR парсер пружа бољу подршку за проширења и аутоматску детекцију граматичких конфликата, што доприноси стабилности и лакшој одрживости пројекта.

Ефикасност је у оба случаја адекватна тренутној граматици, али LR приступ је прикладнији за даљи развој и сложеније граматике. Обје имплементације нуде сличан ниво дијагностике, коју чине информативне поруке и грешке.

4. ЗАКЉУЧАК

Рад представља приказ процеса изградње једноставног компајлера званог Расти, написаног у програмском језику *Rust* уз подршку библиотеке *Rustemo*. Основна мотивација за реализацију рада била је боље разумијевање процеса компајлирања и учење новог програмског језика.

На самом почетку рада дате су теоријске основе о томе шта је то компајлер, процес компајлирања и које разлике има у односу на интерпретере. Затим описана је етапа анализе коју чине лексичка, синтаксна и семантичка анализа. Посебан акценат стављен је на синтаксну анализу, гдје се обрађују методе синтаксне анализе наниже и навише.

У другом дијелу рада описана је спецификација система, која обухвата функционалност и архитектуру Растија. У оквиру тог поглавља дато је и образложење разлика између изгенерисаног и ручно имплементираног парсера.

Тренутна имплементација Растија пружа стабилну основу за будући развој и унапријеђење. Потенцијални правци даљег развоја укључују функционално проширење програмског језика, као што је увођење функција, структура или класа, што омогућава праћење савремених програмских стандарда. Такође, може се надоградити процес компајлирања, додавањем фаза као што су генерисање међукода, оптимизација и креирање циљног кода. Поред тога, простор за напредак јесте могућа имплементација неке од техника опоравка од грешака у лексичкој и синтаксној анализи, што доприноси робусности компајлера. Сви ови правци доприносе свеукупном квалитету компајлера и развојном потенцијалу програмског језика.

5. ЛИТЕРАТУРА

[1] Растемо (енг. *Rustemo*) библиотека - https://www.igordejanovic.net/rustemo/ (посљедњи приступ: 05.05.2025)

[2] Компајлери и интерпретери - https://www.spiceworks.com/tech/tech-general/articles/compiler-vs-interpreter-12-critical-differences-to-know/ (посљедњи приступ: 05.05.2025)

[3] Aho, A. V., Lam, M. S., Sethi, R., Ullman, J. D. *Compilers: Principles, Techniques, and Tools*. 2. издање. Boston: Addison-Wesley, 2006.

[4] Cooper, K. D., Torczon, L. *Engineering a Compiler*. 3. издање. Cambridge, MA: Morgan Kaufmann, 2022.

Кратка биографија:

Марко Бјелица рођен је у Требињу 1999. године. Мастер рад на Факултету техничких наука из области Рачунарство и аутоматика – Електронско пословање је одбранио 2025. године.

контакт: marko.bjelica19@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE30Vulin

JEZIK ZA OPIS PRAVILA ZA IGRE SA KARTAMA

A LANGUAGE FOR DESCRIBING CARD GAME RULES

Ana Vulin, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je predstavljen dizajn i implementacija jezika specifičnog za domen (DSL) namenjenog opisivanju pravila za igre sa kartama. Sistem obuhvata textX parser, interpreter u Python-u, kao i veb klijent sa podrškom za interaktivnu vizuelizaciju. Cilj je omogućiti lako kreiranje igri sa kartama bez potrebe za programerskim znanjem.*

Ključne reči: *DSL, textX, Python, WebSocket, vizualizacija, igre sa kartama*

Abstract – *The paper presents the design and implementation of a domain-specific language (DSL) intended for describing the rules of card games. The system includes a textX parser, a Python-based interpreter, and a web client with support for interactive visualization. The goal is to enable easy creation of card games without the need for programming knowledge.*

Keywords: *DSL, textX, Python, WebSocket, visualization card games*

1. UVOD

Kartaške igre predstavljaju jedan od najrasprostranjenijih oblika društvene zabave, ali i pogodan model za proučavanje i implementaciju različitih pravila, strategija i interakcija. Njihova struktura – sa jasno definisanim fazama, pravilima i uslovima pobede – čini ih idealnim kandidatom za formalizaciju kroz *DSL*. Cilj ovog rada je razvoj i implementacija *DSL* rešenja koje omogućava jednostavno, fleksibilno i proširivo definisanje i pokretanje različitih kartaških igara.

Predloženo rešenje – *CardGame DSL* – omogućava korisnicima da tekstualno opišu pravila igre koristeći specijalizovanu DSL gramatiku, a sistem automatski generiše i izvršava logiku igre, čime se ubrzava razvoj i olakšava učešće osobama bez programerskog znanja.

Sistem koristi *textX* za definisanje i parsiranje *DSL* sintakse, dok je izvršna logika realizovana u *Python-u* kroz specijalizovan interpreter. Klijentska aplikacija je razvijena u *JavaScript-u* sa *HTML* i *CSS* podrškom, a komunikacija klijent–server odvija se u realnom vremenu putem *WebSocket-a*, čime je omogućena modularnost i ponovna upotreba sistema.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Igor Dejanović, red. prof.

2. TEORIJSKE OSNOVE

2.1. Jezici specifični za domen (*DSL*)

Jezik specifičan za domen [1] je jezik napravljen za rešavanje problema u određenom domenu. Na primer, *HTML* je *DSL* za opisivanje veb stranica, a *SQL DSL* za rad sa bazama podataka. Nasuprot DSL-u se nalazi jezik opšte namene (*GPL – General Purpose Language*) koji ima široku primenu u različitim oblastima razvoja softvera (*Python*, *Java*,..)

2.1.1. Apstraktna i konkretna sintaksa

Apstraktna sintaksa [2] – logička struktura koda bez detalja koji nisu bitni za razumevanje. Najčešće se koristi u vidu stable apstraktne sintakse (AST – Abstract Syntax Tree).

Konkretna sintaksa – način na koji korisnik unosi programski kod ili model. Može biti tekstutualna, grafička, tabelarna.

U okviru tekstualne konkretne sintakse definiše se gramatika. Predstavlja formalizovani skup pravila koji definiše dopustive nizove karaktera i njihove međusobne odnose.

2.1.2. Parsiranje

Proces parsiranja uzima tekst u konkretnoj sintaksi i pretvara ga u strukturu u apstraktnoj sintaksi (AST).

2.1.3. Interpretacija i kompajliranje

Postoje dva glavna načina izvršavanja koda ili naredbi – interpretacija i kompajliranje/generisanje koda. Interpretacija – izvorni kod ili neka njegova reprezentacija (npr. *AST*) se direktno čita i izvršava od strane interpretera. Kompajliranje ili generisanje koda – proces stvaranja koda napisanog u nekom jeziku u niz instrukcija ili mašinski kod koji procesor može direktno izvršiti.

2.2. Igre sa kartama

Igre sa kartama [3] predstavljaju raznovrstan oblik društvenih igara i sve više se koriste u obrazovanju. Mogu se klasifikovati prema svrsi i pravilima, npr. igre na sreću, strateške, pamćenja i opažanja, socijalne i edukativne igre.

3. SPECIFIKACIJA SISTEMA

3.1. Jezici specifični za domen (*DSL*)

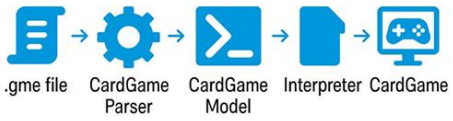
Glavni cilj sistema jeste da omogući lako i brzo kreiranje kartaške igre koristeći unapred definisan skup pravila i to na način koji je intuitivan i pristupačan i osobama bez programerskog znanja.

3.2. Postojeća rešenja

Postojeća rešenja kao što su biblioteka Cardamom.js [4] i Construct 3 [5] podržavaju razvoj kartaških igara. Cardamom.js omogućava rad sa kartama. Špilovima i rukama kroz funkcije za kreiranje, mešanje i upravljanje. Construct 3 pruža vizuelni, objektno orijentisan sistem za razvoj 2D igara bez pisanja koda.

3.3. Arhitektura obrade i interpretacije *DSL-a*

Jezik *CardGame* je eksterni *DSL*. Osnovna komponenta sistema je *Game Parser*, koji na osnovu definisane gramatike parsira specifikaciju koja opisuje igru. *Parser* vrši osnovnu validaciju specifikacije (*.gme* fajla) i kao rezultat vraća model koji se nakon toga prevodi u interni *CardGame Model* čija je uloga semantička validacija. *Interpreter* predstavlja glavni deo sistema i vrši tumačenje pravila igre na osnovu modela, omogućavajući njihovu primenu i izvršavanje. Na slici 1 je dat šematski prikazan put od *.gme* fajla do igre.



Slika 1: *Arhitektura obrade i interpretacije* DSL-*a*

3.3.1. *Meta-model*

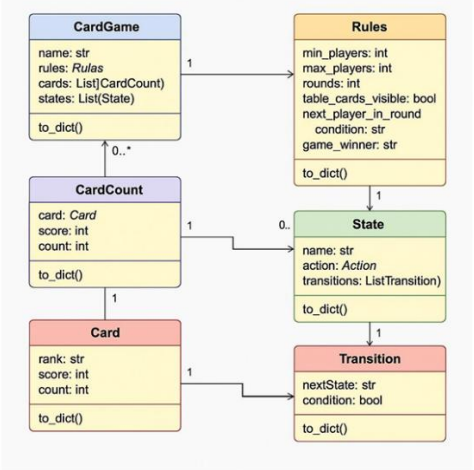
Meta-model se sastoji od elemenata: *Game*, *Rules, State, Transition, Action, CardCount, Card, EnumValue, Rank, Suit* i drugih pomoćnih klasa kao što su *ParamList*, *Param*. Svaka klasa ima svoje atribute i međusobne relacije.

3.3.2. *Parser*

Za obradu specifikacija napisanih u *CardGame DSL-u* koristi se biblioteka *textX*, koja omogućava definisanje gramatike i automatsko generisanje parsera.

3.3.3. *Model*

Radi postizanja veće stabilnosti, podaci dobijeni iz parsera se ne koriste direktno nego se transformišu u interni model. Interni model (slika 2) predstavlja skup specijalizovanih klasa dizajniranih da reprezentuju elemente igre sa svim potrebnim svojstvima i pravilno implementiranom logikom.



Slika 2: *Dijagram klasa internog modela*

3.3.4. *Interpreter*

Interpreter je centralni deo sistema razvijen u *Python-u* koji, koristeći biblioteku *textX*, tumači specifikacije igara definisanih u *CardGame DSL-u* i upravlja logikom i tokom igre. Integrisan je u *backend Flask* aplikaciju organizovanu po *kontroler-servis-repozitorijum* arhitekturi, uz korišćenje *WebSocket* protokola za realnovremensku komunikaciju sa frontend-om. Stanja igre, potezi i rezultati se čuvaju u relacionoj bazi podataka, što omogućava kontinuitet partije. Sistem je dizajniran tako da omogući proširivost i ponovnu upotrebu, bez potrebe za dodatnim kodiranjem pri definisanju novih igara

4. IMPLEMENTACIJA SISTEMA

4.1. Tehnologije

Ovaj sistem kombinuje *textX*, *Python, Flask, PostgreSQL*, *WebSocket, HTTP, JavaScript, HTML i CSS* za izradu interaktivnog *DSL-a* i veb aplikacije. *Backend* upravlja igrom i komunikacijom u realnom vremenu dok *frontend* dinamički generiše interfejs.

4.1.1. *textX*

TextX [6] je *meta-jezik* za definisanje gramatika *DSL-a* u *Python-u*. U ovom rešenju koristi se za parsiranje CardGame skripti i generisanje *AST*.

4.1.2. *Python*

Python [7] je dinamički jezik pogodan za obradu teksta, integraciju i *backend* aplikacije. Korišćen je za *parser*, *backend* server i upravljanje bazom podataka.

4.1.3. *Flask*

Flask [8] je *backend framework* za *Python*, pogodan za veb aplikacije. U sistemu *CardGame* omogućava *HTTP* zahteve, upravljanje stanjem igre i *WebSocket* komunikaciju.

4.1.4. *WebSocket*

WebSocket [9] je protokol koji omogućava dvosmernu komunikaciju u realnom vremenu. Ovde se koristi za razmenu poruka između igrača, bez potrebe za učestalim *HTTP* pozivima.

4.1.5. *HTTP*

HTTP [10] je osnovni protokol za komunikaciju između klijenta i servera. U sistemu se koristi za učitavanje stranica, prijavu i registraciju korisnika.

4.1.6. *PostgreSQL*

PostgreSQL [14] je objektno-relaciona baza podataka poznata po stabilnosti i skalabilnosti. U *CardGame* rešenju služi za skladištenje korisničkih podataka, rezultata i stanja partija.

4.1.7. *HTML, CSS, JavaScript*

HTML [11] definiše strukturu veb interfejsa i sadržaja aplikacije. *CSS [12]* određuje izgled veb stranice, uključujući boje, fontove i razmeštaj elemenata. *JavaScript [13]* omogućava interaktivnost i dinamičko ažuriranje veb interfejsa. U ovom rešenju upravlja *WebSocket* komunikacijom, generisanjem *HTML* sadržaja i reagovanjem na korisničke akcije.

4.2. *CardGame DSL* i gramatika

CardGame DSL je realizovan kao eksterni tekstualni *DSL*. *CardGame*, osnovni element gramatike, prikazan je na slici 3. Sadrži ime igre kao i blokove *rules, states i cards* koji opisuju pravila, stanja i karte

```
CardGame:
    'game' name=ID
     rules=Rules
     states=States
     cards=Cards
;
```

Slika 3: *Element gramatike* CardGame

Element *Rules* prikazan je na slici 4 i predstavlja pravila koja uključuju broj igrača, broj rundi, način izbora sledećeg igrača i kriterijum za pobedu.

```
Rules:
    'rules'
        'min_players' min_players=INT
        'max_players' max_players=INT
        'rounds' rounds=INT
        ('table_cards_visible' table_cards_visible=BOOLEAN)?
        'next_player_in_round_condition' next_player_in_round_condition=NextPlayerCondition
        'game_winner' game_winner=GameWinner
;
```

Slika 4: *Element gramatike* Rules

Element *States* prikazan na slici 5 predstavlja stanja odnosno logičke faze igre koje sadrže akcije i tranzicije ka drugim stanjima.

```
States:
    'states'
        states+=State
;
State:
    'state' name=ID
    'do' action=Action
     transitions+=Transition*
;
```

Slika 5: *Element gramatike* States

Element *Action* prikazan na *slici 6* predstavlja radnju koja se izvršava u okviru određenog stanja. Svaka akcija ima svoj jedinstveni naziv *(name=ID)* kao i opcionalni *deo ('(' params=ParamList ')')?* koji predstavlja parametre akcije koji mogu, a ne moraju biti prisutni.

```
Action:
    name=ID ('(' params=ParamList ')')?
;
```

Slika 6: *Element gramatike* Action

Element *Transition* prikazan na slici 7 predstavlja tranziciju koja pokazuje u koje sledeće stanje igra prelazi. Sastoji se od ključne reči *then* nakon čega sledi naziv sledećeg stanja, a zatim sledi opcioni deo *('if' condition=BOOLEAN)?*. Ovaj deo je opcion jer se mogu javiti različiti scenariji:

1. Ako nema definisanih tranzicija to je kraj igre
2. Ako postoji samo jedna tranzicija bez uslova, prelaz u sledeće stanje se odvija automatski
3. Ako postoji više tranzicija sa uslovima, prelaz se odvija u prvo stanje čiji je uslov ispunjen
4. Ako postoji više tranzicija bez uslova, sledeće stanje zavisi od igrača ili nekog spoljašnjeg faktora

```
Transition:
    'then' nextState=[State] ('if' condition=BOOLEAN)?
;
```

Slika 7: Element gramatike *Transition*

Element gramatike *Cards* prikazan na slici 8 predstavlja karte. Svaka karta (*Card*) je definisana preko ranga i boje, a navodi se i broj njenog pojavljivanja, kao i koliko poena vredi (*CardCount*).

```
Cards:
    'cards'
        cards+=CardCount
;

CardCount:
    'card' card=Card 'appears' count=INT 'times, worth' score=INT 'points'
;

Card:
    rank=Rank 'of' suit=Suit
;
```

Slika 8: *Element gramatike* Cards

4.3. Parsiranje i model

Parsiranje *.gme* datoteka realizovano je uz pomoć *textX* parsera. Parsirani podaci se transformišu u interni model koji vrši dodatnu semantičku validaciju. Interni model predstavlja skup *Python* klasa koje definišu strukturu igre. Ovaj model je osnova za interpretiranje logike igre.

4.4. *Interpreter* i povezivanje komponenti

Interpreter napisan u *Python-u* tumači pravila i upravlja tokom igre. On je integrisan u *Flask backend*, koji kroz *WebSocket* komunicira sa *frontend-om*. Ovo omogućava dinamički tok igre bez učitavanja stranice. Stanja igre i rezultati čuvaju se u *PostgreSQL* bazi podataka.

4.5. *Frontend* i korisnički interfejs

Frontend je razvijen u *HTML/CSS/JavaScript* tehnologijama. Koristi *WebSocket* za interaktivnu komunikaciju i prikazuje tok igre u realnom vremenu. Sistem omogućava registraciju, prijavu, pregled postojećih igara i igranje igre sa protivnikom.

4.6. Pomoćne funkcionalnosti

Sistem sadrži niz funkcionalnosti za podršku korisnicima kao što su *syntax highlighting*, podrška za *snippet-e* i automatska vizualizacija grafa prelaza stanja.

4.7. Konfiguracija i primer upotrebe sistema

Za pokretanje sistema instalirati *Python 3.6+, Flask, PostgreSQL, PyCharm ili VSC.* Klonirati repozitorijum *https://github.com/vulinana/card_game_dsl.git* U *PyCharm-u*, dodati interpreter iz foldera *card_game_dsl*. Dodati *.gme* fajl u *gme/games*. Nakon prijave, u glavnom interfejsu biće dostupna novododata igra.

5. PRIMER SKRIPTE U CARDGAME *DSL-u*

Igra memorijskih karata namenjena je za dva do četiri igrača, a cilj je pronaći što više parova istog ranga. Sastoji se od pet rundi u kojima se igrači smenjuju, dok se karte postavljaju licem nadole. Pobednik je onaj sa najviše poena. Osnovna pravila definisana *CardGame DSL-om* prikazana su na slici 9.

```
game memo_cards
rules

  min_players 2
  max_players 4
  rounds 5
  table_cards_visible false
  next_player_in_round_condition circle_order
  game_winner highest_score
```

Slika 9: *Osnovna pravila igre memorijskih karata*

Na početku igre, na sto se nasumično postavlja osam karata (*deal_table_cards*). Igrač koji je na potezu (*next_player*) bira dve karte (*action_phase*). Ova logika implementirana je kroz stanja prikazana na slici 10.

```
states

state deal_table_cards
  do deal_table_cards(8)
  then any_matching_table_cards

state next_player
  do next_player
  then action_phase

state action_phase
  do select_table_cards(2)
  then reveal
```

Slika 10: *deal_table_cards*, *next_player* i *action_phase*

Karte se okreću (*reveal*) i proverava da li su istog ranga (*matching*_cards). Ako nisu, vraćaju se licem nadole (*flip_back*), a ako jesu, igrač dobija poene (*calculate_points*) i karte se uklanjaju (remove_selected_cards). Prikaz stanja dat je na slici 11.

```
state reveal
  do reveal_selected_cards
  then matching_cards

state matching_cards
  do selected_cards_match(rank)
  then flip_back if false
  then mark_matching_cards_for_scoring if true

state flip_back
  do reset_table_cards_visibility
  then next_player

state mark_matching_cards_for_scoring
  do mark_matching_cards_for_scoring
  then calculate_points

state calculate_points
  do calculate_points
  then remove_selected_cards

state remove_selected_cards
  do remove_selected_cards
  then any_matching_table_cards
```

Slika 11: *reveal, matching_cards, flip_back, mark_matching_cards_for_scoring, calculate points i remove_selected_cards*

Sistem proverava da li na stolu postoje parovi (*any_matching_table_cards*). Ako postoje, igra nastavlja sa sledećim igračem (*next_player*), a ako ne, počinje nova runda (*new_round*). Nakon provere broja rundi (*rounds_remaining*), igra se završava određivanjem pobednika (game_end) ili ponovnim deljenjem karata (deal_table_cards). Prikaz stanja dat je na slici 12

```
state any_matching_table_cards
  do any_matching_table_cards(rank)
  then next_player if true
  then new_round if false

state new_round
  do new_round
  then rounds_remaining

state rounds_remaining
  do check_if_rounds_remaining
  then deal_table_cards if true
  then game_end if false

state game_end
  do determine_game_winner
```

Slika 12: *any_matching_table_cards, new_round, rounds_remainning i game_end*

Na kraju je potrebno još definisati karte koje se mogu pojaviti u toku igre, kao i broj njihovog ponavljanja i broj poena koji nose (slika 13).

```
cards

  card 5 of hearts appears 4 times, worth 0 points
  card 14 of hearts appears 4 times, worth 10 points
  card 13 of hearts appears 4 times, worth 10 points
  card 5 of diamonds appears 4 times, worth 0 points
  card 13 of diamonds appears 4 times, worth 15 point
  card 10 of clubs appears 6 times, worth 10 points
  card 7 of clubs appears 4 times, worth 0 points
  card 6 of clubs appears 6 times, worth 0 points
  card 11 of clubs appears 4 times, worth 15 points
```

Slika 13: *Karte koje se mogu pojaviti u igri*

6. ZAKLJUČAK

Razvijeni sistem pruža efikasno rešenje za dizajn kartaških igara, zasnovano na dostupnosti i jednostavnosti. Deklarativni pristup omogućava intuitivno definisanje pravila, toka igre i ponašanja igrača, čime se uklanja potreba za poznavanjem programskih jezika i sistem postaje dostupan širokom spektru korisnika.

