УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

ЗБОРНИК РАДОВА
ФАКУЛТЕТА ТЕХНИЧКИХ НАУКА

Едиција: Техничке науке – зборници
Година: XL
Број: 3/2025

Нови Сад

Едиција: „Техничке науке – Зборници“
Година: XL Свеска: 3
Издавач: Факултет техничких наука Нови Сад

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

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 трећа 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.

У овом броју штампани су радови студената мастер студија, сада већ мастера, који су радове бранили у периоду од 17. 9. 2024. до 30. 9.2024. год. То су оригинални прилози студената са главним резултатима њихових мастер радова. 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 3, објављени су радови из области електротехнике и рачунарства.

Уредништво се нада да ће и професори и сарадници ФТН-а и других институција наћи интерес да публикују своје резултате истраживања у облику регуларних радова у овом часопису.

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

Електротехника и рачунарство

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.89
DOI: https://doi.org/10.24867/30BE01Jovic

KOMPARATIVNA ANALIZA TEKSTUALNIH MODELA BERT, BART I XLNET ZA PREDIKCIJU POPULARNOSTI YOUTUBE SNIMKA

COMPARATIV ANALYSIS OF TEXTUAL MODELS BERT, BART AND XLNET FOR PREDICTING THE POPULARITY OF YOUTUBE VIDEOS

Nela Jović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – Elektrotehnika i računarstvo

Kratak sadržaj – *Ovaj rad se bavi komparativnom analizom tri tekstualna modela BERT, BART i XLNet. Podaci za treniranje i validaciju su preuzeti sa sajta kaggle.com. Postoje dva skupa podataka. Prvi inicijalni i drugi proširen, koji je nastao zbog pokušaja poboljšanja recall metrike. Nakon preuzimanja urađeno je pretprocesiranje odnosno tokenizacija i stemovanje. Skup je podeljen u razmeri 80:20 za treniranje i validaciju. Isprobane su različite vrednosti za learning rate i batch size, a najbolji rezultat je dobijen korišćenjem BERT large modela kada je learning rate 1e-5, a batch size 8. Druga dva modela dala su nešto lošiji rezultat od BERT-a. Svi eksperimenti i rezultati biće predstavljeni u nastavku.*

Ključne reči: *BERT, BART, XLNet, Fine-tuning, Youtube*

Abstract – *This paper talks about a comparative analysis of three text models: BERT, BART, and XLNet. The data for training and validation were taken from the website kaggle.com. There are two datasets: the initial one and an expanded one, created in an attempt to improve the recall metric. After downloading, preprocessing was done, including tokenization and stemming. The dataset was split in a ratio of 80:20 for training and validation. Different values for learning rate and batch size were tested, and the best result was achieved using the BERT large model with a learning rate of 1e-5 and a batch size of 8. The other two models produced somewhat worse results than BERT. All experiments and results will be presented below.*

Keywords: : *BERT, BART, XLNet, Fine-tuning, Youtube*

1. UVOD

Svedoci smo koliko se savremenih internet zanimanja razvilo u poslednjih par godina. Sve više ljudi u svoju biografiju dodaje i posao influensera kao izvor dodatne zarade ili čak primarno zanimanje. Platforma koju mnogi koriste kada žele da se bave ovim poslom je upravo youtube. Na njoj je moguće objavljivanje snimaka različitog vremenskog trajanja i sadržaja. Moderni influenseri su u trci za što većom popularnošću kako bi sa njom i njihova zarada porasla.

Analizirajući tekstualne podatke o svakom snimku, modeli će pokušati da dođu do zaključka šta to jedan snimak čini

popularnim na ovoj platformi. Koliko se na osnovu naslova snimka, naziva autora, tagova i deskripcije može pretpostaviti da li će se snimak dopasti ljudima koji ovu platformu koriste da ispune svakodnevno vreme.

Problem koji se rešava je multiklasna klasifikacija. Za ovaj problem iskorišćena je fine-tuning tehnika nad već postojećim jezičkim modelima BERT, BART i XLNet. Ova tri modela koriste različite tehnike za rad sa tekstom i iz tog razloga će se porediti koja od postojećih tehnika najbolje rešava ovaj problem.

U svrhu ovog istraživanja trenirani su i validirani i veliki i mali modeli. Korišćena su dva skupa podataka tako što je svaki podeljen u razmeri 80:20 za trening i validaciju.

Najbolje se pokazao BERT large jezički model sa čak 69% tačnosti. Rezultate koje su XLNet i BART dali nisu mnogo lošiji od BERT-a, ali je kod njih bilo više naznaka da počinju da overfituju nego kod BERT-a.