Dalji razvoj sistema omogućava dodavanje funkcionalnosti koje bi unapredile korisničko iskustvo, kao što su:

1. **Proširenje skupa interpretiranih akcija**
2. **Vizuelni editor pravila** – grafički interfejs za kreiranje i uređivanje pravila bez pisanja *DSL* koda.
3. **Automatska dijagnostika grešaka** – prepoznavanje nelogičnih mehanika uz predlog rešenja.
4. **Simulacija testnih partija** – automatsko izvođenje više partija radi analize funkcionalnosti igre.
5. **Interaktivni *debugger* pravila** – korak-po-korak interpretacija pravila i lakše otkrivanje grešaka.

LITERATURA

[1] https://en.wikipedia.org/wiki/Domain-specific_language (pristupljeno u maju 2025.)

[2] https://en.wikipedia.org/wiki/Abstract_syntax_tree (pristupljeno u maju 2025.)

[3] https://gambiter.com/cards/classified-index.html?utm_source=chatgpt.com (pristupljeno u junu 2025.)

[4] https://marianpekar.github.io/Cardamom.js/ (pristupljeno u maju 2025.)

[5] https://www.construct.net/en (pristupljeno u maju 2025.)

[6] https://textx.github.io/textX/ [pristupljeno u avgustu 2025.)

[7] https://www.python.org/doc/ (pristupljeno u maju 2025.)

[8] https://flask.palletsprojects.com/en/latest/ (pristupljeno u maju 2025.)

[9] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (pristupljeno u maju 2025.)

[10] https://developer.mozilla.org/en-US/docs/Web/HTTP (pristupljeno u maju 2025.)

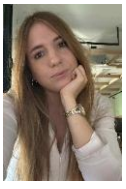
[11] https://developer.mozilla.org/en-US/docs/Web/HTML (pristupljeno u maju 2025.)

[12] https://developer.mozilla.org/en-US/docs/Web/CSS (pristupljeno u maju 2025.)

[13] https://developer.mozilla.org/en-US/docs/Web/JavaScript (pristupljeni u maju 2025.)

[14] https://www.postgresql.org/about/ (pristupljeno u junu 2025.)

Kratka biografija:



Ana Vulin rođena je u Sremskoj Mitrovici 2000. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Softversko inženjerstvo, odbranila je 2025.god.

kontakt: sm.vulinana@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.633
DOI: https://doi.org/10.24867/33BE31Petric

METODE OTKLJUČAVANJA MOBILNIH TELEFONA

METHODS FOR UNLOCKING MOBILE PHONES

Jelena Petrić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad se bavi forenzičkim metodama otključavanja ekrana Android i iOS uređaja, uz analizu savremenih bezbednosnih mehanizama i izazova koje oni postavljaju. Analizirani su operativni sistemi i bezbednosne arhitekture pametnih telefona, kao i postojeći pristupi zaobilaženja biometrijske autentifikacije, lozinki i šablonske zaštite uz primenu forenzičkih alata. Posebna pažnja je posvećena praktičnom testiranju otpornosti biometrijskih sistema i istraživanju karakterističnih obrazaca za zaključavanje, uz osvrt na etičke i pravne aspekte prikupljanja dokaza.*

Ključne reči: *digitalna forenzika, mobilni uređaji, zaštitni mehanizmi, metode otključavanja*

Abstract – *This paper deals with forensic methods for unlocking the screens of Android and iOS devices, along with an analysis of modern security mechanisms and the challenges they pose. Smartphone operating systems and security architectures were analyzed, as well as existing approaches to bypassing biometric authentication, passwords, and pattern protection using forensic tools. Special attention is paid to practical testing of the resilience of biometric systems and research of characteristic locking patterns, with a focus on the ethical and legal aspects of evidence collection.*

Keywords: *digital forensics, mobile devices, protection mechanisms, unlocking methods*

1. UVOD

Otključavanje ekrana predstavlja bitan korak u procesu prikupljanja dokaza tokom forenzičke istrage, jer ovaj čin omogućava direktan pristup podacima u njihovom originalnom i neizmenjenom obliku. Uspešnim otključavanjem izbegava se upotreba rizičnih i potencijalno destruktivnih metoda zaobilaženja zaštite, koje mogu ugroziti integritet ili pravnu prihvatljivost dokaza.

Zadatak ovog rada je da istraži forenzičke metode koje se koriste za otključavanje ekrana pametnih telefona i da putem izvedenih zaključaka demonstrira postojeće metode na odabranim uređajima. Rad je isključivo edukativnog karaktera i nema za cilj podsticanje ili promociju zloupotrebe stečenih znanja, već prvenstveno unapređenje razumevanja metoda otključavanja mobilnih uređaja u kontekstu digitalne forenzike.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Stevan Gostojić, red. prof.

2. FORENZIKA MOBILNIH UREĐAJA

Forenzika mobilnih uređaja predstavlja specifičnu granu digitalne forenzike, usmerenu na prikupljanje i analizu podataka pohranjenih na mobilnim uređajima ili prenošenih putem celularne mreže [1].

2.1. Izazovi forenzike mobilnih uređaja

Brza evolucija mobilnih tehnologija i kratak ciklus razvoja proizvoda doveli su do velikih razlika u hardveru, softveru i strukturama čuvanja podataka između različitih proizvođača i verzija operativnih sistema mobilnih uređaja. Ova raznovrsnost, u kombinaciji sa sve naprednijim bezbednosnim mehanizmima stvara ozbiljne izazove prilikom prikupljanja dokaza u forenzičkim istragama.

2.2. Proces forenzičke istrage

Forenzička istraga je proces koji obuhvata identifikaciju, prikupljanje, čuvanje, pregled, analizu i prezentaciju dokaza korišćenjem pravno i naučno prihvaćenih metoda i alata.

2.3. Forenzički alati

U situacijama kada je telefon zaključan obično je neophodna upotreba specijalizovanih tehnika ili alata za zaobilaženje zaštitnih slojeva kako bi se na bezbedan i legalan način došlo do traženih informacija. Forenzički alati su softverske i hardverske tehnologije koje se koriste u istrazi kao pomoćna sredstva prilikom pristupa, obrade i ekstrakcije podataka sa različitih uređaja.

2.3. Jailbreaking i rooting

Najčešće korišćeni postupci za sticanje najvišeg nivoa privilegija nad mobilnim operativnim sistemima uključuju procese root-ovanja na Android platformama i jailbreak-a na iOS uređajima. Primena ovih metoda nosi velike rizike po integritet digitalnih dokaza i zahteva pažljivo izvođenje i strogo poštovanje forenzičkih principa.

3. OPERATIVNI SISTEMI MOBILNIH TELEFONA

Zbog brojnih mehanizama zaštite koje implementiraju savremene mobilne platforme, alati i metode koji su efikasni na jednom sistemu često nisu primenljivi na drugom zbog specifičnosti arhitekture i nivoa otvorenosti platforme.

3.1. Tipovi operativnih sistema

Tehnike za zaobilaženje zaključanog ekrana često uključuju složene procedure koje zavise od hardvera i verzije operativnog sistema. Prema statistici [2] za mesec avgust 2025. godine, trenutno tržištem mobilnih telefona dominiraju dva operativna sistema, Android i iOS.

3.2. Zaštitni mehanizmi

Zaštitni mehanizmi mobilnih telefona obuhvataju skup mera i tehnologija dizajniranih da zaštite uređaj i privatnost korisnika od krađe i drugih oblika zloupotrebe.

3.2.0. Mehanizmi zaključavanja ekrana

Mehanizmi poput PIN-a, lozinke, šablona i biometrijske autentifikacije se razlikuju po kompleksnosti i nivou bezbednosti koji mogu da pruže korisniku u zaštiti podataka.

3.2.1. Zaštita naloga i enkripcija

Enkripcija podrazumeva pretvaranje svih podataka na telefonu u nečitljiv format, koji se može dešifrovati jedino putem odgovarajućeg kriptografskog ključa. Na starijim modelima implementirana je full disk encryption (FDE), dok se na novijim modelima koristi file-based encryption (FBE).

3.2.2. Izolacija aplikacija

Sandbox predstavlja izolovano okruženje u kome aplikacije mogu nesmetano da obavljaju svoje funkcije, ali bez direktne interakcije sa drugim aplikacijama ili kritičnim komponentama sistema.

3.3. iOS

Za razliku od otvorenijih platformi, iOS sistem ograničava pristup resursima, čime se umanjuje mogućnost neautorizovanog pristupa podacima. Postupak za otključavanje ovih uređaja zahteva upotrebu naprednih forenzičkih alata i tehnika, kao i detaljno poznavanje arhitekture sistema. U pojedinim slučajevima, posebno kod novijih modela, saradnja sa proizvođačem može biti neophodna kako bi se osigurala forenzički prihvatljiva i tehnički izvodljiva obrada digitalnih dokaza.

3.3.0. Arhitektura

Arhitektura je zasnovana na višeslojnom modelu koji obuhvata četiri osnovna sloja: Cocoa Touch, Media, Core Service i Core OS.

3.3.1. Bezbednost sistema

Apple je razvio bezbednosni model koji objedinjuje hardverske i softverske komponente u cilju zaštite korisničkih podataka i očuvanja integriteta operativnog sistema. Ovaj nivo zaštite predstavlja značajan izazov u forenzičkim istragama, posebno kada je uređaj zaključan i nije jailbreak-ovan jer onemogućava pristup kritičnim sistemskim podacima i aplikacijama bez odgovarajućih autorizacija.

3.3.1.0. Sigurnosno pokretanje procesa

Potpisivanje koda je proces digitalnog potpisivanja izvršnih fajlova ili softvera kako bi se verifikovala autentičnost i integritet programa u cilju sprečavanja instalacije neproverenih aplikacija.

3.3.1.1. Bezbedna enklava

Bezbedna enklava predstavlja izolovano okruženje unutar iOS uređaja, koje funkcioniše nezavisno od glavnog procesora i poseduje sopstevni memorijski kontroler i kriptografske funkcije. Njen dizajn se zasniva na principu separacije privilegija.

3.3.1.2. Zaštita podataka

Apple primenjuje enkripciju celog diska, kojom šifruje kompletan sadržaj uređaja, uključujući sistemske i korisničke podatke. Operativni sistem definiše dva ključna stanja: before first unlock (BFU) i after first unlock (AFU). Ključnu ulogu u sigurnosti igra jedinstveni UID ključ, ugrađen u bezbednu enklavu.

3.4. Android

Zbog otvorene prirode Android platforme, uređaji mogu biti podložni zloupotrebama, naročito ukoliko ne dobijaju redovna bezbednosna aužuriranja. U takvim slučajevima, zlonamerne aplikacije mogu iskoristiti poznate ranjivosti, kompromitovati izolovano oktuženje i steknuti neovlašćene privilegije, što dovodi do ugrožavanja sigurnosti sistema.

3.4.0. Arhitektura

U osnovi arhitekture se nalazi Linux kernel, koji predstavlja temelj čitavog sistema. Iznad njege nalaze se Hardware Abstraction Layer, Android Runtime i Application Framework.

3.4.1. Bezbednost sistema

Bezbednosna arhitektura u velikoj meri se oslanja na Trusted Execution Environment ili Secure Element okruženja. Savremene verzije Android-a podržavaju ugrađenu enkripciju podataka za razliku od ranijih verzija koje često nisu imale podrazumevano omogućeno šifrovanje diska, niti su posedovale druge naprednije bezbednosne mere zaštite.

3.4.1.0. Šifrovanje diska

FBE deli skladišni prostor uređaja na dva odvojena kriptografska prostora: Credential Encrypted i Device Encrypted prostor.

3.4.1.1. Trusted Execution Environment

Trusted Execution Environment predstavlja izolovano hardevrsko okruženje dizajnirano za bezbedno čuvanje i obradu osetljivih podataka, kao što su korisničke lozinke, kriptografski ključevi i biometrijske informacije. Kriptografski ključevi nikada ne napuštaju ovaj prostor u nešifrovanom obliku.

3.4.1.2. Sigurnosni moduli

Samsung Knox predstavlja sveoubuhvatan bezbednosni okvir integrisan u određene Android uređaje, koji obezbeđuje zaštitu od hardverskog do aplikacionog sloja sistema. U kontekstu otključavanja ekrana ima bitnu ulogu u zaštiti biometrijskih i autentifikacionih podataka, koji se čuvaju u izolovanim okruženjima kao što su Knox Vault i TrustZone.

3.5. Prikupljanje dokaza

Tipovi metoda koje se koriste u istragama su ručna ekstrakcija, logička ekstrakcija i fizička ekstrakcija, prikupljanje iz oblaka i prikupljanje iz celularne mreže. Svaka od ovih metoda ima svoje prednosti i ograničenja, a izbor odgovarajuće tehnike uslovljen je nivoom pristupa samom uređaju kao i tipom informacija koji je od interesa za istragu.

3.5.0. Forenzičke metode akvizicije podataka

Metoda ručnog prikupljanja je izvodljiva jedino ako je uređaj funkcionalan i otključan. Logičko prikupljanje ne zahteva dublju intervenciju sa hardverom ili operativnim sistemom uređaja, ali je najčešće ograničena ukoliko nije moguće dobiti autorizaciju za pristup. Fizička ekstrakcija je najbolji izbor kada je potreban potpun pristup svim podacima ali je za njeno izvođenje potrebna visoka ekspertiza.

3.5.0.0. JTAG i Chip-off

Tehnike fizičke ekstrakcije poput JTAG i Chip-off mogu se koristiti i za pokušaj zaobilaženja zaključanog ekrana. JTAG se koristi za ekstrahovanje slike memorije, uključujući i fajlove koji sadrže podatke o PIN-u, šablonu ili enkripcionim ključevima

3.5.0.1. Android Debug Bridge

Android Debug Bridge (ADB) je alat koji omogućava komunikaciju između računara i mobilnog uređaja putem USB-a i pod specijalnim uslovima, može da se korsti za otključavanje ekrana starijih Android modela kada šifra i šablon nisu poznati.

3.5.1. Evolucija zaštitnih mehanizama

Evolucija zaštitnih mehanizama na mobilnim uređajima značajno je ograničila forenzičke mogućnosti za prikupljanje dokaza i tradicionalni pristupi postaju sve manje efikasni u situacijama kada je uređaj zaključan i ne postoji mogućnost saradnje sa korisnikom.

4. METODE OTKLJUČAVANJA TELEFONA

U ovom odeljku razmatrane su različite metode otključavanja telefona koji koriste biometrijsku autentifikaciju, šablon ili lozinku. Primeri uspešnih slučajeva otključavanja, opisani u ovom delu rada, preuzeti su iz stručne literature i naučnih radova.

4.1. Spoofing metode i biometrijska autentifikacija

Spoofing napad je napad tokom koga se lažira identitet odnosno biometrijski parametar korisnika kako bi se prevario sistem i dobio pristup uređaju. U cilju sprečavananja ovih napada, savremeni operativni sistemi se oslanjaju na različite tehnike detekcije živosti.

4.1.0. Vrste senzora za prepoznavanje otiska prsta

Postoje tri vrste senzora koji se ugrađuju u mobilne uređaje i to su: optički, kapacitativni i ultrazvučni. Svaki od ovih senzora ima različit stepen otpornosti na prepoznavanje replika.

4.1.0.0. Metode kloniranja otisaka prstiju

Istraživači i forenzički stručnjaci pokazali su da je moguće izraditi veštačke otiske prstiju korišćenjem materijala kao što su silikon, lateks ili čak glina.

4.1.1. Tehnologija skeniranja lica

Postoje dve vrste skenera za prepoznavanje lica koji koriste mobilni uređaji i to su 2D i 3D skeneri. Najpoznatiji primer je Apple Face ID koji koristi 3D tehnologiju skeniranja.

4.1.1.0. Metode replikacije facijalnih karakteristika

Spoofing napad nad ovim mehanizmom se može izvesti upotrebom fotografija visoke rezolucije, videa ili čak izradom kvalitetnih 3D maski.

4.1.2. Tehnologija skeniranja mrežnjače

Skeniranje mrežnjače koristi infracrvenu kameru za skeniranje jedinstvenog obrasca šarenice oka korisnika. Ova autentifikacija se pokazala manje praktična u poređenju sa drugim oblicima autentifikacije zbog čega je i slabo zastupljena kod komercijalnih telefona.

4.2. Brute-force metode oktljučavanja ekrana

PIN-ovi, lozinke i obrasci mogu biti podložni napadima grubom silom koji se svode na sistematsko nagađanje i isprobavanje svih mogućih kombinacija unosa. Sprovođenje ove tehnike je skoro nemoguće na modernim uređajima zbog mehanizama koji definišu dozvoljen broj pokušaja i korišćenje vremenskog intervala za kašnjenje između pogrešnih unosa.

4.2.0. Alati za automatizaciju procesa pogađanja

Kako bi se proces ručnog unosa kredencijala optimizovao i kako bi se izbegle bezbednosne prepreke, u forenzičkim istragama se koriste alati poput IP Box 3, Atiny85 i GrayKey-a.

4.3. Zaobilaženje zaključanog ekrana Android uređaja 5.1 i starijih verzija

U ovom odeljku su opisane metode zaobilaženja zaključanog ekrana koje se oslanjaju na iskorišćavanje slabosti u korisničkom interfejsu starijih modela.

4.3.0. Postupak “rušenja“ ekrana

Ovaj napad zahteva fizički pristup uređaju i uključuje unos dugog niza znakova u polje za lozinku putem funkcije hitnog poziva, što dovodi do rušenja interfejsa i omogućava pristup bez unosa ispravne lozinke.

4.3.1. Forgot pattern/password opcija

Na uređajima sa operativnim sistemom Android verzije 4.4, funkcija forgot password/pattern omogućavala je korisniku da resetuje šablon za otključavanje pomoću validnog email naloga.

4.4. Otključavanje mobilnih uređaja preko Cloud naloga

Ovaj metod je koristan jer omogućava otključavanje mobilnih uređaja direktno sa cloud naloga korisnika.

4.5. Metode socijalnog inženjeringa

Ova tehnika se ne oslanja na hardverske ili softverske alate, već na manipulaciju ljudima kako bi se dobio pristup osetljivim informacijama. Zastupljeni oblici ovog napada su phishing, vishing, pretexting.

4.5.0. Shoulder surfing

Shoulder surfing je tehnika koja podrazumeva direktno ili inderektno posmatranje korisnika dok unosi svoje podatke za autentifikaciju kao što su pin kod, šifra ili obrazac.

5. DEMONSTRACIJA I DISKUSIJA

U ovom odeljku su demonstrirane odabrane metode otključavanja ekrana na testnim uređajima.

5.1. Izrada silikonskog otiska prsta

Proces kreiranja lažnog otiska se odvijao u nekoliko iteracija, tokom kojih je izveden zaključak da je najkvalitetnija replika u ovom eksperimentu dobijena upotrebom silikona RTV i parafin voska. Na slici 1. prikazan je krajnji rezultat replikacije koja iako verna originalu, nije uspela da prevari sistem koji koristi kapacitativni senzor.

Slika 1. *Otisak prsta od silikona*

Zagrejan vosak je prvo izliven u plitku metalnu posudu i posle izvesnog vremena kada je podloga postala dovoljno tvrda, na njoj se mogao utisnuti šablon pravog otiska. Zatim je dodat tanak sloj jestivog skroba i silikona. Nakon što se smesa stvrdnula, posuda je ubačena u ključalu vodu kako bi se materijali prirodno razdvojili usled razlike u temperaturi topljenja silikona i voska.

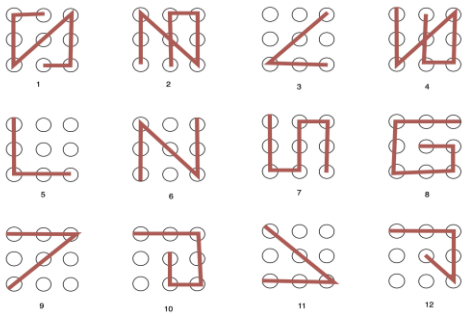
5.2. Otključavanje fotografijom lica

Testni uređaj Redmi 7A sa Android verzijom 9 koji koristi 2D tehnologiju skeniranja lica, uspešno je otključan sa slikom korisnika. Slika je bila visokog kvaliteta i lice je jasno prikazano na fotografiji.

Rezultati studije [3] pokazali su veliki potencijal upotrebe veštačke inteligencije i slika dostupnih na društvenim mrežama korisnika kako bi se izradio virtuelni avatar koji može da prevari sisteme sa naprednijim tehnologijama skeniranja.

5.3. Rekonstrukcija šablona

Sprovedena anketa imala je za cilj da prikupi testni skup šablona zaključavanja ekrana kako bi se uočile i uporedile karakteristike najčešćih odgovora. Na slici 2. je prikazan deo rezultata.



Slika 2. Deo rezultata ankete

Od 42 uzorka, 11 figura je bilo jedinstveno nad datim skupom što dovodi do zaključka da je četvrtina ispitanika koristila šablon koji se nije našao kao zaštita ekrana ni na jednom uređaju ostalih korisnika. Trend koji je primećen kod ovih podataka jeste da u približno 80% slučajeva korisnici su izabrali početak crteža da bude u levom gornjem uglu i da su oblici L i N bili učestal izbor.

5.4. Otključavanje preko ADB-a

Otključavanje ekrana Redmi 7A, ili bilo kog drugog uređaja, preko ADB-a nije uvek izvodljivo jer zavisi od više preduslova koji moraju biti ispunjeni pre nego što dođe do zaključavanja. Na telefonu je potrebno omogućiti režim za programere i uspostaviti komunikaciju radne stanice i testnog uređaja putem verifikacije računara prilikom povezivanja. Zatim ako je ovo ispunjeno, potrebno je izvršiti set komandi za brisanje određenog fajla koji bi omogućio pristup telefonu bez unosa lozinke. Ovaj postupak je bio učinkovit na starijim verzijama Android-a ali usled napredne zaštite sistema uvedene od verzije 9 i kasnije, ovaj metod se pokazao neučinkovit na testnom uređaju.