Prvo poglavlje daje neki opšti uvid u rad, koji je problem i kako će biti rešen. Drugo poglavlje predstavlja uvid u radove koji se bave sličnim problemom i koriste slične tehnologije za rad. Potom, u trećem se vidi koja je to korišćena metodologija, kakvi su podaci, šta podrazumeva pretprocesiranje. Četvrto poglavlje je uvid u eksperimente i rezultate sa kratkom diskusijom i na kraju zaključkom koji je proizašao iz ovog rada.

2. PRETHODNA REŠENJA

U radu [1] korišćena je tehnika fine-tuning XLNet modela za detekciju lažnih vesti na društvenim mrežama. Skup podataka LIAR [2] sadrži oko 12800 vesti. Skup je podeljen na trening (10269), validacioni (1284) i test skup (1283). Od ovog skupa napravljena su dva klasifikaciona problema. Prvi je višeklasni gde postoji šest različitih kategorija, a to su istinite, neistinite, skoro istinite, polu istinite, uglavnom istinite i potpuno neistinite. Drugi problem je binarne prirode, gde postoje samo dve klase, istinite i lažne vesti. U ovom skupu podataka nalaze se podaci: ko je autor, koja je tema, država, posao, zabava, kakav je kredibilitet i koliko je istinita. Modelu je dodat izlazni sloj kako bi izračunao verovatnoću za svaku kategoriju. Najveća vrednost neke kategorije predstavlja konačan rezultat klasifikacije. Model se trenirao pet epoha, batch size je bio veličine 32 i korišćen je AdamW optimizator sa 1e-5 za learning rate vrednost. XLNet je

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

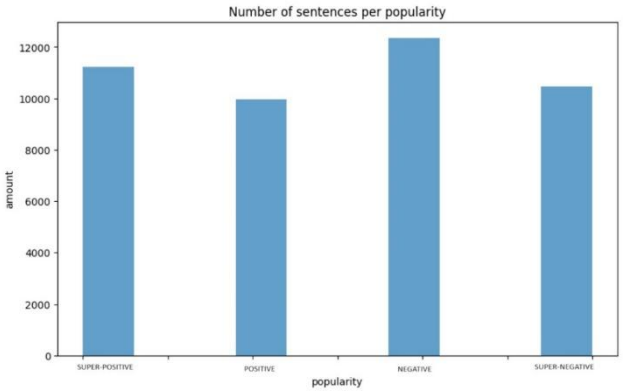
dao tačnost od 71%, BERT je za isti problem i podatke dao 62% za binarnu klasifikaciju, dok za multiklasnu klasifikaciju XLNet daje tačnost od 44%.

U radu [3] studija uvodi model za automatsku procenu povratnih informacija koristeći BERT jezički model. Model je treniran na skupu podataka od 10000 povratnih informacija koje su označene kao dobre ili loše. Dalje usavršavanje modela rađeno je na skupu podataka OULA. Model je dostigao tačnost od 93,4% na validacionom skupu, što ukazuje na njegovu sposobnost da proceni kvalitet povratnih informacija. Što se tiče pretprocesiranja rađeno je čišćenje, tokenizacija, feature extracting odnosno transformacija teksta u sekvencu embedinga koji sadrže semantičko značenje. BERT je poređen sa *Support Vector Machine (SVM)* algoritmom, *Random Forest, K-Nearest Neighbors, Logistic Regression*, Naivnim bajesom, stablom odlučivanja, Konvolutivinim neuronskim mrežama. BERT je dao najbolje rezultate po svim korišćenim metrikama (*F1, Recall, Precision*). Takođe, upoređen je i sa sebi sličnim modelima *RoBERT-a, PARsBERT-a* i *SemBERT-a* gde je dao najbolje rezultate.

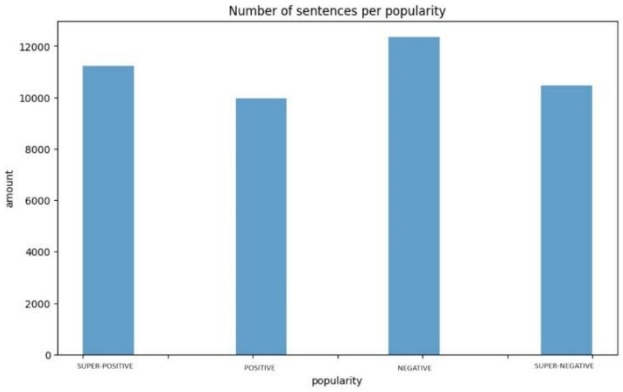
U radu [4] poredila su se tri modela : BERT, GPT i BART nad dva skupa podataka. Prvi je formiran od objava na twitter društvenoj mreži, a drugi od komentara na facebook-u. Prvo su testirani modeli bez treniranja, a zatim i posle treninga nad delom podataka. Prvi zadatak je bio da na osnovu twitter objave model zaključi kome je objava namenjena, Trampu ili Klintonovoj i da li je za, protiv ili neutralan. BART je pokazao najlošiji rezultat i najmanji F1 score dok je GPT dao najbolji rezultat. Slične rezultate dobili su i nakon treniranja. Čak je primećeno da je posle treninga narušena generalizacija koju BART ima.

3. FORMIRANJE SKUPA PODATAKA

Podaci su preuzeti sa sajta kaggle.com. Početni skup imao je 245987 podataka i ažurira se na dnevnom nivou. Kada su svi duplikati izbačeni ostalo je 44009 podataka sa jedinstvenim id-ijem. To predstavlja manji skup podataka nad kojim su se modeli trenirali i validirali. Veći skup nastao je proširivanjem prethodno pomenutog u cilju poboljšanja *recall* metrike. Veći skup podataka ima 67249 podatka i takođe su svi prikupljeni sa sajta *kaggle.com*. Kolone koje su korišćene su naziv snimka, naziv kanala, deskripcija i tagovi kao ulazni parametar, a labela koju je trebalo prediktovati je formirana na osnovu lajkova i dislajkova. Procenat broja dislajkova u odnosu na lajkove definisala je popularnost snimka. Naime, ako je procenat do 2% to je *super positive* klasa odnosno jako popularni videi. Zatim, od 2 do 5% snimci su *positive* odnosno popularni, od 5 do 10% *negative* odnosno nepopularni i preko 10% *super negative* odnosno veoma nepopularni. Na slikama 1. i 2. vidi se broj video snimaka po svakoj klasi. Naime, veći skup nastao je proširivanjem samo onim podacima koji čine *positive* i *negative* klasu jer je za njih uočen niži *recall* u odnosu na *super positive* i *super negative* klasu.



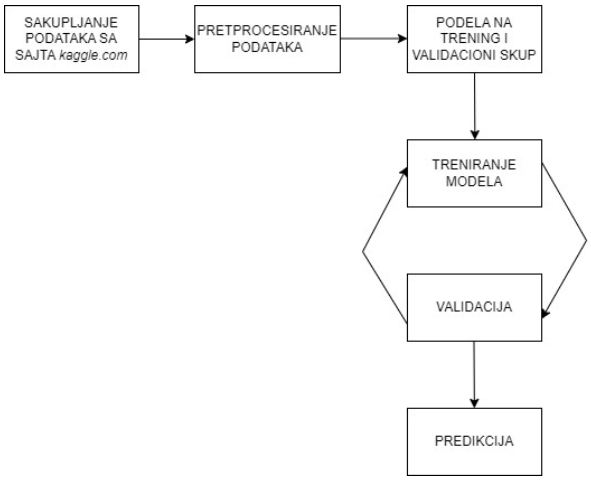
Slika 1. Broj video snimaka po klasama – mali skup