3. ZAKLJUČAK

Rad ukazuje da nema univerzalnog rešenja za otključavanje mobilnih uređaja. Umesto toga, potrebno je proceniti svaki slučaj ponaosob, uz izbor metode koja balansira između efikasnosti, očuvanja dokaza, tehničkih mogućnosti i pravne validnosti.

Predlozi za dalja istraživanja se odnose na primenu saznanja i tehnologija iz oblasti mašinskog učenja i veštačke inteligencije u predikciji biometrijskih šablona i obrazaca korisničkog ponašanja, kao i u optimizaciji napada grubom silom. Može se izdvojiti i orijentacija ka razvoju naprednih softverskih alata za analizu ranjivosti u operativnim sistemima, uključujući istraživanje procesa Secure Boot-a i USB protokola, uz potencijalnu primenu mašinskog učenja za automatizovano otkrivanje slabosti.

7. LITERATURA

[1] Prof. dr Stevan Gostojić, Fakultet tehničkih nauka, Novi Sad, predmet Digitalna forenzika, Uvod u digitalnu forenziku (predavanja 2023, lekcija 1)

[2] Mobile operating systems popularity statistics, https://gs.statcounter.com/os-market share/mobile/worldwide (pristupljeno u septembru 2025.)

[3] Yi Xu, True Price, Jan-Michael Frahm, Fabian Monrose. Virtual U: Defeating Face Liveness Detection by Building Virtual Models from Your Public Photos, USENIX Security Symposium 2016.

Kratka biografija:

Jelena Petrić rođena je u Novom Sadu 2000. Diplomirala je na Fakultetu tehničkih nauka 2023. godine. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva odbranila je 2025. godine

kontakt: jelena.petric@uns.ac.rs

Зборник радова Факултета техничких наука, Нови Сад

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/33BE33Stojkic

ЈЕЗИК СПЕЦИФИЧАН ЗА ДОМЕН ВИЗУАЛИЗАЦИЈЕ ЈЕЗИКА

A DOMAIN-SPECIFIC LANGUAGE FOR THE VISUALIZATION OF LANGUAGES

Радиша Стојкић, *Факултет техничких наука, Нови Сад*

Област –ПРИМЕЊЕНЕ РАЧУНАРСКЕ НАУКЕ И ИНФОРМАТИКА

Кратак садржај – *У овом раду описан је текстуални доменски језик намењен дефинисању визуализације језика креираних помоћу библиотеке textX. Представљене су употребљене технологије и окружења, као и основни појмови везани за језике специфичне за домен и графове. Дат је преглед постојећих решења у овој области, као и кључни делови имплементације самог viewX језика и екстензије за Visual Studio Code. Рад садржи и практичан пример употребе и визуалног приказа, а завршава се освртом на постигнуте циљеве и могућности даљег развоја решења.*

Кључне речи: *Језици специфични за домен, екстензије за VS Code, граф, визуелизација*

Abstract – *This paper describes a textual domain-specific language designed for defining the visualization of languages created using the textX library. It presents the technologies and environments used, as well as the basic concepts related to domain-specific languages and graphs. An overview of existing solutions in this field is provided, along with key parts of the implementation of the viewX language and the Visual Studio Code extension. The paper also includes a practical usage example and a visual representation, concluding with a review of the achieved goals and possibilities for further development of the solution.*

Keywords: *Domain-Specific Languages, VS code extensions, graph, visualization*

1. УВОД

Програми и модели писани са текстуалном синтаксом су често непрегледни и тешко разумљиви. Додатни проблем се јавља ако људи нису упознати са самим појмовима и доменом проблема и језиком на којем је писан програм. Тада је тешко утврдити све односе и кључне детаље у програму и моделу. Један од начина да се превазиђу потешкоће у представи модела и програма јесте визуализација, помоћу које можемо представити све кључне елементе и њихове међусобне релације, што доприноси потпунијем и бољем разумевању модела.

У зависности од мотива и потреба, разликују се различите визуелне репрезентације. Неке од њих су графикони, дијаграми, мреже, графови и многе друге. Главни изазов код визуализације модела је како јасно приказати структуре података у облику графа. Једно од основних питања јесте генеричност решења, како би се омогућило приказивање различитих модела писаних на истом језику уз могућност да корисник манипулише самим приказом датог модела.

Основни мотив овог рада јесте надоградња језика за опис визуализације модела писаних на језицима који су развијени помоћу *textX* библиотеке, као и поједностављење перспективе самог модела. Текстуални језик *ViewX* развијен је као одговор на потребу за једноставнијом и јаснијом визуализацијом доменских модела. Језик је реализован кроз мастер рад на Факултету техничких наука у Новом Саду. *ViewX* служи за директно дефинисање правила визуализације елемената, а развијен је помоћу *textX* библиотеке. Помоћу правила се могу описати особине, структура и елементи графа, као и начин на који се они приказују. Комплетно решење је остварено као екстензија за *Visual Studio Code* едитор, потпомогнуто *third-party* библиотекама, *Python* окружењем и употребом *viewX* језика 0.

Једна од најважнијих предности решења описаног у овом раду јесте концепт одвајања садржаја модела од начина његове визуализације. Садржај чини саму структуру и податке модела, док начин приказа дефинише *viewX* модел, омогућавајући флексибилност приказа. То значи да се један модел може приказати на више различитих начина у зависности од потребе, док се истовремено више различитих модела може представити на исти начин. Овакав приступ обезбеђује прилагодљивост и универзалност у визуализацији, што је кључ за широку примену и лакшу интерпретацију сложених система.

2. ТЕОРИЈСКЕ ОСНОВЕ

Језици специфични за домен (*DSL*) су програмски језици дизајнирани за решавање проблема у јасно дефинисаној области. Они нуде једноставнију синтаксу и већу ефикасност унутар свог домена, али су мање флексибилни од језика опште намене (*GPL*), који су универзални и примењиви у различитим контекстима.

Примери *DSL* језика су *HTML* (структура веб страница) и *SQL* (рад са базама података), док су типични *GPL* језици *Python* и *Java*. Предност *DSL*-а је у прилагођености терминологији и потребама

НАПОМЕНА
Овај рад проистекао је из мастер рада чији ментор је био др Игор Дејановић, ред. проф.

специфичне области, што убрзава развој и смањује потребу за техничким знањем. Мана је мања могућност поновне употребе и теже одржавање у односу на *GPL* језике 0.

Граф је математичка структура *G = (V, E)* где је *V* скуп чворова, а *E* скуп ивица. Може бити усмерен, неусмерен или пондерисан. У рачунарству се користи за моделовање односа, попут мрежних веза, друштвених мрежа или структуре података. Визуализација графова олакшава анализу, али изазови укључују претрпаност приказа и распоред чворова, што се решава филтрирањем, хијерархијским приказима и алгоритмима за аутоматски распоред. Репрезентација графова обично се остварује помоћу матрице суседства (брза провера везе, већа потрошња меморије), листе суседства (ефикасна за ретке графове) или листе грана. Графови налазе примену у рачунарству кроз оптимизацију мрежа, анализу друштвених мрежа, базе података и алгоритме претраге 0.

3. КОРИШЋЕНЕ СОФТВЕРСКЕ ТЕХНОЛОГИЈЕ

Visual Studio Code (*VSC*) је едитор кода који подржава велики број програмских језика и екстензија. Уз интегрисани терминал, *Git* подршку и аутоматско довршавање кода, омогућава продуктиван рад на различитим платформама (*Windows, MacOS, Linux*). *Marketplace* нуди хиљаде екстензија, као што су *Prettier, ESLint* и *Python*, које проширују функционалност едитора 0.

Python је флексибилан и лак за учење програмски језик, погодан за објектно-оријентисано, функционално и процедурално програмирање. Динамичка типизација и богата стандардна библиотека, уз подршку за пакете попут *NumPy, SciPy* и *Pandas*, чине *Python* погодним за научне апликације, машинско учење и анализу података. Једноставност синтаксе омогућава брзо развијање и тестирање апликација.

textX је алат за креирање језика специфичних за домен (*DSL*) у *Python*-у. Користи *PEG* граматику и *Arpeggio* парсер за аутоматско генерисање парсера и мета-модела. Омогућава модуларизацију граматика, конфигурацију парсера, постпроцесирање модела и визуализацију *GraphViz*-ом. Захваљујући једноставности, *textX* је погодан за развој сложених апликација и симулација са прилагођеним језицима.

Cytoscape.js је библиотека за визуализацију и манипулацију графовима у веб окружењу. Подржава усмерене, неусмерене и тежинске графове, интерактивност, динамичке измене и *WebGL/Canvas* рендеровање. Омогућава прилагођавање стила чворова и ивица, као и проширење функционалности додатцима за анализу мрежа и алгоритме.

Jinja2 је *Python* шаблонски механизам за генерисање динамичког *HTML*-а и других текстуалних формата. Подржава *placeholder*-е, петље, услове, филтере и макрое, што олакшава одвојен приказ података од логике апликације. Брзо рендерује сложене шаблоне и интегрише се са веб оквирима као што су *Flask* и *Django*.

4. ПРЕГЛЕД СТАЊА У ОБЛАСТИ

Савремени софтверски системи постају све сложенији, што захтева напредне алате за визуализацију и управљање њиховим компонентама. Алати за визуализацију омогућавају боље разумевање података и ефикасније управљање сложеним структурама. Од графичких едитора до система за визуализацију великих графова, они пружају креирање прилагођених интерфејса, визуелизацију података у реалном времену и интеракцију са елементима. У овом поглављу представљени су *Graphiti*, *Sirius*, *Gephi* и *Neo4j Bloom*.

4.1. *Graphiti*

Graphiti је оквир за визуализацију графичких едитора у *Eclipse* окружењу, омогућавајући лако креирање графичких објеката као што су чворови и ивице. Интеграција са *EMF*-ом омогућава трансформацију текстуалних или табеларних података у интерактивне графичке приказе. Примена је у *UML* дијаграмима, *BPMN* процесима и визуализацији архитектуре софтвера 0.

4.2. *Sirius*

Sirius омогућава креирање прилагођених графичких модела специфичних за домен (*DSL*) у *Eclipse* окружењу. Корисници могу дефинисати статичке и динамичке моделе, као и интерактивне приказе у реалном времену. Флексибилан је за рад у индустрији, инжењерингу, финансијама и научним истраживањима, уз интеграцију са *EMF* и *GMF* 0.

4.3. *Gephi*

Gephi је *open - source* алат за визуализацију и анализу графова, посебно погодан за социјалне мреже, биоинформатику и велике податке. Омогућава интерактивну манипулацију графовима, прилагођавање боја, величине и облика чворова и анализа метрика као што су централност и кластеризација. Ограничење је обрада веома великих графова 0.

4.4. *Neo4j Bloom*

Neo4j Bloom визуелно приказује графове у *Neo4j* бази података. Пружа динамичан и интерактиван приказ, са могућношћу прилагођавања изгледа графова и истраживања релација између чворова. Најпогоднији је за податке већ у *Neo4j*, а за напредну аналитику потребан је *Cypher* 0.

5. ИМПЛЕМЕНТАЦИЈА

Ово поглавље описује кључне делове имплементације екстензије за *Visual Studio Code*, која омогућава интерактивну визуализацију графова и управљање моделима. Решење је реализовано у *TypeScript*-у, интегрисано са *Python* модулима преко *textX* и *Python-shell*, што омогућава креирање граматике *viewX* језика за опис *layout*-а и елемената графа.

Архитектура решења комбинује више компоненти:

- ***BrowserSync*** – синхронизује измене у реалном времену између више клијената, обезбеђујући једноличан приказ.

- ***Socket.io*** – омогућава двосмерну комуникацију, при чему се клијенти региструју у собе и добијају све поруке о стању графа, чиме се постиже слабо спрегнута и флексибилна клијент-сервер архитектура.
- ***Cytoscape.js*** – за приказ и интеракцију са графовима, укључујући креирање подграфова сложених чворова и прилагођене облике чворова.

Python скрипте користе *Jinja2* шаблоне за генерисање *HTML*-а и *layout*-а графова, а интеграција са екстензијом омогућава њихово динамичко извршавање. Решење је *cross-platform* (*Windows* и *Linux*) са динамичким препознавањем ОС-а ради активирања одговарајућег виртуелног окружења.

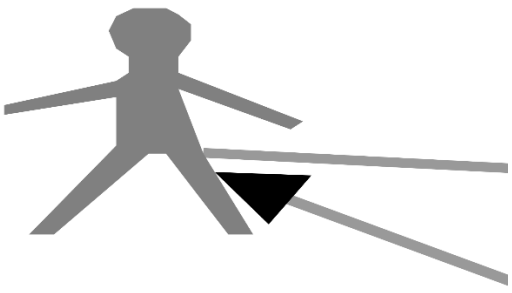
Кроз проширења *viewX* језика и интеракцију са *Cytoscape.js*, екстензија омогућава оптимизован приказ графова са скривањем и приказом потомака сложених чворова, дефинисање облика чворова полигонима и флексибилну визуелизацију која реагује на измене модела у реалном времену. Ово омогућава дугорочну функционалност и прилагодљивост решења различитим сценаријима рада.

6. ДЕМОНСТРАЦИЈА

Проширења реализована у *viewX* екстензији представљена кроз пример примене *mini BPMN* језик за моделовање пословних процеса. *BPMN (Business Process Model and Notation)* је стандардизован графички нотацијски систем за опис пословних токова, омогућавајући комуникацију између техничких и пословних корисника. Циљ примера је демонстрирати могућности екстензије за визуализацију модела креираних помоћу библиотеке *textX*.

ViewX омогућава дефинисање облика чворова кроз координате полигона, боју и величину. На пример, ентитет *Human* може се описати као полигон са дефинисаним тачкама, бојом и стрелицама ка другим ентитетима (Слика 1):

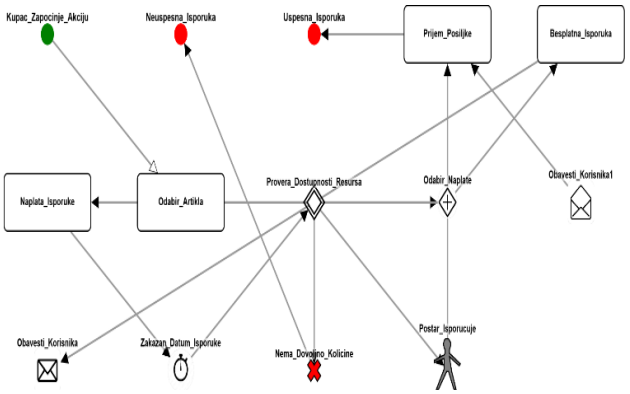
```
view Human as polygon {
    shape-polygon-points: 0.04 -0.88, 0.09 -0.84, ..., -0.09 -0.88
    background: gray
    width: 400
    height: 400
    link {to: Human.action {arrow: black 2 triangle}}
}
```



Слика 1. *Визуализација чвора Human као полигон*

Екстензија омогућава дефинисање распореда елемената графа. На пример, grid layout организује елементе у мрежу (Слика 2):

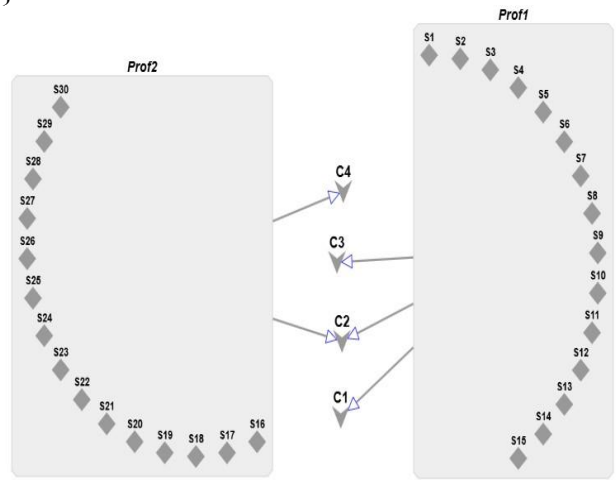
```
layout {
    name: grid
    rows: 3
    cols: 5
    animate: true
    animationDuration: 300
    fit: true
    avoidOverlap: true
}
```



Слика 2. *Grid layout графа*

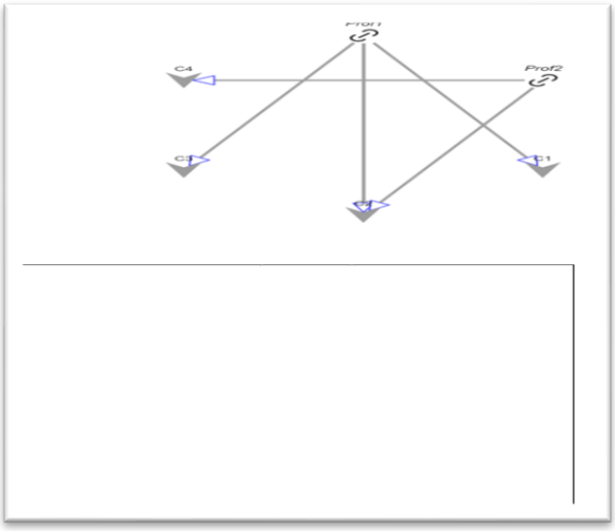
ViewX подржава приказ сложених чворова и њихових потомака. Кључна реч *show* приказује све потомке сложеног чвора, као што је приказано на слици 3.

```
view Student as diamond child of Professor show {
    label: Student.name {font: 15}
}
```

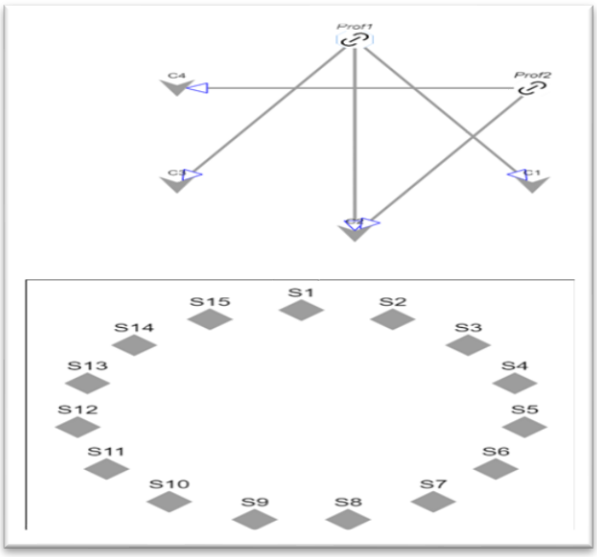


Слика 3. *Сложени чворови са потомцима*

Ради оптимизације, сложени чворови могу се приказати без потомака (Слика 4), а подграф потомака се генерише у посебном прозору испод главног графа (Слика 5).



Слика 4. *Визуализација сложеног чвора без потомака*



Слика 5. *Визуализација подграфа сложеног чвора*

7. ЗАКЉУЧАК

Да би се ово решење издвојило као универзално и свеобухватно, неопходна је његова додатна надоградња. Тренутно је визуализација реализована као *2D* просторна репрезентација. Као једно од могућих унапређења издваја се увођење *3D* просторне репрезентације графа, која би омогућила интуитивније разумевање и већи ниво детаља. На тај начин би се превазишла ограничења која произилазе из дводимензионалног приказа.

У оквиру овог рада реализовано је решење које се издваја од постојећих тиме што нуди различите нивое визуализације и степене детаљности графа. Омогућено је оптимизовање приказа дефинисањем сложених чворова и креирањем подграфова, што знатно смањује количину информација у једном приказу и тиме повећава прегледност. Посебан значај има увођење механизама који кориснику омогућавају да самостално управља приказом, било кроз приказ само сложених чворова, било кроз њихово експлицитно развијање у посебним подграфовима.

Креирање прилагођених облика чворова, спецификација изгледа и анимација представљају додатне функционалности које обогаћују решење. Ове особине, у комбинацији са пажљивим избором новијих и стабилнијих верзија компоненти, доприносе дуговечности и поузданости решења. Оптимизација приказа графа постигнута је не само кроз управљање нивоом детаља, већ и кроз примену алгоритама распоређивања елемената који омогућавају већу читљивост и структурисаност визуализације.

Решење пружа висок степен прилагодљивости и слободе у креирању и управљању визуализацијом. У будућности, проширење функционалности кроз графичку синтаксу и интеграцију више различитих библиотека за приказ графова представљаће значајан корак ка унапређењу.

Иако решење не подржава *3D* форму, оно се истиче флексибилношћу, стабилношћу и могућношћу оптимизације, пружајући корисницима алат за дубље разумевање сложених структура и њихових релација.

8. ЛИТЕРАТУРА

[1] Мастер рад – "Подршка визуализацији језика креираних употребом *textX* библиотеке у оквиру *Visual Studio Code* едитора", Даниел Купчо, Факултет техничких наука у Новом Саду, 2018

[2] *Martin Fowler:* "*Domain-Specific Languages*", *Addison-Wesley Signature Series*, 2010

[3] "*Graph theory*", *Wikipedia*, слободна енциклопедија, https://en.wikipedia.org/wiki/Graph_theory, (приступљено у децембру 2024.)

[4] https://en.wikipedia.org/wiki/Visual_Studio_Code, (приступљено у децембру 2024.)

[5] https://eclipse.dev/graphiti/, (приступљено у децембру 2024.)

[6] Sirius - https://www.eclipse.org/sirius/ (приступљено у децембру 2024.)

[7] https://gephi.org/users/supported-graph-formats/, (приступљено у децембру 2024.)

[8] https://neo4j.com/product/bloom/, (приступљено у децембру 2024.)

Кратка биографија:

Радиша Стојкић је рођен 24.01.2000. године у Зворнику, Босна и Херцеговина. Основну школу "Десанка Максимовић" у Челопеку завршио је 2015. године. Исте године уписује гимназију, општи смер, у ЈУ Средњошколски центар "Петар Кочић" Зворник. Године 2019. гимназију завршава као носилац Вукове дипломе. Факултет техничких наука, смер Рачунарство и аутоматика, уписује 2019. године у Новом Саду, где 2023. године завршава основне академске студије.