Slika 2. Broj video snimaka po klasama – veći skup

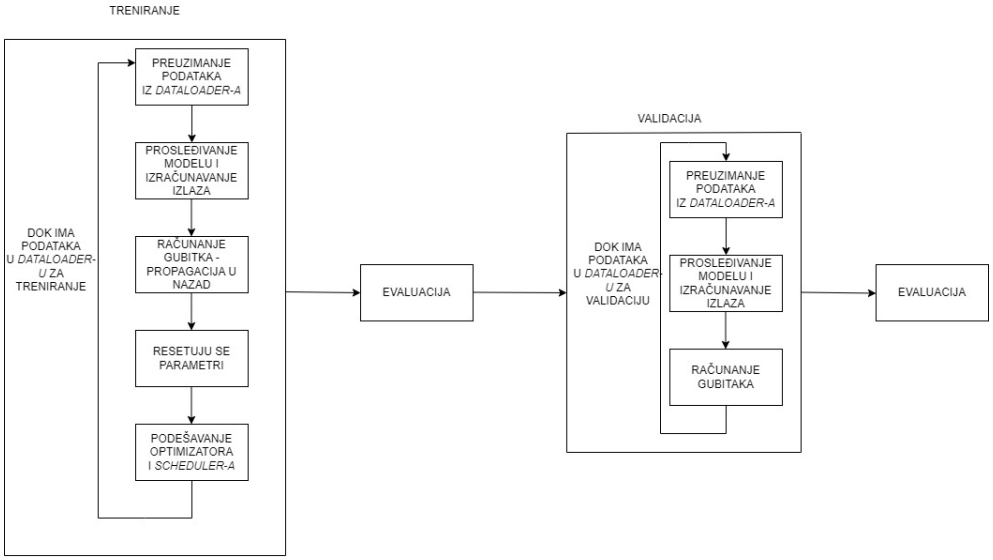
4. METODOLOGIJA

Dijagram na slici 3. prikazuje metodologiju rada.



Slika 3. Dijagram metodologije

Nakon dobavljanja podataka radi se pretprocesiranje koje uključuje tokenizaciju i stemovanje. Tako dobijeni skup podataka deli se u odnosu 80:20 na trening i validacioni skup. Definiše se *AdamW* optimizator kao i *scheduler*. Vrednosti *learning rate* i *batch size* su menjane tokom eksperimenata. Inicijalno su bile definisane četiri epohe u kojoj su se obavljali i trening i validacija. Na kraju svake epohe ispisivan je izveštaj metrika *recall, F1, precision, accuracy* i *loss* i za trening i za validaciju. Na taj način kontolisano je koliko model napreduje u učenju kao i *overfitting*. Na slici 4. prikazano je kako izgleda jedna epoha.



Slika 4. Tok jedne epohe

5. EKSPERIMENTI I REZULTATI

Eksperimenti su grupisani po modelima koji su korišćeni. Za svaki model obavljena su četiri eksperimenta, gde svaki predstavlja jedan od modela (*large* ili *base*) i jedan od skupa podataka (veći ili manji).

Naime, najbolji se pokazo BERT *large* model sa manjim skupom podataka, ali slične performanse daje i BERT *base* za veći skup podataka. *Loss* funkcija tokom epoha opada. U tabeli 1. prikazani su rezultati po klasama u četvrtoj epohi za vrednosti parametara learning rate = 1e-5 i batch size = 8 kada se koristi BERT large model i manji skup podataka.

Tabela 1. Metrike po klasama BERT large model

TRENING			
	precision	recall	F1
super positive	0.65	0.66	0.66
positive	0.58	0.51	0.54
negative	0.64	0.67	0.66
super negative	0.78	0.79	0.78
Macro avg	0.66	0.66	0.66
VALIDACIJA			
super positive	0.64	0.75	0.69
positive	0.61	0.54	0.57
negative	0.70	0.66	0.68
super negative	0.80	0.80	0.80
Macro avg	0.69	0.69	0.69

Tačnost koju je taj model postigao je 0.69 što je ujedno najbolja tačnost celokupnog eksperimenta.

Nakon BERT-a ista tehnika sa istim podacima pokušana je i sa BART modelom. Naime ni *large* ni *base* nisu uspeli da prevaziđu BERT-ove rezultate. Veći skup podataka je pomogao da se rezultati poboljšaju, posebno kod large modela gde BART dostiže tačnost od 0.67. U tabeli 2. prikazani su rezultati BART *large* modela kada je korišćen veliki skup podataka.

Tabela 2. Metrike po klasama BART large model

TRENING			
	precision	recall	F1
super positive	0.65	0.81	0.72
positive	0.78	0.72	0.75
negative	0.75	0.65	0.70
super negative	0.77	0.85	0.81
Macro avg	0.74	0.76	0.74
VALIDACIJA			
super positive	0.55	0.73	0.63
positive	0.73	0.69	0.71
negative	0.69	0.57	0.62
super negative	0.70	0.75	0.73
Macro avg	0.67	0.69	0.67

Treća grupa eksperimenata uključuje *large* i *base* XLNet modele kao i iste skupove podataka koji su se koristili i za prethodne eksperimente. U tabeli 3. predstavljeni su najbolje ostvareni rezultati sa XLNet modelom, odnosno rezultati nakon četvrte epohe treniranja XLNet large modela sa velikim skupom podataka. Tačnost koja je postignuta iznosi 0.67.

Tabela 3. Metrike po klasama XLNet large modeli

TRENING			
	precision	recall	F1
super positive	0.68	0.82	0.75
positive	0.77	0.74	0.75
negative	0.77	0.68	0.72
super negative	0.80	0.87	0.83
Macro avg	0.76	0.78	0.76
VALIDACIJA			
super positive	0.57	0.70	0.63
positive	0.73	0.68	0.70
negative	0.68	0.59	0.63
super negative	0.69	0.77	0.73
Macro avg	0.67	0.69	0.67

U tabeli 4. nalaze se tačnosti svakog eksperimenta za svaki skup podataka. Na osnovu nje možemo uporediti koji model se kako pokazao.

Tabela 4. Tačnost po modelima za skupove podataka

MODEL	TAČNOST ZA MANJI SKUP	TAČNOST ZA VEĆI SKUP
BERT base	0.59	0.67
BERT large	**0.69**	0.66
BART base	0.26	0.22
BART large	0.24	0.21
XLNet base	0.26	0.29
XLNet large	0.26	0.19

Iako se iz tabele iznad vidi da najbolju tačnost daje BERT large za manji skup, potrebno je deset sati treniranja i L4 GPU, dok za manji base model i veći skup podataka potrebno je duplo manje vreme i T4 GPU, a tačnost nije mnogo manja. XLNet i BART dali su malo lošiji rezultat. BART ne zahteva dugo treniranje, odnsno najmanje vremena je bilo potrebno upravo za njegovo obučavanje. Supotrno se ispostavilo za XLNet model čije treniranje je trajalo i preko šesnaest sati.

Analizom podataka za koje BERT nije dao dobar rezultat uočeno je da se među dosta primera nalaze naslovi koji u sebi sadrže link do nekog web sajta ili drugog youtube videa.

6. ZAKLJUČAK

U ovom radu predstavljena je komparativna analiza velikih jezičkih modela na istom problemu sa istim skupovima podataka. Upoređeni su modeli koji imaju različitu arhitekturu na istom klasifikacionom problemu. Motivacija za rad proizilazi iz sve veće popularnosti influenserskog zanimanja i pitanja da li će u budućnosti postojati jedan virtuelni asistent kao savetnik kako napraviti popularan youtube snimak.

Svaki eksperiment je tekao slično. Prvo su učitani i filtrirani podaci, tako da duplikati budu izbačeni. Potom su pretprocesirani podaci što uključuje tokenizaciju i stemovanje. Skup podataka podeljen je na trening i validacioni i inicijalizovane su četiri epohe.

Na osnovu rezultata uočava se da je BERT jedini model koji je uspeo da savlada ovaj zadatak u poređenju sa ostala dva modela. Tačnost u najboljem slučaju je 0.69, dok druga dva modela jako malo zaostaju za njim.

Kada sumiramo sve eksperimente vidimo da BERT base model nad većim skupom podataka daje tačnost 0.67 dok BERT large sa manjim skupom podataka daje 0.69. Treba uzeti u obzir da za base model treba duplo manje vremena i dovoljan je manji GPU, pa je zbog toga možda on optimalnije rešenje.

7. LITERATURA

[1] Kumar, Ashok; Trueman, Tina Esther; Cambria, Erik. Fake news detection using XLNet fine-tuning model. In: *2021 International Conference on Computational Intelligence and Computing Applications (ICCICA)*. IEEE, 2021. p. 1-4.

[2] W. Y. Wang, „Liar, liar, pants on fire“ : A new benchmark dataset for fake news detection, *in Proceedings of 55th Annual Meeting of Association for Computational Linguistics* 2017, pp 2931-2937

[3] Spristav, Gaurav; Kant, Shri; Spristava, Durgesh. Design of an AI-Driven Feedback and Decision Analysis in Online Learning with Google BERT. *International Journal of Intelligent Systems and Applications in Engineering*, 2024, 12.10s: 629–643-629–643

[4] Chae, Youngjin; Davidson, Thomas. Large language models for text classification: From zero-shot learning to fine-tuning. *Open Science Foundation*, 2023.

Kratka biografija:

Nela Jović rođena je u Zvorniku 2000. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Energetska elektronika i električne mašine odbranila je 2024.god.
kontakt: nelanekijovic@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.41
DOI: https://doi.org/10.24867/30BE03Blagojevic

ВИРТУЕЛНИ АСИСТЕНТ БАЗИРАН НА РАДУ ТРАНСФОРМЕР МОДЕЛА УЗ КОРИШЋЕЊЕ ВЕКТОРСКЕ БАЗЕ ПОДАТАКА

A VIRTUAL ASSISTANT BASED ON TRANSFORMER MODEL USING A VECTOR DATABASE

Његош Благојевић, *Факултет техничких наука, Нови Сад*

Област – УПРАВЉАЊЕ ДИГИТАЛНИМ ДОКУМЕНТИМА

Кратак садржај – *У овом раду биће приказана имплементација виртуелног асистента базираном на раду трансформер модела уз коришћење векторске базе података и напредних техника вештачке интелигенције. Систем ће бити специјализован на одређеном скупу података везаних за Факултет техничких наука у Новом Саду. Циљ система је олакшавање долажења до информација о факултету студентима који желе да упишу техничке науке.*