Зборник радова Факултета техничких наука, Нови Сад

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/33BE34Kanjuh

ОПТИМИЗАЦИЈА ЧЕТБОТА КОРИШЋЕЊЕМ ТРАНСФОРМЕР МОДЕЛА

OPTIMIZATION OF A CHATBOT USING TRANSFORMER MODELS

Нађа Кањух, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Циљ рада је оптимизација конверзацијског четбота за домен осигурања коришћењем алгоритама природне обраде језика и трансформер модела, конкретно BERT модела, ради бољег разумевања језика и термина специфичних за осигурање. У истраживању је четбот обучен и тестиран у поређењу са LSTM моделом, при чему су експериментални резултати показали да BERT модел даје боље резултате због своје способности разумевања ширег контекста у упитима корисника. Ова способност омогућава четботу да прецизније интерпретира сложене и контекстуално богате упите, што је посебно важно за пружање тачних и поузданих информација у домену осигурања.*

Кључне речи: *четбот, трансформер модели, BERT, LSTM, природна обрада језика, велики језички модели*

Abstract – *The goal of this thesis is to optimize a conversational chatbot for the insurance domain using natural language processing algorithms and transformer model, specifically the BERT model, to enhance understanding of language and insurance-specific terminology. In the research, the chatbot was trained and tested in comparison with the LSTM model, with experimental results showing that the BERT model performs better due to its ability to understand broader context in user queries. This capability allows the chatbot to interpret complex and contextually rich queries more accurately, which is especially important for providing precise and reliable information in the insurance domain.*

Keywords: *chatbot, transformer models, BERT, LSTM, natural language processing – NLP, large language models*

1. УВОД

Имплементација четбота доноси низ предности. Првенствено, четботоби могу побољшати корисничко искуство пружањем брзих и тачних одговора на захтеве клијената, било да се ради о захтевима за информацијама, обради одређених или пружању подршке у реалном времену. Поред тога, они омогућавају уштеду ресурса и времена, аутоматизујући рутинске задатке и смањујући потребу за директним људским ангажовањем.

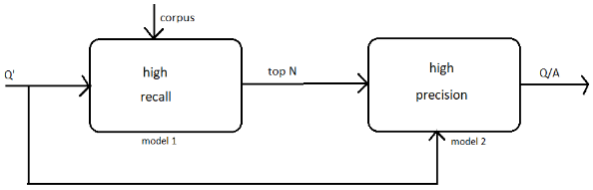
НАПОМЕНА:

Овај рад проистекао је из мастер рада чији је ментор био др Милан Сечујски, ред. професор.

С обзиром на то да *LSTM*, као врста рекурентних неуронских мрежа, често није у стању да ухвати довољно широк контекст, све је већи интерес за њихову замену ради унапређења перформанси конверзацијских система. У овом раду испитује се побољшање конверзацијског система применом трансформер модела, конкретно *BERT* модела. *BERT* модел пружа напредне функције кодирања са двосмерном пажњом, што омогућава боље разумевање значења речи у контексту целе реченице.

Циљ овог рада је да покаже како се систем може унапредити коришћењем савремених трансформер модела, као и да упореди резултате *LSTM* и *BERT* модела.

Иако савремени приступи попут *Retrieval-Augmented Generation – RAG* омогућавају четботовима да приступе обимним базама знања и генеришу ажурне одговоре, њихова примена може бити ограничена доступношћу квалитетно структуираних података и прихватљивим временом одзива. Због тога је у овом раду изабран приступ двостепеног одлучивања, који комбинује модел високог одзива (енг. *high recall model*) и модел високе прецизности (енг. *high precision model*). Модел високог одзива је дизајниран да идентификује што већи број релевантних кандидата за одговор на упит из корпуса. Углавном се за издвајање релевантних кандидата бирају статистички модели због своје брзине и ефикасности. Са друге стране, циљ модела високе прецизности јесте да идентификује најадекватније питање од свих издвојених најбољих кандидата и да се на тај начин кориснику пружи одговарајући одговор. У овом кораку често се користе архитектуре попут *LSTM* и *BERT* модела. На слици 1 се налази блок дијаграм двостепног одлучивања.



Слика 1. *Блок дијаграм двостепеног одлучивања*

2. ДЕФИНИЦИЈЕ

У овом сегменту је дато објашњење најбитнијих теоријских појмова и алгоритама који се користе за израду четбота.

2.1. Модел високог одзива

У наставку ће укратко бити описани основни кораци за имплементацију модела високог одзива, обрада текста и векторизација.

2.1.1. Стеминг

Стеминг (енг. *Stemming*) је процес обраде текста који подразумева свођење рели на основни или коренски облик. Циљ стеминга је поједностављивање речи уклањањем префикса или суфикса, али уз задржавање основног облика. Неке од предности стеминга су смањење речника, побољшање претраге и брже процесирање. Са друге стране неке од мана јесу потенцијално прекомерно редуковање, затим изазови стеминг методе за поједине језике, као свођење различитих речи на исти корен што често може довести до конфузије у задацима као што су генерисање текста.

2.1.2. Лематизација

Лематизација (енг. *Lemmatization*) је процес свођења речи на основни облик или облик из речника, познат као лема. За разлику од стеминга који једноставно уклања префиксе или суфиксе, лематизација узима у обзир морфолошку анализу речи и има за циљ одређивање њеног основног облика. Предности коришћења лематизације јесу боље задржавање семантичког значења, побољшање претраге и боље разумевање језика. Са друге стране, лематизација може бити поприлично сложена и захтева по питању ресурса, а такође може доћи и до губитка одређених морфолошких информација приликом свођења речи.

2.1.3. *N*-грами

N-грами представљају узастопне низове од *n* елемената одређеног скупа, при чему елемент може бити реч, карактер или чак фонема, у зависности од конкретног проблема. У контексту обраде природног језика, *n*-грами се најчешће односе на секвенце речи. *N*-грами бележе локални заједнички контекст речи и могу пружити корисне увиде у односе и обрасце унутар текста. Један од недостатака јесте тај да коришћење *n*-грама може довести до реткости података, а због фиксне величине прозора ограничени да ухвате шири контекст.

2.1.4. Учесталост термина

Учесталост термина (енг. *Term frequency – TF*), представља један од најједноставнијих начина векторизације текстуалних података. *TF* је основни концепт у обради природног језика који мери фреквенцију појављивања термина у документу. Када се примењује *TF* као метода векторизације, сваки документ се представља као вектор чија дужина одговара броју јединствених термина у целокупном корпусу. Сваки елемент вектора представља фреквенцију одговарајућег термина унутар датог документа. Овај начин векторизације је погодан због своје једноставности и флексибилности у примени, а такође је добар и за истицање значаја термина унутар документа. Нека ограничења ове методе јесу та што додељује веће тежине речима које се често појављују, а такође не узима у обзир шири контекст, па може доћи до губитка информација.

2.1.5. Учесталост термина – инверзна учесталост на нивоу документа

Учесталост термина - инверзна учесталост на нивоу документа (енг. *Term Frequency – Inverse Document Frequency, TF-IDF)* је метода векторизације текста која комбинује локалну фреквенцију термина (енг. *Term Frequency - TF*) са глобалном значајношћу термина у корпусу (енг. *Inverse Document Frequency – IDF*). *TF* компонента наглашава значајне термине унутар појединачног документа. Са друге стране, *IDF* компонента има улогу да смањи важност честих термина који се појављују у целом корпусу. Ова метода може бити погодна за идентификацију кључних речи у документу, а такође и смањује важност неинформативних термина који се често појављују у целом корпусу. Неки од недостатака ове методе јесте не узимање у обзир редоследа речи у документу, као и недостатак контекста и семантичког разумевања. Ова метода не узима у обзир сложеније језичке елементе попут синонима, вишезначности речи итд.

2.1.6. *Word2Vec*

Основни циљ *Word2Vec* методе јесте извлачење семантичких информација и других битних особина из речи. *Word2Vec* користи контекстне информације речи како би научио њихове векторске репрезентације. Идеја је да сличне речи буду представљене у векторском простору једна близу друге. *Word2Vec* ради по сличном принципу као аутокодери. Постоје два приступа рада са *Word2Vec* методом. Први јесте коришћење претрениранog модела чиме се добија велики број речи, али недостатак је што често модел није специјализован за домен над којим се ради. Други приступ подразумева тренирање модела, чиме добијамо модел који је специјализован за одређени домен, али мана овог приступа јесте недовољно речи. Коришћење *Word2Vec* доводи до тога да вектори генерисани помоћу овог алгоритма имају својство да задрже семантичке односе између речи, такође омогућава ефикасно рачунање сличности речи помоћу метрика удаљености. Недостатак је тај што квалитет векторских репрезентација зависти од величине и квалитета корпуса, као и то што *Word2Vec* узима у обзир само локални контекст око речи.

2.2. Модел високе прецизности

У наставку ће бити описана архитектура трансформера, њихова предност у односу на рекурентне неуронске мреже, а такође ће бити и описан начин функционисања *BERT* модела.

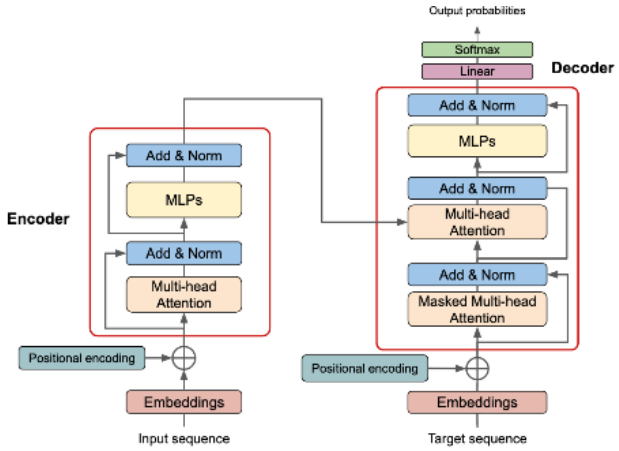
2.2.1. Предности трансформера у односу на рекурентне неуронске мреже

Трансформери (енг. *Transformers*) су један од најсавременијих и најнапреднијих модела вештачке интелигенције који омогућавају разумевање и генерисање текста на начин сличан човеку. Пре њих, рекурентне неуронске мреже су податке обрађивале

секвенцијално и то је довело до проблема споре обуке и губитка информација на дужим секвенцама. Трансформери решавају ове проблеме користећи механизам пажње, који омогућава моделу да истовремено обрађује све делове текста и фокусира се на кључне елементе без обзира на њихову удаљеност [1].

2.2.2. Архитектура трансформера

Када је у питању архитектура трансформера, они се састоје од два основна дела: кодера и декодера. Оба поменута дела се састоје од неколицине других слојева, а у наставку рада биће детаљно описан сваки део заједно са његовим слојевима. На слици 2 је приказана архитектура трансформер модела.



Слика 2. *Архитектура трансформера*

Један слој кодера (енг. *encoder*) се састоји од слоја вишеглавне пажње, два слоја нормализације, двослојне потпуно повезане мреже, а уз све то постоје и резидуалне конекције. Оваквих слојева има *N.* Улаз у кодер јесу векторисане речи са додатим позиционим ембединзима који су неопходни како би модел имао информацију о позицијама речи.

Механизам пажње (енг. *attention mechanism*) је главни елемент трансформер архитектуре. Он омогућава моделу да обради више пажње на део улазних података који садржи значајније информације и да посвети мање пажње остатку улаза [2]. У пракси, обично се не добијају добри резултати коришћењем једног слоја, па се из тог разлога обично израчунава више слојева пажње у паралели и резултати се конкатенирају. Вишеглавна пажња (енг. *Multi-head attention*) представља механизам који служи за поменуто паралелно израчунавање.

Слојеви нормализације помажу у одржавању стабилности током тренинга, смањују проблем нестајања или експлодирања градијента, помажу бољој генерализацији модела, а такође помажу моделу да се прилагоди различитим скалама улаза [2]. Резидуалне конекције имају кључну улогу у побољшању ефикасности тренирања и смањењу ризика од нестајања или експлодирања градијента у дубоким моделима. Оне омогућавају да све информације остану очуване током тренирања.

Потпуно повезана мрежа има за циљ да додатно обради податке и научи комплксне и нелинеарне везе између података. Она се углавном састоји од два скривена слоја и *ReLU* активационе фунцкије.

Излаз из слоја кодера је скуп вектора, при чему сваки представља улазни низ обогаћен контекстом. Овај излаз се користи као улаз у декодер трансформера.

Декодер (енг. *Decoder*) је практично исти кодер, осим што садржи додадни слој вишеглавне пажње која ради над излазом кодера. Циљ декодера јесте да комбинује излаз кодера са циљном секвенцоми прави предвиђања, односно да предвиди следећи токен. Број слојева декодера је углавном исти као и број слојева кодера.

2.2.3. Велики језички модели

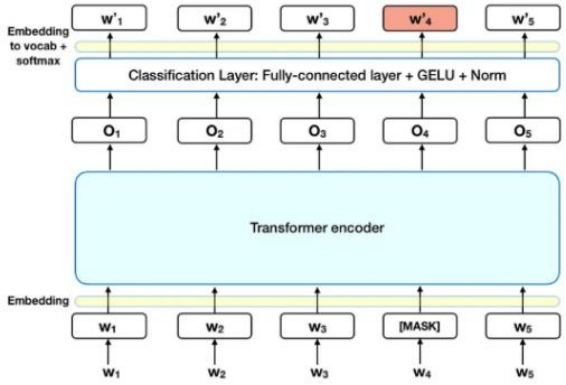
Велики језички модели (енг. *Large language Models – LLM*) су револуционисали комуникацију са системима машинског учења омогућавајући обављање задатака у реалном времену на основу природног језика. Ови модели, који садрже милијарде параметара, тренирани су на огромним количинама текстуалних података, чиме стичу широко разумевање света. За обуку се користи самонадгледано учење (енг. *self-supervised learning*), које омогућава рад са неозначеним подацима и побољшава способност генерисања текста [2]. Након почетког трениранја модели могу решавати задатке користећи *zero-shot* и *few-shot* техника, које им омогућавају да извршавају задатке са мало или без примера. Инжењеринг упита (енг. *prompt engineering*) и техника ланац мисли (енг. *chain of thought – CoT*) омогућавају прецизније одговоре и боље решавање сложенијих проблема. За специфичне задатке у различитим областима, ови модели захтевају додатно дообучавање на доменским подацима.

2.2.4. *BERT*

*BERT (*енг. *Biderectional Encoder Representations from Transformer*) је врста трансформера, где је кључна иновација примена бидирекционог тренирања. Овај приступ тренирања модела у оба смера доводи до резултата који показују да језички модел на овај начин може имати дубље разумевање контекста и тока језика него модел трениран у једном смеру. За разлику од основног модела трансформера, *BERT* користи само кодер. Кодер *BERT* модела обрађује читаву секвенцу речи одједном. Ова карактеристика омогућава моделу да разуме реч у контексту свих околних речи, како с лева, тако и с десна. *BERT* користи две специфичне стратегије тренирања: моделовања маскираног језика и предвиђање наредне реченице [3].

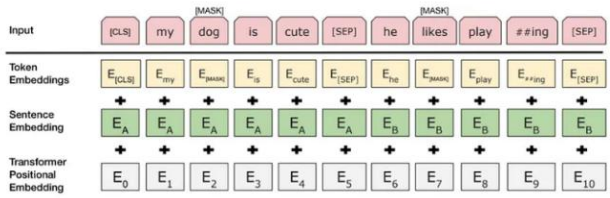
Моделовање маскираног језика подразумева да се 15% речи у низу замени са [MASK] токеном, а модел затим покушава да предвиди те речи на основу контекста. Око 80% замењених речи је означено са [MASK], 10% се замењује случајним речима, а преосталих 10% остаје нетакнуто. Овај приступ помаже моделу да научи контекст, не само предвиђање маскираних речи, тако што га приморава да разуме како се маскиране и немаскиране речи повезују. Предвиђање се постиже додавањем класификационог слоја и применом *softmax* функције како би се израчунала вероватноћа за сваку реч. Ова

метода омогућава моделу боље разумевање контекста, иако тренирање може бити спорије. На слици 3 је приказана описана стратегија тренирања.



Слика 3. *Стратегија тренирања помоћу моделовања маскираног језика*

Стартегија предвиђања следеће реченице у *BERT* моделу користи парове реченица како би модел научио да одреди да ли друга реченица у пару следи прву у оригиналном тексту. У половини случајева, друга реченица је заиста следећа, док у осталих 50% случајева друга реченица је насумично изабрана и обично није повезана с првом. За обраду ових парова, [CLS] токен се додаје на почетак, [SEP] на крај сваке реченице, и сваком токену се додаје ембединг који означава његов положај и реченицу којој припада. Ова стратегија помаже *BERT* моделу да боље разуме контекст и односе између реченица. На слици 4 приказана је обрада улаза за описану стратегију.



Слика 4. *Обрада улаза за стратегију предвиђања следеће реченице*

Да би *BERT* утврдио да ли је друга реченица повезана са првом, цео низ пролази кроз трансформер, при чему се излаз [CLS] токена трансформише у вектор за класификацију. *Softmax* затим израчунава вероватноћу повезаности реченица. Моделовање маскираног језика и предвиђање следеће реченице тренирају се истовремено како би се минимизовала заједничка функција губитка. Овакво удружено тренирање омогућава *BERT* моделу боље разумевање контекста и структуре језика.

Како би *BERT* био успешан при решавању различитих језичких задатака, углавном је довољно додати мали слој на основни модел.

3. КОРИШЋЕНИ СКУПОВИ ПОДАТАКА

За потребу тестирања различитих модела у комуникацији са корисницима, коришћен је скуп података који се састоји од питања и одговора везаних за осигурање. Овај скуп података добијен је уз допуштење клијента. За дотренирање *BERT* модела искоришћен је *Semantic Textual Similarity Benchmark* скуп података [4], који је уједно и један од најчешће коришћених скупова за проналажење сличних питања.

4. РЕЗУЛТАТИ И ДИСКУСИЈА

Најбољи резултати при тестирању модела високог одзива добијени су коришћењем претрениранog *Word2Vec* модела уз сумирање вектора речи и еуклидску удаљеност за мерење сличности.

За модел високе прецизности, чије је унапређење уједно и циљ овог истраживања, спроведено је и квантитативно и квалитативно тестирање перформанси сијамског *LSTM* и *BERT* модела на истом тест скупу који је споменут у претходном поглављу. Квантитативна анализа је показала да *LSTM* често није успевао да обухвати све кључне аспекте корисничког упита, што је доводило до враћања делимично релевантних или потпуно нетачних питања, до је *BERT* постигао значајно боље метрике прецизности и тачности. Квалитативно посматрано, *BERT* је, захваљујући бољем разумевању контекста, доследно проналазио семантички најприближнија питања, што је резултирало знатно адекватнијим и поузданијим одговорима у реалним сценаријима.

5. ЗАКЉУЧАК

Имплементација трансформер модела за четбота у домену осигурања значајно побољшава тачност, брзину и укупно корисничко искуство у односу на претходно имплементирани *LSTM* приступ. Трансформери омогућавају бољу обраду комплексних и дугих упита, што је од великог значаја како у осигурању, тако и у другим областима. У циљу даље применљивости у реалним условима и даљем унапређењу система могуће је интегрисати *RAG* методологију, а такође и применити различите механизме заштите података како би се добило још напредније, флексибилније и безбедније решење којим би се додатно унапредило корисничко искуство.

6. ЛИТЕРАТУРА

[1] Ferrer, J. (2024) How transformers work: A detailed exploration of Transformer architecture, DataCamp.

[2] Nyandwi, J. (2023) Ai Research Blog - The Transformer Blueprint: A holistic guide to the transformer neural network architecture, Deep Learning Revision

[3] Horev, R. (2018) Bert explained: State of the art language model for NLP, Medium.

[4] https://paperswithcode.com/dataset/sts-benchmark

Кратка биографија:

Нађа Кањух рођена је у Врбасу 2000. године. Мастер рад на Факултету техничких наука из области Електротехнике и рачунарства одбранила је 2025. године.

контакт: nadjakanjuh00@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.4
DOI: https://doi.org/10.24867/33BE35Trivunovic

SAMOOBNAVLJAJUĆI KOD NA AWS PLATFORMI

SELF HEALING CODE ON AWS PLATFORM

Dragana Trivunović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad istražuje mogućnost rešenja samoobnavljajućeg koda koji koristi AWS platformu i generativnu veštačku inteligenciju. Mogućnosti koje otvara složeno rešenje koda koji se samoobnavlja se ogleda u smanjivanju utrošenog vremena razvojnog inženjera i brzom detektovanju greške u softveru i njenom otklanjanju. Glavni servis koji je centar arhitekture ovog rešenja je servis generativne veštačke inteligencije koji se koristi za sugestiju, dopunu, ispravku, pojašnjenje, optimizaciju, transformaciju i poboljšanje softverskog rešenja.*

Ključne reči: *Cloud, AWS, Generativni AI*

Abstract – *This paper explores the feasibility of a self-healing code solution utilizing the AWS platform and generative artificial intelligence. The potential of a complex self-healing code solution lies in reducing the time spent by software engineers, enabling quick error detection in the software, and facilitating error correction. The core service at the heart of this architecture is a generative AI service, used for suggesting, completing, correcting, clarifying, optimizing, transforming, and enhancing the software solution.*

Keywords: *Cloud, AWS, Generative AI*

1. UVOD

Era naprednog razvitka veštačke inteligencije dovela je do, nekad nezamislivih, tehničkih rešenja. Primena ovakvih rešenja je u velikoj meri olakšala niz zadataka u razvoju softvera. Optimizacija i automatizacija su ključni faktori koji izdvajaju rešenja sa primenom veštačke inteligencije smanjujući cenu realizacije, utrošeno vreme i potrebne resurse za razvoj softvera. Pomoć koju veštačka inteligencija pruža prilikom razvoja i održavanja softverskih rešenja omogućila je smanjenje vremena koje je potrebno da se pronađe i popravi greška (engl. bug). U trci za napredna rešenja veštačke inteligencije i mašinskog učenja, sa svojim inovativnim pristupom, ističe se AWS (Amazon Web Services) kompanija. Načini korišćenja veštačke inteligencije unutar njihovih proizvoda dovode do poboljšanog iskustva korisnika, povećane produktivnosti zaposlenih, kreativnijeg marketinga i optimizovanih procesa unutar kompanije. Ovaj rad istražuje AWS Self healing kod, analizirajući njegovu arhitekturu, funkcionalnosti i praktičnu primenu.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Vukmirović, red. prof.