Кључне речи: *Семантичка претрага докумената, Трансформер модели, Виртуелни асистенти, Обрада природног језика.*

Abstract – *This paper presents the implementation of a virtual assistant based on transformer model using a vector database and advanced artificial intelligence techniques. The system will be specialized in a specific dataset related to the Faculty of Technical Sciences in Novi Sad. The goal of the system is to make it easier for students who want to enroll in technical sciences to get information about the faculty.*

Keywords: *Semantic search, transformer models, virtual assistants, natural language processing*

1. УВОД

Брз развој вештачке интелигенције и алгоритама машинског учења довео је до тога да технологија тренутно може, у мањој или већој мери, да замени човека у одређеним областима. Све више се развијају система који могу обрадити велике количине података и пружити корисне информације у реалном времену чиме се значајно повећава продуктивност и ефикасност.

У овом раду биће представљено решење које се базира на семантичкој претрази докумената које има за циљ добављање релевантних докумената за дати упит у реалном времену. Основна идеја је да се коришћењем модерних техника вештачке интелиген-ције, као што су трансформер модели, омогући пре-цизно и брзо проналажење информација у великом корпусу текстуалних података.

Конкретно, биће приказано решење које представља виртуелног асистента специјализованог на подацима везаним за Факултет техничких наука у Новом Саду. Овај алат ради на принципу семантичке претраге докумената где се документи чувају у виду вектора у векторској бази података. Упити корисника се такође преводе у векторе и на основу сличности се добављају релевантни подаци.

У овом раду најпре ће бити извршен преглед стања у области уз осврт на актуелна најсавременија решења. Потом ће бити описана архитектура и спецификација система након којих ће бити представљени најзначајнији имплементациони детаљи. Потом ће бити представљени и дискутовани резултати мерења перформанси алата у односу на друге, традиционалне начине претраге докумената. На крају, биће представљени закључци рада уз дискусију о могућим правцима даљег развоја.

2. ПРЕГЛЕД СТАЊА У ОБЛАСТИ

Обрада природног језика (*енг natural language processing*) је област вештачке интелигенције и лингвистике која се бави проучавањем проблема аутоматског производњења и разумевања природних људских језика [1].

Значајну прекретницу у развоју области обраде природног језика представља и настанак модела машинског учења од којих су најбитнији трансформер модели [2]. Примери ових модела јесу ”*BERT*”, “*GPT-3*” и разне варијанте настале од ових модела. Тренсформери су нашли примену у разним областима, од семантичке претраге, аутоматског превођења али и виртуелних асистената.

2.1. Виртуелни асистенти

Виртуелни асистенти представљају посебне алате и софтвере који могу да обављају низ задатака или услуга за корисника на основу корисничког уноса кроз текстуалне, али и вербалне команде.

Да развој ових софтвера представља примамљиво поље истраживања говори и чињеница да готово већина великих компанија из сфере софтверских информација улажу огроман новац у развој сопствених решења, тако да су неки од најпопуларнијих примера оваквих агената “*SIRI*” компаније “*Apple*”, амазонова “*Alexa*”, “*Google Assistant*” и “*Bixby*” компаније *Samsung*.

НАПОМЕНА:
Овај рад је проистекао из мастер рада чији ментор је био др Драган Ивановић, ред. проф.

2.2. *ChatGPT*

ChatGPT је модел заснован на машинском учењу који користи дубоке невронске мреже за генерисање текста [3]. Овај модел је трениран на великом обиму текстуалних података из различитих извора, укључујући интернет, књиге и новинске чланке. Његова основна функција је да одговара на корисничке упите и успостави дијалог са њим.

Шта чини *ChatGPT* тако напредним је његова способност да производи природан и разнообразан текст који одговара на различите захтеве корисника. Напредни алгоритми обраде природног језика омогућавају разумевање контекста упита и генерисање релевантних одговора, често са изненађујућом тачношћу.

3. КОРИШЋЕНЕ ТЕХНИКЕ И ТЕХНОЛОГИЈЕ

Серверска страна апликације развијана је уз коришћење *Flask* радног окружења у *Python* програмском језику.