Na AWS platformi se mogu koristiti već istrenirani modeli veštačke inteligencije i infrastruktura napravljenih servisa. Osim fokusa na veštačku inteligenciju koji čini samo jedan deo ponuđenih rešenja, na AWS platformi postoji čitav niz softverskih rešenja koji čine infrastrukturu AWS-ovog „oblaka“ (engl. Cloud). Usluge koje se pružaju su skladištenje i arhiviranje podataka, procesuiranje podataka, izgradnja API interfejsa, privatni VPN, elastično balansiranje opterećenja nadolazećeg saobraćaja na veb sajtu, razne analitike i metrike podataka korišćenjem Big Data tehnologija, implementiranja politike sigurnosti, identiteta i saglasnosti, itd.

U ovom radu će biti obrađeni neki od servisa koji su iskorišćeni u stvaranju rešenja za kod koji ima mogućnost samoobnavljanja, detekcije grešaka i podizanja ispravaka koda u izvorni kod.

2. AMAZON WEB SERVICES

U poređenju sa klasičnim računarstvom, AWS kao pionir ideje računarstva u oblaku 2000-ih godina, donosi novosti koje su zauvek izmenile računarstvo. Najveći benefiti se ogledaju u skalabilnosti, gde manje firme imaju mogućnost da plaćaju samo onoliko koliko im u datom trenutku treba, te sa povećanjem obima posla povećaju i obim AWS-ovih usluga. U prošlosti su firme koristile privatne centre podataka sa privatnim hardverom i serverima, koji su u poređenju sa servisima na „oblaku“ bile izuzetno skuplje. Skaliranje na AWS-u može da bude i automatizovano i brzo, te je ovakav vid poslovanja mnogo pogodniji. U današnje vreme sve modernizovane firme i kompanije koriste rešenja računarstva u oblaku [5].

2.1. AWS servisi

AWS takođe nudi svojim korisnicima razvijeni sistem za implementiranje politike sigurnosti, saglasnosti i identiteta. Postoje servisi za nadzor sistema koji povećavaju sigurnost. Štiti svoje korisnike od ispada sistema stavljajući akcenat na pouzdanost svojih usluga. AWS garantuje dostupnost korisničkih aplikacija pružanjem usluga na 80 zona dostupnosti i više od 25 globalnih regija, što ga čini odličnim izborom za kompanije koje posluju globalno. Pruža razvojne alate za softver koji su laki za korišćenje i omogućava brzo podizanje korisničkih aplikacija i veb sajtova. Postoje nekoliko tipova razvojih modela računarstva u oblaku [3].

U modelu infrastrukture kao servisa, korisniku je omogućeno korišćenje računarske

infrastrukture u vidu virtuelne platforme kao što su virtuelne mašine i serveri zaskladištenje i upravljanje podacima. Primeri modela infrastrukture kao servisa u AWS-u su servisi poput Elastic Computing 2, Simple Storage Service – S3, Relational Database Service – RDS.

Softver kao servis je model koji se fokusira na dostavljanje aplikacije korisniku preko interneta. AWS ne pruža gotova Softver kao servis rešenja, ali pruža mogućnost svojim korisnicima da naprave svoje samostalne aplikacije bazirane na ovom modelu.

Platforma kao servis je model koji pruža platformu, alate i razvojna okruženja softverskim inženjerima za razvoj softverskih rešenja. Primer je Content Delivery Network – CDN servis koji predstavlja mrežu povezanih servera koji ubrzavaju učitavanje veb stranica za aplikacije koje su orijentisane na podatke. Takav jedan servis je AWS-ov CloudFront. Platforma kao servis je implementirana kao model u servisima za balansiranje protoka podataka. Elastic Load Balancing – ELB je servis koji omogućava distribuciju opterećenja aplikacije na serverima zarad poboljšanja skalabilnosti aplikacije. Još jedan primer ovog modela se nalazi u Amazon CloudWatch servisu koji služi za monitoring podataka i aplikacije, preko konstrukcije log datoteka i raznih metrika.

Back-end kao servis je model koji pomaže razvojnim inženjerima da se fokusiraju na front-end deo aplikacije. U ovom modelu je serverski deo aplikacije već napravljen i umesto korisnika upravlja bazom podataka, skladištenjem podataka, autentikacijom korisnika, notifikacija na aplikaciji, veb-hostinga, itd. Primer ovog modela na AWS-u je AWS Amplify. Ovaj model omogućava front-end inženjerima da konstruišu full-stack aplikacije.

Funkcija kao servis je model računarstva u oblaku gde je fokus stavljen na pokretanje funkcija kao odgovor na neki događaj, bez potrebe da se konfiguriše kompleksna infrastruktura virtuelnih mašina i upravljanje operativnim sistemima i procesima održavanja veb servera. Sa ovim modelom, veb server se može podeliti na funkcionalnosti koje se mogu automatski skalirati. Sa ovim modelom, plaća se samo kada se funkcionalnost iskoristi, a ne po satu ili količinskom najmu kao kod ostalih modela. Primer ovog modela je AWS Lambda servis. Ovaj servis izvršava kod na visoko dostupnoj infrastrukturi za obradu podataka i vrši administraciju svih resursa za obradu, uključujući i administraciju operativnog sistema i servera. Takođe, ovaj servis automatski skalira i potražuje potrebne resurse za obradu podataka.

3. AWS SELF HEALING CODE

Pod pritiskom brzog razvoja, programeri i menadžeri su često suočeni sa izborom trošenja vremena na popravke grešaka koda koji je već u produkciji, ili ostavljanjem greške i problema u „back log“-u gde se gomila kao tehnički dug. Prvi izbor znači više utrošenog vremena i novca, a drugi izbor dovodi do nezadovoljnih klijenata i lošeg korisničkog iskustva.

Rešenje do kojeg su došli inženjeri iz Amazon AWS kompanije je kod koji ima mogućnost introspekcije na osnovu prijavljenih grešaka na aplikaciji, mogućnost ispravke koda i podizanja te ispravke na izvorni kod. Ključni deo ovog rešenja je generativni model veštačke inteligencije koji nadopunjuje i ispravlja kod.

3.1. ARHITEKTURA REŠENJA

Ovo rešenje je napravljeno kombinovanjem Amazon CloudWatch-a, AWS Lambda i Amazon Bedrock servisa kako bi kreirali sveobuhvatan sistem koji automatski otkriva i popravlja greške radi poboljšavanja pouzdanosti aplikacije i celokupnog korisničkog iskustva. U ovom sistemu, ispravljač grešaka se povezuje sa CloudWatch evidencijom zapisa grešaka aplikacije preko Lambda pretplate. Svi zapisnici koji sadrže greške aplikacije šalju se na obradu, gde Lambda funkcija kreira prompt, uključujući praćenje steka (engl. „stack“) i relevantne datoteke koda, a zatim ga šalje u Amazon Bedrock (Claude v1 model) da generiše ispravke koda. Izmenjeni kod se zatim šalje u kontrolu izvora (Git) i kreira zahtev za povlačenje za pregled i primenu [6].

Ovakvo rešenje pruža niz funkcionalnosti: Automatsko otkrivanje praćenja steka (greške): Implementira pretplate na CloudWatch evidencije za automatsko filtriranje i otkrivanje tragova steka; Praćenje grešaka: Automatski prati stanje obrade grešaka u Amazon DynamoDB; Deduplikacija: Deduplikuje tragove steka da bi se izbegla suvišna obrada; Kreiranje zahteva za povlačenje: Integriše se sa sistemima kontrole izvora za automatsko kreiranje zahteva za povlačenje, koji uključuju ispravke grešaka

Postoji nekoliko servisa koji se koriste u ovom rešenju a koji su spomenuti u prethodnim poglavljima. Amazon CloudWatch Core servis pruža podatke evidencije grešaka za problematičan izvorni kod. AWS Lambda Core servis implementira automatizaciju za brzo generisanje i interakciju sa BedRock-om. Amazon Bedrock Core servis pruža širok skup mogućnosti za izgradnju generativnih AI aplikacija koje analiziraju, popravljaju i vraćaju izmenjeni problematičan izvorni kod. Amazon Simple Queue Service (Amazon SQS) servis pruža grupnu obradu i kontrolu istovremenosti za Lambda funkciju. Amazon DynamoDB Core servis čuva trag steka koda sa evidencijama grešaka i prenosi podatke. Pomoćni servis Amazon Systems Manager čuva tajne parametre u skladištu [12].

3.1.1. Amazon CloudWatch

Amazon CloudWatch je servis koji nadgleda aplikacije, reaguje na promene u performansama, optimizuje korišćenje resursa i pruža uvid u operativno zdravlje. Prikupljanjem podataka preko AWS resursa, CloudWatch daje uvid u performanse celog sistema i omogućava korisnicima da podese alarme, automatski reaguju na promene i steknu jedinstven pogled na operativno zdravlje. CloudWatch je kao servis napravljen po modelu Platforma kao servis.

CloudWatch funkcioniše tako što akumulira metriku i evidenciju iz AWS resursa, analizira te podatke i pruža vizuelne prikaze i upozorenja. Sastoji se od 3 glavne komponente: deo za praćenje i prikupljanje metrike koji prikuplja podatke iz resursa, dugoročno skladištenje tih podataka u CloudWatch-u, kao i vizuelizacije i alarme na osnovu uskladištenih podataka.

3.1.2. AWS Lambda

Moguće je koristiti AWS Lambda za pokretanje koda bez obezbeđivanja ili upravljanja serverima.

Lambda pokreće kod na računarskoj infrastrukturi visoke dostupnosti i obavlja svu administraciju računarskih resursa, uključujući održavanje servera i operativnog sistema, obezbeđivanje kapaciteta i automatsko skaliranje i evidentiranje. Sa Lambda-om, sve što treba da uradite je da unesete svoj kod u jednom od jezika izvođenja koje Lambda podržava [1].

Kod se organizuje u Lambda funkcije. Lambda usluga pokreće funkciju samo kada je to potrebno i automatski se podešava. Plaća se samo vreme koje je utrošeno na pokretanje i izvršavanje funkcije — nema naknade kada kod nije pokrenut.

Lambda je idealna računarska usluga za scenarije aplikacija koje treba brzo da se povećaju i smanje na nulu kada nisu tražene.

Kada se koristi Lambda, odgovornost je na inženjeru samo za njegov kod. Lambda upravlja računarskom flotom koja nudi balans memorije, CPU-a, mreže i drugih resursa za pokretanje vašeg koda. Pošto Lambda upravlja ovim resursima, nije moguće prijaviti se da bi se izračunale instance ili prilagodili operativni sistem na predviđenim vremenima izvođenja. Lambda obavlja operativne i administrativne aktivnosti u korisničko ime, uključujući upravljanje kapacitetom, praćenje i evidentiranje korisničkih Lambda funkcija [2].

3.1.3. Amazon DynamoDB

DynamoDB je bez-serverska, NoSQL, potpuno upravljana baza podataka sa jednocifrenim milisekundnim performansama na bilo kojoj skali. Kao baza podataka bez servera, plaća se samo ono što se iskoristi. DynamoDB skalira na nulu, nema hladnih pokretanja, nema nadogradnje verzije, nema prozora za održavanje, nema zastoja održavanja. Ova baza podataka nudi širok skup bezbednosnih kontrola i standarda usklađenosti. Za globalno distribuirane aplikacije, DynamoDB globalne tabele su multi-regionalna, multiaktivna baza podataka sa SLA dostupnošću od 99,999% i povećanom otpornošću. Pouzdanost DynamoDB-a je podržana upravljanim rezervnim kopijama i oporavkom u trenutku. Sa DynamoDB tokovima, moguća je izgradnja aplikacija vođenih događajima bez servera [4] [9].

3.1.4. Amazon SQS

Amazon Simple Queue servis (Amazon SQS) nudi siguran, izdržljiv i dostupan red podataka koji omogućava integraciju i odvojanje distribuiranih softverskih sistema i komponenti. Amazon SQS nudi uobičajene konstrukcije kao što su redovi za „mrtve“ poruke i oznake za alokaciju troškova. Pruža generički API za veb usluge kojima se može pristupiti koristeći bilo koji programski jezik koji podržava AWS SDK [10].

Amazon SQS razdvaja i skalira distribuirane softverske sisteme i komponente kao uslugu čekanja. Obično obrađuje poruke preko jednog pretplatnika, što je idealno za tokove posla gde su prevencija narudžbine i gubitka kritični. Za širu distribuciju, integracija Amazon SQS-a sa Amazon SNS-om omogućava razgranati obrazac za razmenu poruka, efektivno prosleđujući poruke većem broju pretplatnika odjednom [8].

3.1.5. Amazon Bedrock

Amazon Bedrock je servis koji omogućava da dostupnost osnovnih modela veštačke inteligencije visokih performansi (engl. Foundation Model) iz vodećih startapa veštačke inteligencije i Amazon za korisničku upotrebu putem objedinjenog API-ja. Moguće je birati između širokog spektra osnovnih modela veštačke inteligencije kako bi se pronašao model koji je najprikladniji za korisnikov slučaj upotrebe. Amazon Bedrock takođe nudi širok skup mogućnosti za izgradnju generativnih AI aplikacija sa bezbednošću, privatnošću i odgovornom veštačkom inteligencijom. Koristeći Amazon Bedrock, moguće je da se lako eksperimentiše nad i procenjuju osnovni modeli za korisničke slučajeve upotrebe, privatno da se prilagođavaju korisničkim podacima koristeći tehnike kao što su fino podešavanje i proširena generacija preuzimanja (RAG) i prave agenti koji izvršavaju zadatke koristeći sisteme korisnikovog preduzeća i izvore podataka.

Amazon Bedrock je rešenje bez servera, gde je moguće privatno prilagoditi osnovne modele sopstvenim podacima i lako i bezbedno ih integrisati i primeniti u aplikacije koristeći AWS alate bez potrebe za upravljanjem infrastrukturom i njenom izmenom [11].

4. PRINCIPI ARHITEKTURE SAMOOBNAVLJAJUĆEG KODA

AWS-ovi inženjeri su pioniri sistema za kod koji se sam obnavlja, pronalazi greške (engl. bag) i sam podiže zahteve za ispravku originalnog koda.

Arhitektura ovog rešenja pomaže softverskim kompanijama da postave sistem za otkrivanje i evidentiranje grešaka, generisanje ispravki grešaka i kreiranje zahteva za promenom koda. Svaka kompanija koja kreira softver neizbežno mora da uravnoteži rešavanje grešaka, a istovremeno se takmiči sa pritiskom razvoja proizvoda i funkcionalnosti. Greške mogu odvratiti pažnju programera, pogoršati korisničko iskustvo i uzrokovati pogrešne pokazatelje. Ovo idejno rešenje pomaže softverskim kompanijama da implementiraju automatizovani sistem koji otkriva i ispravlja greške kako bi poboljšao pouzdanost aplikacija i poboljšao celokupno korisničko iskustvo.

Rešenje za kod koji se sam popravlja je zasnovano na 6 osnovnih stubova najboljih arhitektonskih praksi za dizajniranje sistema u oblaku.

AWS Well-Architected Framework opisuje ključne koncepte, principe dizajna i najbolje arhitektonske prakse za projektovanje i pokretanje radnih opterećenja u oblaku. Ovi principi i koncepti su opisani detaljnije u radu [7].

8. ZAKLJUČAK

Inovativan pristup inženjera iz kompanije Amazon nam je donelo rešenje koje će, ukoliko se dobro podesi i iskoristi, dovesti do smanjivanja tehničkog duga i do olakšavanja i ubrzavanja toka razvoja aplikacije. Način na koji je iskorišćen generativni model veštačke inteligencije je

doveo do automatizacije velikog dela posla programera, a to je pronalaženje i ispravak grešaka u kodu.

Ovo rešenje smanjuje troškove razvoja softverskih rešenja, poboljšava produktivnost timova, obezbeđuje visok nivo sigurnosti i pouzdanosti,

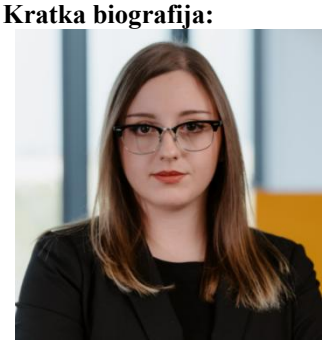
Potrebno je imati u vidu da je ovo rešenje dostupno samo u određenim AWS regionima i da se troškovi mogu značajno razlikovati u zavisnosti od obima upotrebe i konkretnih potreba korisnika. Pažljivo planiranje i praćenje troškova, uz korišćenje AWS alata kao što je Cost Explorer, omogućava korisnicima da maksimiziraju koristi od ovog sistema uz minimalne finansijske rizike

Na kraju, ovakve vrste rešenja predstavljaju budućnost razvoja softvera, gde automatizacija i veštačka inteligencija igraju ključnu ulogu u poboljšanju kvaliteta softverskih rešenja i korisničkog iskustva. Ovakve tehnologije su već krenule da transformišu način na koji timovi softverskih inženjera rade, smanjujući tehnički dug i omogućavajući brže i efikasnije rešavanje problema.

9. LITERATURA

[1] Funkcija kao servis [na mreži]. Dostupno na: https://www.ibm.com/topics/faas (Pristupljeno u junu 2024. godine)
[2] Lambda Servis [na mreži]. Dostupno na: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (Pristupljeno u junu 2024. godine)
[3] Proizvodi AWS-a [na mreži]. Dostupno na: https://aws.amazon.com/products/ (Pristupljeno u junu 2024. godine)
[4] AWS načini skladištenja podataka [na mreži]. Dostupno na: https://www.educba.com/aws-storage-services/ (Pristupljeno u junu 2024. godine)
[5] O bazama podatala [na mreži]. Dostupno na: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/database.html (Pristupljeno u junu 2024. godine)
[6] Self healing code uputstvo [na mreži]. Dostupno na: https://aws.amazon.com/solutions/guidance/self-healing-code-on-aws/ (Pristupljeno u junu 2024. godine)
[7] Principi arhitekture na AWS-u [na mreži]. Dostupno na: https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitepapers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc (Pristupljeno u junu 2024. godine)
[8] Cloudwatch [na mreži]. Dostupno na: https://aws.amazon.com/cloudwatch/ (Pristupljeno u junu 2024. godine)
[9] DynamoDB [na mreži]. Dostupno na: https://aws.amazon.com/dynamodb/ (Pristupljeno u junu 2024. godine)
[10] Simple Queue Service [na mreži]. Dostupno na: https://docs.aws.amazon.com/AWSSimpleQueueService/ (Pristupljeno u junu 2024. godine)
[11] Bedrock [na mreži]. Dostupno na: https://docs.aws.amazon.com/bedrock/ (Pristupljeno u junu 2024. godine)
[12] Self healing code [na mreži]. Dostupno na: https://aws-solutions-library-samples.github.io/ai-ml/self-healing-code-on-aws.html (Pristupljeno u junu 2024. godine)

Kratka biografija:



Dragana Trivunović rođena je 21. oktobra 1998. godine u Sremskoj Mitrovici. Osnovnu školu "Čaki Lajoš" završila je u Bačkoj Topoli, a gimnaziju, opšti smer, je završila u srednjoj školi "Dositej Obradović" u Bačkoj Topoli. Školske 2017/2018 godine upisuje Fakultet tehničkih nauka u Novom Sadu, na studijski program Primenjeno softversko inženjerstvo. Osnovne akademske studije završila je školske 2022/2023, posle kojih upisuje master studije, takođe, na studijskom programu Primenjeno softversko inženjerstvo.

Зборник радова Факултета техничких наука, Нови Сад

UDK: 621.3
DOI: https://doi.org/10.24867/33BE36Nesic

РАЗВОЈ СОФТВЕРА НА *RISC-V* АРХИТЕКТУРИ СА ФОКУСОМ НА *RPC* ИМПЛЕМЕНТАЦИЈУ

RISC-V SOFTWARE DEVELOPMENT WITH FOCUS ON RPC IMPLEMENTATION

Марија Нешић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Овај рад представља детаљан опис софтверске архитектуре и имплементације механизма удаљених позива процедура (Remote Procedure Call – RPC) у оквиру уграђеног система заснованог на RISC-V архитектури. Рад документује комплетан процес подизања софтвера на новом чипу, од првобитне иницијализације хардвера до успостављања функционалне комуникације са спољашњим системима.*

Кључне речи: *Развој софтверског окружења, RISC-V, FreeRTOS, RPC*

Abstract – *This thesis presents a detailed description of the software architecture and implementation of the Remote Procedure Call (RPC) mechanism within an embedded system based on the RISC-V architecture. It documents the complete process of bringing up software on a new chip, from the initial hardware initialization to establishing functional communication with external systems.*

Keywords: *Software Bring-Up, RISC-V, FreeRTOS, RPC*

1. УВОД

Идеја удаљеног позива процедуре (*RPC*), коју су први пут увели *Birrell* и *Nelson* 1984. године, представљала је велики корак напред у дистрибуираној обради. Она омогућава да се процедуре у процесима на удаљеним рачунарима позивају као да су процедуре у локалном адресном простору. *RPC* систем управља основним механизмима — кодирањем и декодирањем података, слањем порука и обезбеђивањем да се позив понаша као регуларан позив функције [1].