Клијентска страна система је развијена уз помоћ *React* библиотеке у *JavaScript* програмском језику.

Као што је већ напоменуто, за складиштење података је коришћена векторска база података. Векторске базе података су тип базе података која складишти податке у векторском формату, што омогућава ефикасно извршавање упита и манипулацију огромним скуповима података [4]. За овај систем коришћена је *Pinecone* база података.

Обзиром да систем ради са великим бројем захтева који представљају питања корисника која се често понављају, наопходно је оптимизовати систем коришћењем технике кеширања. *Redis* представља брзу и флексибилну *in-memory* базу података намењену најчешће за кеширање података. Карактерише је брз приступ подацима и решавање проблема складишта малих количина података.

3.1. Трансформер модели

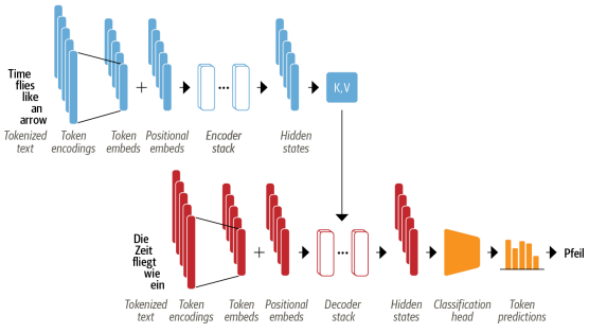
Трансформер модели представљају моделе дубоког учења развијене од стране *Google-a* 2017. године. Тада су представљени као замена за тадашње доминантне моделе трансформисања који су се ослањали на комплексне рекурентне или конволуционе неуронске мреже које укључују енкодер и декодер архитектуру кроз механизам пажње [5]. Трансформери су предложени као једноставније решење које ће се ослањати само на механизам пажње.

За разлику од традиционалних енкодер-декодер модела који су посматрали улазну секвенцу униформно (обрађиван је елемент по елемент), механизам пажње омогућава селективни приступ различитим деловина улазних података додељујући различите степене битности [6].

Механизам пажње је првобитно имплементиран у оквиру машинског превођења кроз модел назван *RNNsearch*. Овај модел користи двосмерну рекурентну неуронску мрежу као енкодер и декодер који опонаша претраживање кроз изворну реченицу приликом генерисања превода [6].

Енкодер израчунава скривена стања (анотације) за сваку реч у улазној секвенци, док декодер користи блок пажње за динамичко одређивање који делови улазне секвенце су најважнији за генерисање сваког следећег симбола у излазу [7].

На следећој слици је представљена архитектура енкодер-декодер трансформера узета из овог рада [7].



Слика 1. *Архитектура*

Оваква трансформер архитектура је оригинално била дизајнирана за *sequence to sequence* задатке попут машинског превођења. Међутим, временом су и енкодер и декодер постали посебне компоненте тако да сада постоји више трансформер модела базирано на томе које компоненте садржи.

3.1. Модели за генерисање одговора

Модели за одговор на питања (*QA* модели) су једна од најпопуларнијих области у обради природних језика. Ови модели имају способност да разумеју и генеришу одговоре на питања постављена на природном језику, што их чини изузетно корисним у бројним апликацијама, укључујући виртуелне асистенте, претраживаче и системе за подршку корисницима.

QA модели функционишу тако што обрађују улазни текст, који се састоји од пасуса (контекста) и питања, и затим генеришу одговор који је обично фраза или реченица из контекста. Уопштено, ови модели могу бити класификовани у две категорије:

- Екстрактивни модели
- Генеративни модели

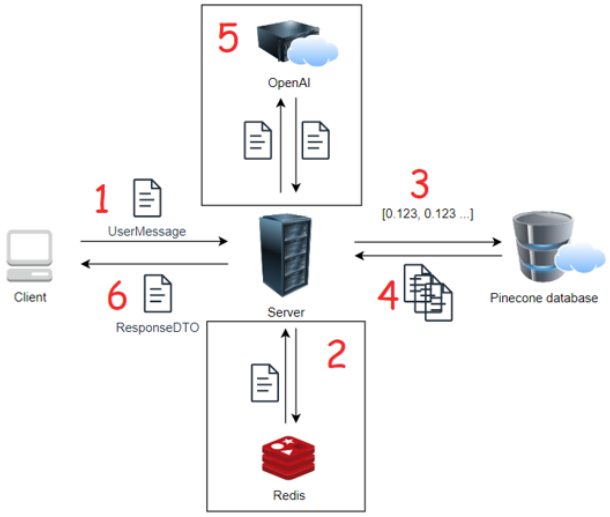
У оквиру овог рада нису имплементирани ови модели него се рад система ослања на коришћење *OpenAI* интерфејса и њиховог модела за генерисање одговора.