RPC модел комуникације изабран је као основни механизам интеракције између *host*-а и *Wi-Fi HaLow* чипа заснованог на *SweRV EH1* архитектури.

Основна мотивација за овај избор лежи у потреби да се сложене операције са стране *host*-а иницирају једноставним функцијским позивима, а да се при томе изолују детаљи *low-level* комуникације, синхронизације и управљања меморијом.Имплементирани *RPC* механизам припада категорији *Hardware-Level RPC* (*Chip-to-Chip*, *On-Board*), конкретно *SPI/I2C RPC* типу. Иако чип интерно користи *AXI* протокол, класификација се одређује према спољашњем интерфејсу и комуникационим карактеристикама. Из перспективе спољашњег *host*-та, комуникација се реализује кроз стандардне *SPI* трансакције где *host* (*master*) шаље команде које функционишу као удаљени позиви процедура ка *Wi-Fi HaLow* чипу (*slave*). Што се тиче комуникационог модела нуди комбинацију синхроне и асинхроне комуникације.

Циљ рада је приказ конкретног решења развијеног у индустријском контексту за *Wi-Fi HaLow* чип, које омогућава комуникацију између чипа и спољашњег *host* уређаја.

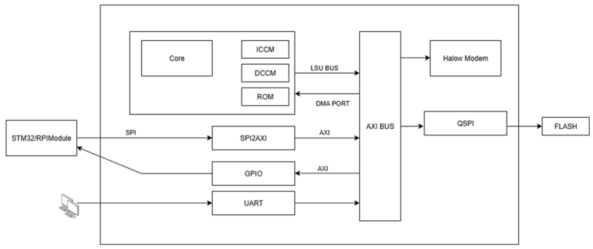
Главни проблеми решавани у оквиру овог пројекта обухватају иницијализацију и *bring-up* софтвера на *SweRV EH1* процесору заснованом на *RISC-V* архитектури, портовање оперативног система *FreeRTOS* на циљну платформу, развој *RPC* механизма за транспарентну комуникацију између уграђеног чипа и екстерног *host*-а, имплементацију *SPI* комуникационог слоја са синхронизацијом и управљањем прекидима, развој *host* стране *RPC* комуникације, као и оптимизацију ресурса и меморијских захтева система.

2. СОФТВЕРСКА И ХАРДВЕРСКА ПЛАТФОРМА

Цео систем се састоји из два процесора, од којих је један *HaLow modem* процесор који нуди *Wi-Fi* функционалности. Да би та функционалност остала изолована, остале функционалности су део интерног *host* процесора, *SweRV EH1*. У овом раду је описан његов развој. Чип представља специјализовану *Wi-Fi HaLow* станицу која прима команде од спољашњих *host*-ова преко *SPI* интерфејса.

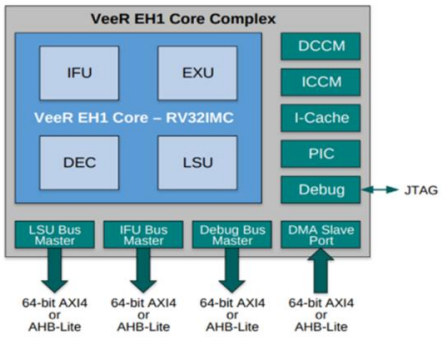
НАПОМЕНА:

Овај рад проистекао је из мастер рада чији ментор је био др Предраг Теодоровић, ванр. проф.



Слика 1. *Дијаграм система*

На слици 1 је приказан цео систем заједно са процесором.



Слика 2. *Архитектура SweRV EH1 Core Complex-a*

Као што је приказано на слици Слика 2, *SweRV EH1 Core Complex* садржи *RV32IMC* језгро са пратећим компонентама као што су *DCCM*, *ICCM*, *I-Cache*, *PIC* и *Debug* интерфејс, као и више *AXI4*/*AHB-Lite* интерфејса за спољашње конекције.

Систем осим интерног *host* и *HaLow modem* процесора садржи још и *SPI-to-AXI*, *UART* и *QSPI* модуле. Комуникација од екстерног *host-a* до чипа се одвија преко *SPI-to-AXI* модула који омогућава комуникацију тако што *SPI* сигнали од спољашњег контролера (нпр. *STM32*) бивају конвертовани у *AXI* протокол, прослеђени на *AXI* магистралу, а затим преко *DMA* порта приступају *DCCM* меморији за размену података. Комуникација од чипа ка екстерном *host-y* функционише у супротном смеру - сигнал који процесор емитује преко *LSU BUS-a* се рутира кроз *AXI* магистралу и појављује се на *GPIO* портовима за даље прослеђивање екстерном *host-y*.

FreeRTOS је изабран као оперативни систем за *Wi-Fi HaLow* чип, што представља логичан избор за ову врсту уграђених апликација. У питању је оперативни систем у реалном времену отвореног кода, дизајниран посебно за микроконтролере и уграђене процесоре са ограниченим хардверским ресурсима.

Избор шеме *heap_4* заснован је на њеном идеалном односу између детерминисаног понашања, отпорности на фрагментацију и компатибилности са *RISC-V (SweRV EH1)* окружењем.

Подешавање окружења укључује инсталацију *RISC-V GNU Toolchain*-а. *RISC-V GNU Toolchain* представља комплетно окружење за развој, које обухвата *GCC* компајлер, *Binutils*, *GDB debugger* и *Newlib C* стандардну библиотеку [2]. За подршку *SweRV EH1* језгру, алат је конфигурисан са *rv32imc* архитектуром и *ilp32 ABI*-јем.

Приликом успешног компајлирања генеришу се следећи изворни фајлови *firmware.elf*, *firmware.disasm* и *firmware.map*. За дебаговање кода коришћен је *J-Link* адаптер преко *JTAG* интерфејса.

3. ПОРТОВАЊЕ *FREERTOS*-А НА *SWERV EH1*

Портовање оперативног система *FreeRTOS* на процесор *SweRV EH1* архитектуре *RISC-V* обухвата адаптацију његових основних механизама, као што су замена контекста, руковање прекидима и конфигурација системског тајмера, у складу са специфичностима циљне хардверске платформе.

Портовање *FreeRTOS*-а на *SweRV EH1* почиње укључивањем изворних фајлова који садрже основне *RTOS* функционалности и специфичног *RISC-V* адаптационог слоја. Затим се конфигурише *trap handler*, иницијализује се систем прекида и ради се интеграција екстерних прекида.

FreeRTOS кернел се ослања на периодичне прекиде тајмера (*RTOS tick*) за пребацивање контекста између *task*-ова, праћење временских интервала и управљање *timeout* механизмима.

У функцији *vPortSetupTimerInterrupt()* се врши конфигурација тајмера. У портовању *FreeRTOS*-а за *SweRV EH1* језгро, *vPortSetupTimerInterrupt()* користи *Low Power Controller* (*LPC*) као извор системског тајмера. *LPC* има сопствени интерни тајмер који је у овом случају конфигурисан да активира прекид на сваких 1000 микросекунди.

Такође се у *linker* скрипти дефинише *IRQ stack* који се користи за управљање прекидима.

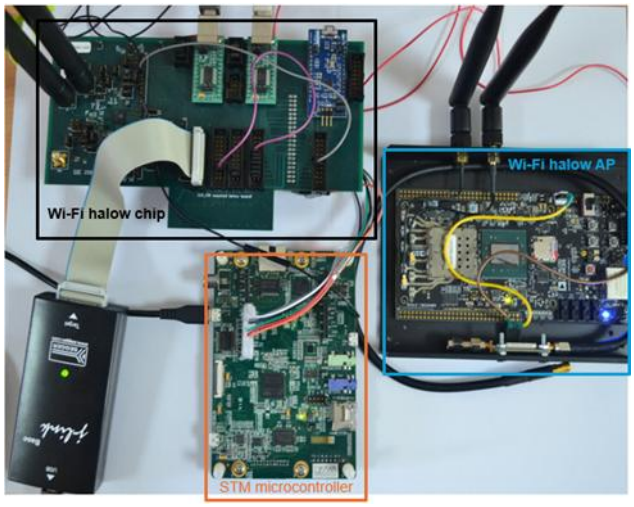
Након успешно извршене иницијализације система, конфигурације прекидне логике и покретања *scheduler*-а, извршена је провера рада *FreeRTOS* окружења кроз креирање и извршавање више *task*-ова са различитим приоритетима и функционалностима. Главна функција апликације иницијализује основни апликативни *task* и покреће *task* за комуникацију, након чега се позива *vTaskStartScheduler()*, чиме се управљање системом предаје *RTOS*-у.

4. *RPC* ДИЗАЈН

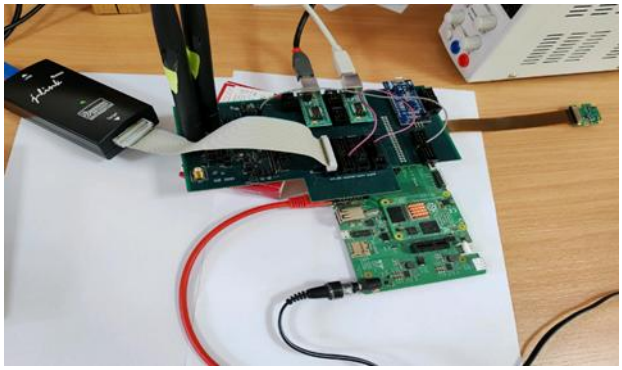
Дијаграм система који видимо на слици 1 је *Wi-Fi HaLow* са слике 3. Он преставља *Wi-Fi HaLow* станицу. Затим је направљена подршка да се станица преко *SPI* протокола може користити преко *STM* микроконтролера или преко *RPi Compute* модула. Слика 3 приказује чип повезан са *STM* микроконтролером. *Wi-Fi HaLow* приступна тачка је постављена поред њих. Слика 4 приказује чип повезан са *RPi Compute Module 4 IO* плочом.

Систем користи посвећен меморијски регион који служи као дељена меморија између *host* процесора и локалног микроконтролера. Кључни аспект система је хардверска транслација адреса. Екстерни *host* увек пише на адресе почевши од 0x00000000 из своје перспективе, али *SPI slave* контролер аутоматски транслира те адресе у стварне физичке адресе локалне меморије. Ово омогућава *host* процесору да види

дељену меморију као континуиран блок почевши од нуле, док се заправо приступа било ком меморијском региону микроконтролера.

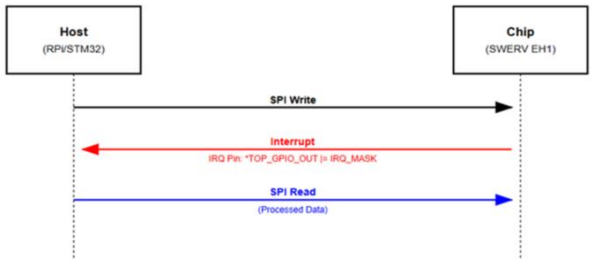


Слика 3. *Конфигурација са STM микроконтролером*



Слика 4. *Конфигурација са Raspberry Pi Compute Module*

На слици 5 приказан је поједностављен *RPC* механизам система.



Слика 5. *Поједностављен дијаграм RPC механизма*

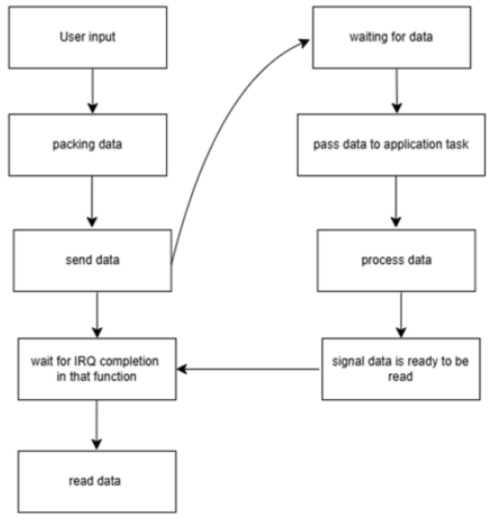
Комуникациони протокол користи флексибилан механизам паковања података. На слици 6 се види формат пакета.

Слика 6. *Формат пакета*

При упису података од стране *host* процесора аутоматски се генерише прекид који обавештава локални систем о пристиглим подацима.

По пријему прекида, систем врши анализу садржаја *RX buffer*-а, идентификује врсту захтева и позива одговарајући механизам обраде. Након обраде, резултати се уписују у *TX buffer* или *async TX buffer*. Затим се у *interrupt_status* структури бележи да је резултат доступан за читање и сетује се бит *GPIO* регистра да сигнализира да је податак спреман за читање.

Да би комуникација између екстерног *host*-а и *RISC-V* чипа била поуздана и ефикасна, неопходан је механизам синхронизације који обе стране информише када су подаци спремни за пријем или обраду. Овај систем користи комбинацију хардверског *SPI* прекида, *FreeRTOS* нотификација, *GPIO* сигнала и семафора на страни *host*-а.



Слика 7. *Дијаграм тока података*

RPC систем на чипу реализован је кроз два *FreeRTOS task*-а:

- *RPC task*, који је одговоран за пријем података преко *SPI* интерфејса, парсирање улазне поруке и прослеђивање података ка апликационом слоју
- *APP task*, који је посвећен извршавању логике на основу примљене команде

Овакво раздвајање одговорности осигурава да је *RPC task* увек спреман да реагује на нови прекид. Након што прими податке и упакује их у структуру *data_packet_t*, он их прослеђује *APP task-y* преко *message queue*-а. Обрада команде се врши независно, омогућавајући паралелну комуникацију и извршавање.

На страни екстерног *host*-а такође постоје 2 *task*-а. *CLI* интерфејс омогућава корисницима да иницирају *RPC* позиве преко командне линије. Ове функције шаљу податке ка чипу и затим чекају резултат. У исто време други *task* чека одговор од стране чипа. Функције се након слања блокирају семафором док не стигне одговор након чега се исти прочита. Свака функција има свој семафор.

5. РАЗВОЈ НА *HOST* СТРАНИ

Први део развоја система обухватао је креирање *SPI Linux* драјвера. Ово је омогућило директну и ефикасну контролу над *SPI* комуникацијом. *RPC* систем је први пут имплементиран на *Raspberry Pi Compute Module 4 IO Board-y*, због подршке за *spidev* интерфејс у *Linux* окружењу, што је значајно поједноставило имплементацију прототипа.

Због *AXI* протокола, адресе на које се шаљу или читају подаци морају бити умножак 4. А због тога како је имплементиран *SPI*, број података који се шаље мора бити дељив са 4, због тога се додаје *padding*. У *cmd_parser* модулу може се видети да је потребно потпуно 32-битно писање за све трансакције, као и да се строго захтева поравнање адреса на 4 бајта [3,4].

Као сигнал за завршетак трансцакције имплеметиран је *interrupt* механизам на *Raspberry Pi* страни, где сетовање бита *GPIO* регистра означава крај обраде података.

6. УНАКРСНА ПЛАТФОРМА И ПРОШИРЕЊА

Иако обе платформе користе исти *SPI* протокол, имплементације на *STM32* и на *Linux*-у се суштински разликују у погледу начина организације и руковања *SPI* трансакцијама.

У имплементацији *Linux SPI* драјвера користе се кернел сервиси. Синхронизација између позива функција и одговора хардвера се управља кроз *completion mechanism*. Док се за *STM32* платформу користе *RTOS* кернел сервиси. Конкретно синхронизациони механизми које он нуди.

У оквиру развоја *RPC* система, успостављена је *Host* библиотека реализована као *Git* submodule, која омогућава апстраховање заједничких *RPC* функционалности и њихово лако дељење између различитих пројеката и платформи. Овај механизам омогућава да се у више независних пројеката користи исти интерфејс, док се платформи специфичне имплементације могу прилагодити или заменити без утицаја на остатак система.

7. ИСКОРИШЋЕЊЕ РЕСУРСА

За анализу искоришћења меморије примењени су алати *riscv64-unknown-elf-nm* и *elf-size-analyze*, који омогућавају идентификацију највећих потрошача меморије и прецизно профилисање *ELF* фајла.

Утврђено је да секција *rodata* заузима значајан део *DCCM*-а, што је узроковало прекорачење расположиве меморије током компилације.

Проблем је решен релокацијом *rodata* секције у *ICCM*, што је подржано архитектуром *SweRV EH1*, и смањењем величине *stack*-а у складу са реалним потребама.

Овим оптимизацијама постигнуто је ефикасније коришћење ресурса и очувана је стабилност система.

8. ЗАКЉУЧАК

У овом раду успешно је реализован *RPC* систем који омогућава стабилну комуникацију између уграђеног *Wi-Fi HaLow* уређаја и спољашњег *host*-а, уз поуздано функционисање у реалном времену. Систем је прилагођен ограничењима уграђених окружења и подржава проширивост на више хардверских платформи.

Иако безбедност није била приоритет у овој фази, потенцијал за даље унапређење постоји увођењем механизама за аутентикацију и енкрипцију података. Уколико будуће апликације буду захтевале поуздану заштиту комуникације, систем се лако може проширити тим функционалностима. *RPC* може бити проширен механизмима за аутентикацију и шифровање, чиме може испунити критеријуме за *SESIP Level 3*, што се често захтева у индустријским апликацијама [5].

9. ЛИТЕРАТУРА

[1] “Distributed Systems : Concepts and Design ”, *by George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair.*

[2] https://github.com/riscv-collab/riscv-gnu-toolchain (приступљено у априлу 2025.)

[3] https://github.com/pulp-platform/axi_spi_slave/blob/master/spi_slave_regs.sv (приступљено у априлу 2025.)

[4] https://github.com/pulp-platform/axi_spi_slave/blob/master/spi_slave_cmd_parser.sv(приступљено у априлу 2025.)

[5] https://globalplatform.org/sesip/ (приступљено у јулу 2025.)

Кратка биографија:

Марија Нешић рођена је у Кикинди 1999. год. Дипломски рад на Факултету техничких наука из области Електротехнике и рачунарства – Ембедед системи и алгоритми одбранила је 2022. године.
контакт: nesicmarija123@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

УДК: 4.3
ДОИ: https://doi.org/10.24867/33BE37Miskovic

УНАПРЕЂЕЊЕ *MODBUS* СИМУЛАТОРА ЗА РЕАЛИСТИЧНУ СИМУЛАЦИЈУ УРЕЂАЈА У СИСТЕМИМА ЗА УПРАВЉАЊЕ ЕНЕРГИЈОМ (*EMS*)

ENHANCEMENT OF A MODBUS SIMULATOR FOR REALISTIC SIMULATION OF DEVICES IN ENERGY MANAGEMENT SYSTEMS (EMS)

Андреа Мишковић, *Факултет техничких наука, Нови Сад*

Студијски програм – РАЧУНАРСКА ТЕХНИКА И РАЧУНАРСКЕ КОМУНИКАЦИЈЕ

Кратак садржај – *Тема овог рада јесте унапређење Modbus симулатора, развијеног у циљу тестирања система за управљање енергијом. Унапређење се огледа у могућности симулације рада уређаја који учествују у системима управљања енергијом, користећи статичке и динамичке симулације. Развијено решење омогућава тестирање функционалности и перформанси система без потребе за физичким уређајима.*

Кључне речи: *Симулатор, Modbus, системи управљања енергијом, симулације*

Abstract – *The topic of this thesis is the improvement of a Modbus simulator, developed for testing energy management systems. The improvement is reflected in the ability to simulate the operation of devices involved in EMS systems, using both static and dynamic simulations. The developed solution enables functionality and performance testing without the need for physical devices.*

Keywords: *Simulator, Modbus, energy management systems (EMS), simulations*

1. УВОД

Концепт симулатора представља кључан алат у развоју и тестирању различитих система. Главни разлог јесте јер омогућава прецизно моделовање и анализу понашања уређаја у контролисаним условима и намерно изазваним околностима. У савременим системима управљања енергијом, рад са уређајима чије је понашање предвидиво представља изазов, имајући у виду ограничену доступност или високу цену [2]. Као одговор на овај изазов, развијен је *Modbus* симулатор који опоната реалистична понашања уређаја у систему управљања енергијом. Симулатор омогућава истовремено тестирање великог броја уређаја и пружа могућност испитивања различитих сценарија, без потребе за физичким уређајима. У овом раду описан је развој и унапређење наведеног симулатора, чијом имплементацијом је омогућено детаљније праћење и валидација понашања мерних уређаја у системима управљања енергијом.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је била др Јелена Ковачевић, ред. проф.

2. ТЕОРИЈСКЕ ОСНОВЕ

На самом почетку, потребно је упознати се са основним теоријским принципима, у циљу бољег разумевања рада.

2.1. Систем управљања енергијом (*EMS*)

Energy Management System (*EMS*) је систем дизајниран за праћење, контролу и оптимизацију производње и потрошње енергије у различитим објектима, мрежама или индустријским постројењима [1]. *EMS* интегрише хардверске и софтверске компоненте како би омогућио ефикасно управљање енергијом и смањење трошкова, док истовремено повећава поузданост и сигурност енергетских система.

Главне функције *EMS* система укључују:

1. Праћење и прикупљање података
2. Анализа и визуелизација
3. Контрола и аутоматизација
4. Оптимизација и предиктивно управљање

EMS функционише тако што прикупља податке са различитих уређаја (нпр. сензори, паметна бројила), обрађује их како би идентификовао обрасце потрошње и неефикасности, визуализује информације кроз графиконе и дашборде, омогућава аутоматско или полуаутоматско управљање уређајима ради смањења непотребне потрошње и користи алгоритме за предиктивно управљање како би оптимизовао рад система и балансирао коришћење различитих извора енергије.

Примена *EMS*-а се јавља у индустрији, комерцијалним зградама, паметним кућама и електродистрибуцији. Предности се огледају у смањењу трошкова енергије, повећању енергетске ефикасности, смањењу емисије штетних гасова и повећању поузданости система, као и могућности интеграције обновљивих извора енергије са постојећим системом [1,3].

2.2. Уређаји у *EMS*

EMS интегрише различите уређаје који омогућавају праћење, контролу и оптимизацију енергије [5, 6]:

1. Уређаји за мерење и прикупљање података
2. Комуникациони уређаји
3. Уређаји за складиштење података
4. Управљачки уређаји
5. Кориснички интерфејси
6. Уређаји за заштиту и безбедност

Уређаји за мерење и прикупљање података бележе потрошњу и производњу енергије, као и кључне електричне параметре попут напона, струје и фреквенције, пружајући основу за детаљну анализу система [6]. Комуникациони уређаји омогућавају поуздану размену информација између сензора, актуатора и софтверског дела *EMS*-а, користећи стандардизоване протоколе и мрежне технологије. Уређаји за складиштење података чувају историјске информације, што омогућава анализу трендова, праћење перформанси и доношење одлука заснованих на дугорочним обрасцима. Управљачки уређаји омогућавају аутоматско и полуавтоматско управљање потрошачима енергије, ради оптимизације потрошње и повећања ефикасности. Кориснички интерфејси пружају оператерима преглед стања система кроз визуализације, олакшавајући праћење и контролу у реалном времену. На крају, уређаји за заштиту и безбедност штите *EMS* и осигуравају заштиту података и стабилност инфраструктуре, штите комуникацију од сајбер претњи и омогућавају шифровану комуникацију.

2.3. *Modbus* протокол

Modbus је комуникациони протокол који се широко користи у индустрији за повезивање електронских уређаја и сензора, посебно у системима за аутоматизацију и управљање енергијом. Протокол омогућава различитим уређајима као што су контролери, сензори и мерни уређаји, да размењују податке на једноставан и стандардан начин.

Modbus функционише по принципу клијент-сервер (раније познатом као *master-slave*) архитектуре. Клијент је обично уређај или софтверска апликација која иницира комуникацију, док је сервер уређај који прима захтеве и враћа одговоре, и то су уређаји као што су паметно бројило, сензор и слични [4].

Постоји више варијанти *Modbus* протокола, а у оквиру овог рада коришћене су:

1. *Modbus RTU* – користи серијску комуникацију и погодан је за индустријска окружења због једноставности, поузданости и ниских трошкова. Идеалан је за повезивање великог броја уређаја.
2. *Modbus TCP/IP* – ради преко мрежа заснованих на *Ethernet* протоколу, омогућава бржу и лакшу интеграцију у савремене мрежне системе.

Modbus протокол је једноставан, поуздан и лак за имплементацију, што је довело до тога да данас буде један од најчешће коришћених протокола за повезивање индустријских уређаја. Омогућава *EMS* системима да прецизно и стабилно прикупљају податке о потрошњи, производњи и другим параметрима, што је кључно за праћење и оптимизацију система.

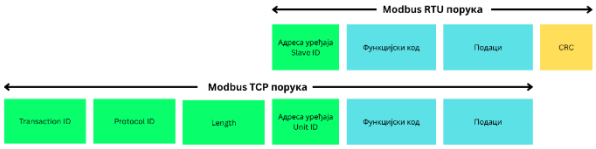
2.3.1. *Modbus* поруке

Свака *Modbus* порука има унапред дефинисану структуру. Пример структуре у *RTU* формату:

- Адреса уређаја
- Функцијски код
- Подаци
- Контрола интегритета података (*CRC* или *LRC*)

У *Modbus TCP* формату, структура поруке укључује *MBAP* (eng. *Modbus Application Protocol*) заглавље:

- *Transaction ID*
- *Protocol ID*
- *Length*
- *Unit ID*

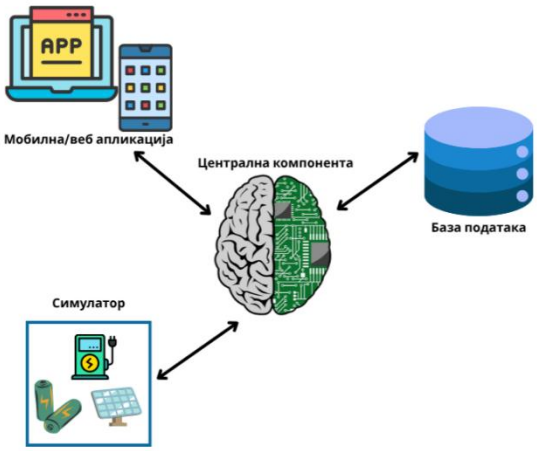


Слика 1 - *Структура Modbus порука*

3. ОПИС РЕШЕЊА

У оквиру овог рада фокус је био на унапређењу симулатора, имплементираног да буде главни алат приликом тестирања система за управљање енергијом. Главни циљ унапређења јесте могућност тестирања понашања читавог система у реалним условима без потребе за физичким уређајима.

3.1. Архитектура система



Слика 2. *Архитектура система*

На слици је приказана архитектура система који се састоји из четири основне компоненте:

1. **Централна компонента** – представља кључни елемент архитектуре система и делује као посредник између свих осталих компоненти. Њена улога је да у реалном времену прима и обрађује телеметријске податке, врши њихову валидацију и упис, као и да прослеђује одређене захтеве.
2. **База података**
3. **Мобилна/веб апликација** – представља директну везу корисника са системом. Служи као визуелни и контролни слој који омогућава надзор, анализу и управљање уређајима.

Корисници имају јасно представљен преглед потрошње и производње у реалном времену, као и историјске податке у виду графикона и табела.

4. **Симулатор уређаја** – опонаша рад реалних уређаја (паметна бројила, *PV* инвертери, батеријски системи, *EV* пуњачи и други) и тако омогућава реалистично тестирање система без потребе за физичким уређајима [7, 8]. Имплементиран је у програмском језику *Python*, коришћењем *Pymodbus* библиотеке.

3.2. *Pymodbus* библиотека

Pymodbus је *open-source Python* библиотека која представља имплементацију *Modbus* протокола [13]. Потпуно је написана у *Python*-у и омогућава једноставно креирање *Modbus* клијента и сервера, што је чини погодном за развој симулатора и тестирање индустријских система.

У оквиру овог рада коришћена је за:

1. Имплементацију *Modbus* сервера у симулатору који емитује податке који потичу од стварних уређаја
2. Дефинисање регистара у којима се чувају симулиране вредности
3. Обраду захтева клијента

3.3. Симулације

Симулације представљају контролисано, намерно и поновљиво извођење сценарија рада уређаја у циљу тестирања, мерења, оптимизације и верификације њиховог понашања. У оквиру овог рада, проширење симулатора је извршено додавањем симулација понашања стварних уређаја током 24 сата. Симулације могу бити статичке или динамичке, у зависности од начина генерисања мерних података.

3.3.1. Статичке симулације

Статичке симулације се заснивају на низу унапред дефинисаних података, који су прикупљени са стварног уређаја током 24 сата. Мерне вредности су снимане сваке секунде како би се обезбедила висока временска резолуција података. Приликом симулације, снимљени подаци се шаљу у истом редоследу централној компоненти система, чиме се обезбеђује верна репродукција рада уређаја у реалним условима.

Предност овог приступа јесте веродостојност генерисаних вредности, док је ограничење у томе што сценарио остаје непромењив и не може подржати нове или неочекиване услове рада. Због тога се статичке симулације често користе за тестирање рада уређаја у идеалним условима, што је корисно за верификацију основних функционалности система.

3.3.2. Динамичке симулације

Динамичке симулације представљају напреднији приступ у тестирању и верификацији рада уређаја и *EMS* система, јер омогућавају генерисање података у складу са дефинисаним сценаријима и условима рада, а не само репродукцију већ постојећих података.

Основна идеја динамичке симулације је да се користи конфигурациона датотека у *JSON* формату, у којој су прецизно дефинисани временски интервали и одговарајуће вредности активне снаге (P) за сваки интервал током једног дана. Овај *JSON* фајл служи као улаз за симулатор, који затим, на основу вредности активне снаге у датом интервалу, применом одговарајућих физичких формула израчунава остале релевантне величине као што су: напон (*U*), струја (*I*), реактивна снага (*Q*), привидна снага (*S*), фреквенција (*f*), укупно утрошена или произведена енергија (*E*).

Пример формуле за израчунавање напона:

$$U_{t+1} = \begin{cases} U_t + \Delta^+, ako\ je\ d_t = 1 \\ U_t - \Delta^-, ako\ je\ d_t = -1 \end{cases}$$
$$d_t = \begin{cases} -1, ako\ U_{t+1} \geq 240V \\ 1, ako\ U_{t+1} \leq 228V \end{cases} \quad (1)$$

Где је:

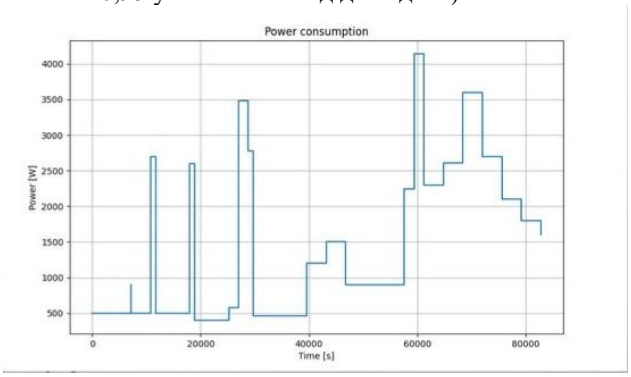
- U_t – вредност напона у тренутном кораку t
- d_t – правац промене
- Δ^+ – корак повећања који износи 0,2V
- Δ^- – корак смањења који износи 0,15V

Пример формуле за израчунавање струје:

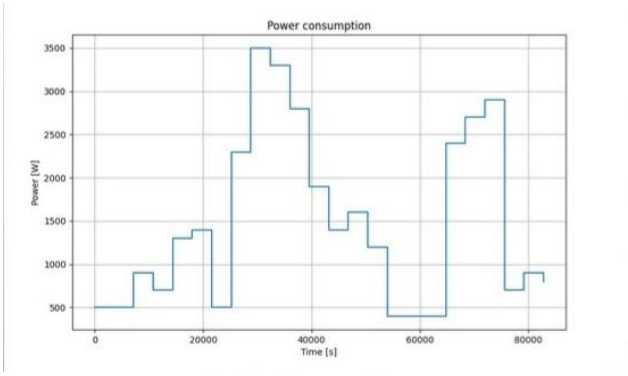
$$I_t = \frac{P_t}{U_t \cdot \cos\varphi} \quad (2)$$

Где је:

- P_t – активна снага у тренутку t
- U_t – напон у тренутку t
- $\cos\varphi$ – фактор снаге (варира између 0,92 и 0,95 у зависности од доба дана)



Слика 3. *Пример симулације током 24 сата*



Слика 4. *Пример симулације током 24 сата*

4. ТЕСТИРАЊЕ

Тестирање је спроведено са циљем да се провери тачност, поузданост и скалабилност развијеног симулатора. За анализу су коришћени унапред дефинисани сценарији који обухватају нормалан рад, рад при повећаном оптерећењу, као и услове у којима долази до поремећаја у комуникацији. Тестирање је подељено у две групе – функционално и тестирање перформанси.

Табела 1 - *Функционални тестови*

Тест случај	Очекивани резултат	Статус
Генерисање динамичких података	Подаци прате дефинисани JSON сценарио	✓
Генерисање статичких података	Подаци одговарају снимљеним вредностима	✓
Читање података из регистра	Исправно очитане вредности	✓
Писање вредности у регистар	Исправно уписана вредност	✓
Симулација прекида рада уређаја	Систем пријављује грешку и не емитује податке	✓
Брзо мењање вредности регистара	Систем успешно бележи све промене без губитака података	✓

Табела 2. *Тестови перформанси*

Број уређаја	Просечно време одзива [ms]	Максимална латенција [ms]	CPU оптерећење [%]
10	15	40	12
50	28	75	27
100	45	120	46
200	92	210	78

Спроведени тестови потврђују да развијени симулатор испуњава циљеве постављене у раду. Он омогућава верну репродукцију рада уређаја, стабилан пренос података, скалабилност већег броја уређаја и отпорност на грешке.

5. ЗАКЉУЧАК

У овом раду представљен је развој и унапређење *Modbus* симулатора са могућношћу реалистичног опонашања рада уређаја у оквиру система за управљање енергијом (*EMS*). Развијени симулатор успешно омогућава тестирање и валидацију функционалности без употребе физичких уређаја, чиме се значајно смањују трошкови, убрзава развој и повећава безбедност тестирања.

Имплементација је заснована на програмском језику *Python* и *Pymodbus* библиотеци, што је омогућило флексибилну конфигурацију понашања симулираних уређаја и подршку за велики број паралелних инстанци. Унапређење се огледа у два могућа типа рада симулатора – статичке и динамичке симулације. Статичке симулације омогућавају репродукцију унапред дефинисаних скупова података, прикупљених са реалних уређаја, погодних за тестирање у идеалним условима. Динамичке симулације нуде већу флексибилност, јер омогућавају конфигурисање сценарија у *JSON* формату и динамичко израчунавање релевантних електроенергетских параметара према физичким формулама, што је посебно корисно за тестирање система у променљивим или екстремним условима.

Резултати имплементације показују да овако развијен симулатор може верно да опонаша рад великог броја уређаја, генерише реалистичне податке и омогући тестирање кључних *EMS* функционалности.

4. ЛИТЕРАТУРА

[1] A. R. Al-Ali, I. A. Zualkernan, M. Rashid, R. Gupta, and M. AliKarar, “A Smart Home Energy Management System Using IoT and Big Data Analytics Approach,” IEEE Transactions on Consumer Electronics, vol. 63, no. 4, pp. 426–434, Nov. 2017.

[2] Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson F., Moestedt A., Werner, B. Simics: A full system simulation platform. Computer, 2002: 35(2), 50–58.

[3] V. K. Barai, S. R. Mohanty, and N. Kishor, “A review on modeling and simulation of energy management systems,” International Journal of Energy Research, vol. 39, no. 10, pp. 1311–1323, Aug. 2015.

[4] "Modbus Application Protocol Specification V1.1b3," Modbus Organization, 2012. [Online]. Available: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[5] F. C. Schweppe, Energy Management Systems for Electric Utilities. Springer, 2013.

[6] A. S. Bouhouras, P. M. Georgilakis, and D. P. Labridis, “A comprehensive review of energy management systems for microgrids,” Electric Power Systems Research, vol. 122, pp. 159–168, May 2015.

[7] Park, S., Kim, H., Moon, H., Heo, J., and Yoon, S. Concurrent simulation platform for energy-aware smart metering systems. IEEE Transactions on Consumer Electronics, 2010: 56(3), 1918–1926.

[8] Cen, Z. Modeling and Simulation for an 8 kW Three-Phase Grid-Connected Photo-Voltaic Power System. Open Physics, 2017: 15(1), 603–612

[9] Jovanović, B., Filipović, J., & Bakić, V. (2017). Energy management system implementation in Serbian manufacturing – Plan–Do–Check–Act cycle approach. Journal of Cleaner Production, 162, 1144–1156.

[10] Alahakoon, D., & Yu, X. (2016). Smart electricity meter data intelligence for future energy systems: A survey. IEEE Transactions on Industrial Informatics, 12(1), 425–436.

[11] S. K. Rathor and D. Saxena, “Energy management system for smart grid: An overview and key issues,” Int. J. Energy Res., vol. 44, no. 6, pp. 4067–4109, 2020.
[12] S. K. Khaitan and J. D. McCalley, “Design Techniques and Applications of Cyber-Physical Systems: A Survey,” IEEE Systems Journal, vol. 9, no. 2, pp. 350–365, Jun. 2015.
[13] GitHub – *Pymodbus Documentation*. Available: https://github.com/pymodbus-dev/pymodbus

Кратка биографија:

Андреа Мишковић рођена је у Шапцу 1999. године. Основне студије завршила је на Факултету техничких наука 2022. године. Мастер рад на тему „Унапређење *Modbus* симулатора за реалистичну симулацију уређаја у системима за управљање енергијом“ одбранила је 2025. године.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.41
DOI: https://doi.org/10.24867/33BE38Rajnovic

MIGRACIJA MIKRO KLIJENTSKIH APLIKACIJA IZ VEB U DESKTOP OKRUŽENJE

MIGRATION OF MICRO-FRONTEND APPLICATIONS FROM WEB TO DESKTOP ENVIRONMENT

Teodora Rajnović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – SOFTVERSKO INŽENJERSTVO I INFORMACIONE TEHNOLOGIJE

Kratak sadržaj – *U radu je predstavljena arhitektura mikro klijentskih aplikacija (eng. micro-frontend) i prednosti koje ovaj pristup donosi u razvoju veb aplikacija. Opisani su savremeni pristupi u razvoju desktop aplikacija, pri čemu je Electron identifikovan kao najprikladnije rešenje za migraciju postojeće mikro klijentske aplikacije u desktop okruženje. Prikazana je softverska arhitektura konkretne veb aplikacije i proces njene transformacije u desktop aplikaciju, uz očuvanje modularnosti, internet konekcije i postojećeg koda.*

Ključne reči: *Mikro klijentska arhitektura, Desktop aplikacije, Electron radni okvir*

Abstract – *The paper presents the architecture of micro-frontend applications and the advantages that this approach brings in the development of web applications. Modern approaches in desktop application development are described, with Electron identified as the most suitable solution for migrating an existing micro-frontend application to a desktop environment. The software architecture of a specific web application and the process of its transformation into a desktop application are presented, while preserving modularity, internet connection and existing code.*

Keywords: *Micro-frontend architecture, Desktop applications, Electron framework*

1. UVOD

Micro-frontend arhitektura postaje sve popularniji pristup u razvoju modernih i savremenih veb aplikacija. Ovaj pristup omogućava podelu velikih monolitnih klijentskih aplikacija na manje, nezavisne module, što doprinosi lakšem održavanju, unapređenju i timskom radu. Micro-frontend koncept koristi principe mikro-servisne arhitekture, ali se primenjuje na frontend deo aplikacije [1]. Iako su veb aplikacije danas dominantne zbog svoje dostupnosti preko veb pregledača, korisnici informacionih sistema često preferiraju desktop aplikacije zbog boljih performansi, integracije sa operativnim sistemom i veće kontrole nad bezbednosti i privatnosti podataka.

U slučajevima kada su veb aplikacije realizovane kroz upotrebu micro-frontend tehnologije, mogu se uočiti značajni benefiti u kontekstu velikih sistema koji obrađuju velike količine informacija i vrše komunikaciju između različitih komponenti. Ipak, zbog specifičnih zahteva korisnika, kao što su brzina, rad van mreže i direktan pristup hardveru, javlja se potreba za migracijom u desktop okruženje.

Migracija micro-frontend veb aplikacije prikazana je na primeru DevAdmin aplikacije, čija je namena konfiguracija, dijagnostika i analiza hardverskih uređaja. DevAdmin je razvijena kao Angular aplikacija sa modularnom arhitekturom, koja omogućava efikasno upravljanje komponentama i funkcionalnostima. Korisnicima je omogućeno da podešavaju mrežne i bezbednosne parametre, upravljaju licencama i pristupaju statusnim izveštajima radi rešavanja tehničkih problema.

Ovaj rad istražuje tehnološke aspekte te migracije, uz očuvanje modularne arhitekture i postojećeg koda, kao i prednosti koje se postižu u pogledu performansi, bezbednosti i korisničkog iskustva.

2. TEHNOLOŠKE OSNOVE

Primer migracije micro-frontend aplikacije u desktop aplikaciju predstavljen je na primeru Angular [2] aplikacije, uz primenu micro-frontend arhitekture zasnovane na Module Federation konceptu. U svrhu odabira najpogodnijeg rešenja za migraciju iz micro-frontend u desktop okruženje, dat je sažet prikaz savremenih pristupa u razvoju desktop aplikacija.

2.1. Angular

Angular [2] je TypeScript [3] radni okvir koji omogućava izgradnju dinamičkih, single-page aplikacija kroz upotrebu komponenti, šablona i reaktivnog programiranja. Komponente definišu izgled i ponašanje aplikacije, dok servisi omogućavaju deljenje logike i podataka. Pruža podršku za reaktivno programiranje putem signala, odloženo učitavanje komponenti radi poboljšanja performansi, kao i sistem za rutiranja za navigaciju bez ponovnog učitavanja stranice. Angular CLI (Command-line interface) olakšava razvoj aplikacija kroz brzo generisanje elemenata, a njegova arhitektura obezbeđuje strukturiran i održiv razvoj modernih aplikacija.

2.2. JavaScript

JavaScript je interpretirani programski jezik visokog nivoa, uveden 1995. godine, koji omogućava manipulaciju korisničkog interfejsa u veb pregledačima [4]. JavaScript podržava više programskih paradigmi, uključujući i objektno i funkcionalno programiranje, i poseduje ugrađene API (Application Programing Interface) interfejse, za rad sa tekstom, datumima i DOM (Document Object Model) objektima. Iako direktno ne podržava ulazno/izlazne operacije, okruženja poput Node.js proširuju njegovu primenu van pregledača. ECMAScript

standard je nastao kao posledica potrebe za proširenjem, unapređenjem konzistentnosti i standardizacijom JavaScript programskog jezika u različitim okruženjima.

2.3. Micro-frontend aplikacije

Micro-frontend arhitektura predstavlja savremeni pristup razvoju veb aplikacija, koji omogućava podelu klijentske aplikacije na manje, nezavisne i lako upravljive module [5]. Za razliku od tradicionalne arhitekture klijentskih aplikacija, gde je aplikacija monolitnog karaktera i zahteva visok njivo koordinacije među članovima tima, micro-frontend omogućava timovima da samostalno razvijaju, testiraju i primenjuju funkcionalnosti, što značajno unapređuje skalabilnost i brzinu isporuke novih funkcionalnosti. Ovaj pristup je inspirisan mikro-servisima na serverskoj strani i omogućava veću fleksibilnost, nezavisnu primenu i lakše održavanje aplikacija. Velike kompanije poput Spotify, Ikea i Zalando već uspešno primenjuju micro-frontend arhitekturu za razvoj kompleksnih sistema [6].

Za implementaciju micro-frontend aplikacija koriste se alati poput Webpack, Module Federation, Single-SPA i Angular Elements. Module Federation omogućava učitavanje nezavisnih modula, kao što bi na primeru veb aplikacija za kupovinu to bile korisničke korpe za kupovinu ili sistem za plaćanje. Ovakvom arhitekturom izbegava se dupliranje koda i omogućava se deljenje UI (User Interface) komponenti između više aplikacija. Ukoliko se ispoštuju konvencije imenovanja i izoluje kod radi izbegavanja konflikata, ova arhitektura podstiče timsku autonomiju, jer svaki tim može da radi na svom modulu bez potrebe za koordinacijom sa drugim timovima.

Micro-frontend arhitektura se može organizovati kroz dve podele: horizontalna i vertikalna. Horizontalna podela podrazumeva više modula na istoj stranici, gde se timovi fokusiraju na tehničke aspekte poput UI komponenti ili API integracije. Ovaj model je pogodan za velike timove i poslovne poddomene, ali zahteva veću koordinaciju. Vertikalna podela organizuje aplikaciju oko poslovnih funkcionalnosti, gde svaki micro-frontend predstavlja jednu celinu koju razvija jedan tim. Shell aplikacija upravlja učitavanjem ovih modula i obezbeđuje da se, u datom trenutku, prikazuje samo jedan micro-frontend, što je posebno korisno kod SPA aplikacija.

Kompozicija micro-frontend aplikacija omogućava integraciju nezavisnih modula u jedan korisnički interfejs, uz podršku radnih okvira kao što su Angular, React i Vue.js. Web Components omogućavaju kreiranje inkapsuliranih HTML (Hyper Text Markup Language) elemenata, čime se postiže modularnost i lakša saradnja među timovima.

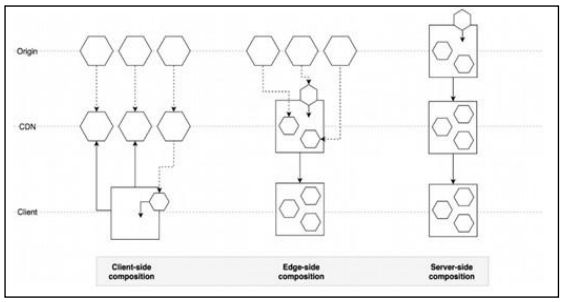
Postoje tri strategije kompozicije micro-frontend aplikacija:

• kompozicija na strani servera – server sastavlja micro-frontend module pre nego što pošalje konačni HTML klijentu, čime se ubrzava učitavanje i smanjuje opterećenje na pregledaču. Idealna je za visoko indeksirane ili kritične veb stranice, ali zahteva tesnu integraciju sa serverom, što ograničava nezavisnost modula.

• kompozicija na strani ivice (edge) – slična je serverskoj kompoziciji, ali se izvršava bliže korisniku, konkretno preko CDN (Content Delivery Network) infrastrukture. Omogućava brzu isporuku sadržaja, ali unosi dodatnu složenost zbog različitih CDN implementacija i ograničenog broja alata za podršku.

• kompozicija na strani klijenta – moduli se učitavaju i sklapaju direktno pomoću shell aplikacije. Ovaj pristup omogućava dinamičko upravljanje, veću autonomiju timova i bolje korisničko iskustvo, bez potrebe za ponovnim učitavanjem cele stranice.

Strategije kompozicije micro-frontend arhitekture prikazane su na slici 1.



Slika 1. *Metode kombinacije micro-frontend arhitektura [6]*

Module Federation je tehnologija koja je uvedena u Webpack verziji 5 i koja omogućava deljenje koda i zavisnosti između više aplikacija [7]. Aplikacije se dele na nezavisno razvijene, testirane i primenjene micro-frontend module (remotes), koji mogu biti ili "proizvođači" (remotes) i "potrošači" (hosts) ili imati obije uloge. Proizvođači izlažu svoje module, a potrošači su aplikacije koje koriste te module. Ovaj pristup smanjuje dupliranje koda, poboljšava performanse i omogućava skalabilan razvoj nezavisnih micro-frontend modula.

2.4. Savremeni pristup u razvoju desktop aplikacija

Desktop aplikacije se izvršavaju lokalno na korisnikovom uređaju, bez potrebe za internet konekcijom, što im omogućava bolje performanse, veću kontrolu nad resursima i veću bezbednost. Iako imaju prednosti u stabilnosti i brzini, zahtevaju instalaciju, redovno ažuriranje i često su vezane za određeni operativni sistem [8].

Razvoj desktop aplikacija uključuje različite tehnologije, zavisno od potreba samog projekta. Pri odabiru tehnologije za realizaciju migracije micro-frontend aplikacije u desktop aplikaciju razmatrani su aspekti poput stepena izmene koda, podrške za složenim funkcionalnostima i multiplatformske podrške. U svrhu migracije i prema datim kriterijumima razmatrane su sledeće tehnologije:

• WPF (.NET) – nudi bogat skup UI elemenata, ali je vezan za Windows. Zahteva potpun prelaz na C# i visok stepen refaktorizacije.

• JavaFX – omogućava razvoj u Java programskom jeziku; slaba integracija sa modernim veb stekom. Zahteva potpunu refaktorizaciju.

• PyQT – dobar GUI alat u Python programskom jeziku, ali zahteva napuštanje postojećeg koda i ima ograničenja u sistemskoj integraciji.

• Flutter – namenjen prvenstveno za mobilne aplikacije, koristi programski jezik Dart. Veliki stepen

refaktorizacije i nije pogodan za kompleksne desktop sisteme.

• Tauri – moderan i lagan okvir sa dobrim performansama, ali manje zreo za složene aplikacije i zahteva prelaz na Rust programski jezik.

• PWA (Progressive Web Apps) – nizak stepen refaktorizacije, ali ne zadovoljava uslove za offline rad sa hardverom.

• Electron – srednji stepen refaktorizacije, omogućava ponovnu upotrebu Angular koda. Podržava multiplatformski razvoj, bogate sistemske funkcionalnosti, ali ima veće memorijsko opterećenje.

Za migraciju postojeće veb aplikacije u desktop okruženje, izabran je Electron radni okvir. Electron omogućava ponovnu upotrebu postojećeg Angular koda, podržava više operativnih sistema i nudi stabilnost i funkcionalnost potrebnu za kompleksne aplikacije. Uprkos većem memorijskom opterećenju, njegova zrelost i široka primena čine ga pouzdanim izborom.

2.5. Electron

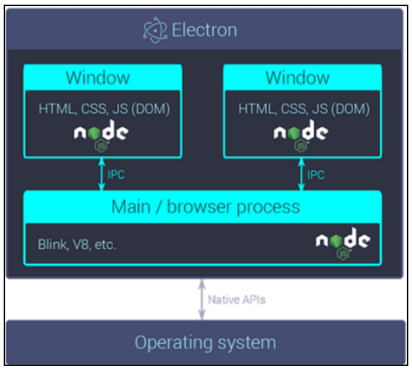
Electron je multiplatformski radni okvir koji omogućava razvoj desktop aplikacija koristeći veb tehnologije kao što su HTML, CSS i JavaScript [9]. Izgrađen je na Chromium i Node.js projektima, što mu omogućava pristup sistemskim resursima i funkcionalnostima kao što su dijalozi, obaveštenja i automatska ažuriranja. Podržava Windows, macOS i Linux, čime obezbeđuje konzistentno korisničko iskustvo na više platformi.

Electron aplikacije su sačinjene od dva procesa:

• glavni proces - upravlja logikom aplikacije, prozorima, sistemskim funkcijama i životnim ciklusom aplikacije.

• proces prikazivanja – upravlja veb sadržajem svakog prozora.

Komunikacija između ovih proces se vrši preko IPC (Inter-Process Communication) mehanizma, što omogućava stabilnost i izolaciju procesa. Glavni proces koristi module kao što su BrowserWindow modul za kreiranje prozora i app modul za upravljanje životnim ciklusom aplikacije. Procesi prikazivanja koriste HTML, CSS i JavaScript za prikaz sadržaja, ali, iz bezbednosnih razloga, nemaju direktan pristupa Node.js API interfejsima. Slika 2 prikazuje arhitekturu Electron aplikacije, kao i odnos između glavnog i procesa prikazivanja.

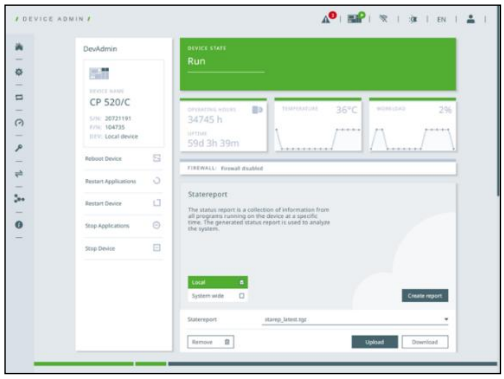


Slika 2. *Arhitektura Electron aplikacije [10]*

Da bi se omogućila bezbedna interakcija između proces, Electron koristi preload skripte koje se izvršavaju pre učitavanja veb sadržaja. Preload skripte imaju pristup Node.js API interfejsima i preko contextBridge modula mogu bezbedno izložiti funkcionalnosti procesu prikazivanja. Pored toga, Electron podržava kreiranje uslužnih procesa preko UtilityProcess API interfejsa, koji se koriste za izvršavanje zahtevnih ili nepouzdanih zadataka. Procesi prikazivanja komuniciraju sa uslužnim procesima preko MessagePort kanala.

3. MIGRACIJA ANGULAR VEB APLIKACIJE U STANDALONE ELECTRON APLIKACIJU

Migracija Angular veb aplikacije u standalone Electron okruženje prikazana je kroz primer DevAdmin aplikacije. DevAdmin je alat za korisničku podršku i analizu hardverskih uređaja. Aplikacija omogućava pregled sistemskih informacija, upravljanje podešavanjima, generisanje izveštaja o statusu i greškama, kao i direktnu interakciju sa uređajima preko lokalne mreže. Zahvaljujući micro-frontend arhitekturi i intuitivnom interfejsu (slika 3.), DevAdmin pruža fleksibilno i efikasno rešenje za tehnički podršku, a njena migracija u Electron aplikaciju omogućava stabilniji i nezavisniji rad u desktop okruženju.

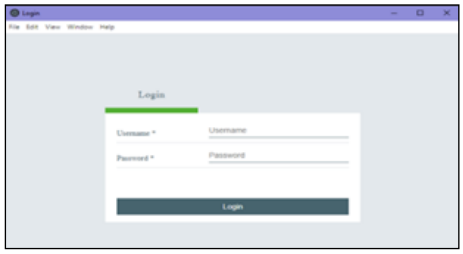


Slika 3. *Pregled interfejsa DevAdmin aplikacije*

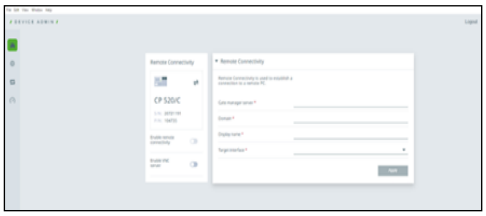
DevAdmin aplikacija predstavlja konkretan primer primene micro-frontend arhitekture uz korišćenje Module Federation koncepta u okviru Angualr radnog okvira. Aplikacija je organizovana kao modularna klijentska veb aplikacije, sa dve shell aplikacije koje dinamički učitavaju više nezavisnih remote modula. Ovaj pristup omogućava timovima da razvijaju funkcionalnosti izolovano, uz bolju skalabilnost i održivost koda. Migracija DevAdmin aplikacije koristeći Electron omogućila bi korišćenje postojećeg veb koda za multiplatformsku distribuciju.

Kompozicija micro-frontend modula u DevAdmin aplikaciji realizovana je na strani klijenta, gde shell aplikacija dinamički integriše remote module tokom izvršavanja. Module Federation omogućava da se ovi moduli učitaju na zahtev, čime se postiže fleksibilna i prilagodljiva arhitektura. Da bi se konfigurisao Module Federation potrebno je definisati webpack.config.js, module-federation.config.json i default-module-federation-config.js fajlovi, u kojima se definišu remote aplikacije i deljeni Angular moduli. Ovaj sistem omogućava optimizaciju učitavanja i izbegavanje konflikta u zavisnostima, čime se unapređuju performanse i stabilnost aplikacije.

Migrirana DevAdmin Electron aplikacija sačinjena je od prozora za prijavu korisnika (slika 4) i glavnog prozora (slika 5). Shell i remote aplikacije se mogu učitati bilo putem udaljene veb adrese ili direktno sa lokalnog diska. Korišćenjem Module Federation koncepta, omogućena je dinamička integracija nezavisnih modula, dok Electron omogućava njihovo prikazivanje u odvojenim karticama ili prozorima.



Slika 4. *Stranica za prijavu korisnika*



Slika 5. *Glavni ekran sa učitanom remote aplikacijom*

Učitavanje aplikacija realizovano je kroz dve metode: preko loadURL metode za učitavanje sa lokalnog servera ili preko loadFile metode za učitavanje kompajliranih HTML fajlova. U drugom slučaju, micro-frontend aplikacije se učitavaju unutar jednog dashboard.html fajla putem <iframe> elementa, čime se simulira ponašanje shell aplikacije. Ovaj pristup omogućava bolju izolovanost komponenti, poboljšava performanse i bezbednost, ali zahteva ručnu izgradnju aplikacije nakon svake izmene.

Da bi se izbegli problemi sa apsolutnim putanjama unutar <iframe> aplikacija, implementirano je preusmeravanje HTTP i WebSocket zahteva na nivou Electron sesije, koristeći webRequests.onBeforeRequest metode. Ovo omogućava da aplikacije zadrže originalnu logiku komunikacije sa serverom, bez izmene koda. Pored toga, upravljanje životnim ciklusom prozora i korisničkom sesijom realizovano je kroz IPC mehanizam i localStorage, čime se obezbeđuje nezavisnost i izolovanost svake remote aplikacije.

Celokupno rešenje omogućava da se postojeća veb arhitektura lako migrira u desktop okruženje, uz minimalne izmene u kodu i zadržavanje osnovnih principa micro-frontend arhitekture. Ovakva kombinacija tehnologija pruža fleksibilnost, stabilnost i mogućnost daljeg proširenja aplikacije u različitim razvojnim i produkcionim scenarijima.

4. ZAKLJUČAK

U ovom radu je uspešno istražena i demonstrirana migracija Angular micro-frontend veb aplikacija u desktop okruženje uz korišćenje Electron radnog okvira. Zbog svoje zrelosti, podrške za naprednim funkcionalnostima i mogućnosti duboke integracije sa operativnim sistemom, Electron je odabran kao najprikladnije rešenje za migraciju, dok su alternative poput Tauri i PWA odbačene zbog nezrelosti tehnologije, podrške za offline rad i slabe integracije sa operativnim sistemom. Migracija je realizovana bez drastičnih izmena u postojećem kodu, uz očuvanje modularnosti i principa koje propisuje micro-frontend arhitektura.

Tehničke strategije uključuju učitavanje remote aplikacija, upravljanje zavisnostima, preusmeravanje HTTP i WebSocket zahteva i komunikacije preko IPC mehanizma. Electron aplikacija preuzima ulogu nove shell aplikacije, omogućujući postepenu i kontrolisanu migraciju. Iako ovaj pristup donosi određene izazove, kao što su povećana potrošnja resursa i bezbednosni rizici, uz primenu preporučenih praksi, moguće je obezbediti stabilno i bezbedno okruženje.

Rezultati pokazuju da je predloženo rešenje efikasno i održivo za prelazak kompleksne veb aplikacije u desktop okruženje, uz zadržavanje fleksibilnosti, skalabilnosti i agilnog razvoja. Ovakva arhitektura omogućava de se micro-frontend aplikacije pokreću nezavisno, čime se zadržavaju sve prednosti originalnog veb pristupa u kontekstu desktop aplikacija.

5. LITERATURA

[1] Prajwal, Y., Parekh, J. V., & Shettar, R. (2021). A brief review of micro-frontends. United International Journal for Research and Technology, 2(8), 18.

[2] https://angular.dev (pristupljeno u septembru 2025.)

[3] https://www.typescriptlang.org (pristupljeno u septembru 2025.)

[4] Haverbeke, M. (2018). Eloquent javascript: A modern introduction to programming. No Starch Press.

[5] https://martinfowler.com/articles/micro-frontends.html (pristupljeno u septembru 2025.)

[6] Peltonen, S., Mezzalira, L., & Taibi, D. (2021). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. Information and Software Technology, 136, 106571.

[7] https://module-federation.io (pristupljeno u novembru 2025.)

[8] https://www.geeksforgeeks.org/what-is-standalone-application (pristupljeno u novembru 2025.)

[9] https://www.electronjs.org (pristupljeno u novembru 2025.)

[10] Alymkulov, D. (2019). Desktop Application Development Using Electron Framework: Native vs. Cross-Platform. [Bachelor's Thesis]. South-Eastern Finland University of Applied Sciences.

Kratka biografija:

Teodora Rajnović rođena je 1. avgusta 1999. godine u Zvorniku. Osnovnu školu „Stefan Mitrov Ljubiša" završila je u Budvi, 2014. godine. U istom gradu je i završila srednju školu „Danilo Kiš", smer turistički tehničar 2018. godine. Kasnije te godine, upisuje Fakultet tehničkih nauka u Novom Sadu, smer Softversko inženjerstvo i informacione tehnologije. Studije završava u roku, 2022. godine.

У реализацији Зборника радова Факултета техничких наука у току 2025. године учествовали су следећи рецензенти:

Ацо Антић
Александар Анђелковић
Александар Ковачевић
Александар Купусинац
Александар Селаков
Александар Станисављевић
Александра Радуловић
Анка Старчев-Ћурчин
Андрија Рашета
Атила Зелић
Богдан Павковић
Бојан Батинић
Бојан Матић
Бојан Тепавчевић
Бојан Јовановић
Борис Агарски
Борис Стојић
Бранко Бркљач
Бранко Милосављевић
Дамир Ђаковић
Данијела Ћирић
Данијела Грачанин
Данијела Лалић
Дарко Чапко
Дарко Стефановић
Дејан Ецет
Дејан Рељић
Дејан Моврин
Дејан Убавин
Дејана Недучин
Драган Адамовић
Драган Дину
Драган Ивановић
Драган Иветић
Драган Јовановић
Драган Пејић
Драган Ружић
Драгана Константиновић
Драгољуб Шевић
Драго Жарковић
Дуња Врбашки
Ђорђе Ћелић
Ђорђе Вукелић
Ђорђије Дупљанин
Горан Јефтенић
Горан Маринковић
Горан Савић
Горан Сладић
Горан Стојановић
Горан Тепић
Гордан Стојић
Гордана Остојић
Гордана Милосављевић
Игор Дејановић
Игор Мараш
Игор Пешко
Илија Башићевић
Исидора Ђурић
Иштван Пап
Ива Шиђанин
Иван Мезеи
Иван Прокић
Ивана Јурич
Ивана Михајловић
Ивана Томић
Ивана Васиљевић
Ивана Катић
Ивана Мараш
Ивана Мишкељин
Јелена Атанацковић Јеличић
Јелена Бороцки
Јелена Иветић
Јелена Радић
Јелена Радонић
Јелена Сливка
Јелена Спајић
Калман Бабковић
Лазар Ковачевић
Лидија Крстановић
Љиљана Поповић
Љубица Дуђак
Љубомир Будински
Магдолна Пал
Маја Турк Секулић
Маја Петровић
Марија Силађи
Маринко Масларић
Марко Марковић
Марко Тодоров
Марко Векић
Маша Букуров
Мијодраг Милошевић
Милан Челиковић
Милан Делић
Милан Гаврић
Милан Маринковић
Милан Мирковић
Милан Рапајић
Милан Рацков
Милан Сегединац
Милан Тривунић
Милан Видаковић
Милена Крклješ
Милица Врачарић
Милица Миличић
Милица Кисић
Милош Симић
Милош Шешлија
Милован Лазаревић
Миља Симеуновић
Миљана Прица
Миљана Зековић
Миодраг Милутинов
Миодраг Жигић
Мирослав Драмићанин
Мирослав Зарић
Мирко Раковић
Миро Говедарица
Мирослав Кљајић
Мирослав Зарић
Младен Томић
Младен Радишић
Наташа Милосављевић
Небојша Бркљач
Небојша Пјевалица
Небојша Радовић
Небојша Ралевић
Неда Милић Керестеш
Немања Кашиковић
Немања Сремчев
Немања Тасић
Ненад Рушкић
Ненад Симеуновић
Никола Лубурић
Никола Војновић
Платон Совиљ
Предраг Теодоровић
Радивоје Динуловић
Радомир Којић
Романа Бошковић-Живановић
Сандра Дедијер
Саша Медић
Савка Адамовић
Слађана Милићевић
Славица Митровић
Слободан Морача
Слободан Шупић
Слободан Табаковић
Срђан Милићевић
Срђан Попов
Срђан Вукмировић
Стеван Гостојић
Стеван Милисављевић
Стеван Станковски
Сузана Драганић
Светлана Бачкалић
Светлана Николичић
Тамара Шкорић
Татјана Ковачевић
Татјана Кочетов-Мишулић
Теодора Вучковић
Весна Стојаковић
Вијолета Врховац
Вишња Жугић
Владимир Ђаковић
Владимир Илић
Владимир Мученски
Војин Илић
Вук Богдановић
Вук Врањковиц
Зоран Брујић
Зоран Чепић
Зоран Јеличић
Жељко Јакшић
Жељко Кановић
Жељко Вуковић
Живко Павловић