4. СПЕЦИФИКАЦИЈА

Систем се састоји од два главна дела, клијентске апликације са корисничким интерфејсом и серверске апликације. Серверска апликација комуницира са базом података преко *Pinecone* интерфејса али и користи *OpenAI* интерфејс за генерисање одговора на основу докумената добијених из базе података.

На следећој слици је приказана архитектура система са свим његовим модулима и приказаним корацима у

извршавању главног корисничког случаја, комуникације са асистентом.



Слика 2. *Архитектура и кораци извршавања*

У оквиру корака један, корисник уноси питање и шаље се захтев серверу. Питање је произвољног облика, може бити или реч или реченица. Пошто је могуће да се питања понављају проверава се у оквиру *Redis-a* да ли је дато питање већ било и враћа се кеширан одговор. Тиме се додатно оптимизује извршавање. Уколико то није сличај, питање се векторизује коришћењем истог модела који је коришћен за креирање базе података. Тај вектор се шаље бази за претрагу. База враћа листу од три најрелевантнија документа. Контексти се извлаче из резултата и шаље се упит преко *OpenAI* интерфејса за генерисање одговора на основу контекста. Одговор се шаље клијентској страни заједно за контекстима од кога је настао одговор.

5. ИМПЛЕМЕНТАЦИЈА

5.1. Скуп података

За задовољавајући рад трансформер модела неопходно је постојање добро припремљеног скупа података на коме би се модел могао тренирати. Пронађен је скуп података који је описан у раду [8] и који је синтетички направљен превођењем *SQUAD* скупа података са енглеског језика.

SQUAD скуп садржи око 87 хиљада елемената скупљених са разних извора, најчешће артикли са *Wikipedia*-е заједно са питањима везаним за тај артикал који су ручно формирани од стране великог броја људи.

Коришћени скуп података је дакле преведен скуп података на српски језик коришћењем *Translate-Align-Retrieve* методе описане у овом раду [21]. Начин на који је искоришћен скуп података полази од претпоставке да ће систем радити са питањима о факултету и контекстима који су издвојени део текста са сајта факултета или било ког другог извора, тако да се може искористити контекст и питање из *SQUAD* скупа података. Да би се трансформер модел могао тренирати потребно је да постоје нумеричке лабеле које одређују сличност између контекста и питања. Пошто у овом скупу лабеле не постоје, синтетички су креиране коришћењем више трансформер модела над скупом података на енглеском језику. Тиме се добијају релевантне лабеле које су касније коришћене у комбинацији са скупом података на српском језику.

5.2. Модел

Модел који је коришћен као основа за *finetuning* је *sentence-transformers_paraphrase-multilingual-MiniLM-L12-v2* . Овај модел мапира реченице и параграфе на векторе са 384 елемента. Модел се састоји од 118 милиона параметара и трениран је на 50 различитих језика, укључујући и српски.

Разлог зашто је изабран овај модел је управо тај што је у основи трениран и за српски језик. Тиме се у његовом речнику налазе и српске речи и способан је да ради и са текстом на српском језику. То је било неопходно да би се модел могао накнадно претренирати над скупом података на српском језику.

Скуп података је подељен на тренинг и валидациони део у односу 80-20. Функција губитка која је коришћена је *cosine similarity loss* а евалуатор је *embedding similarity evaluator*.

Batch size је подешен на 16 а број епоха је износио 7.

5.3. База података

Да би систем могао да функционише неопходно је имати базу података са релевантним информацијама о факултету. Најбољи извор информација је био званични сајт факултета тако да је већина информација прикупљена са сајта методом *web scraping*-а као текст директно из *HTML*-а. Такође, ручно су додате и информације са осталих извора као што су групе за упис на факултет које садрже одговоре на разна питања о факултету па су послужили као идеалан скуп података за овај систем. Једна од таквих група је и *Facebook* група „Бићу студент ФТН – упис 2024“ са најрелевантнијим подацима везаним за факултет.

6. РЕЗУЛТАТИ

Када се говори о резултатима модела трансформера, јако је тешко нумерички одредити прецизност алата управо због, како је већ наведено, синтетичког генерисања скупа података над којим је модел трениран. Чак и да то није случај, додатну проблематику представља чињеница да сличност два текста зависи и од самих субјеката који раде лабелирање, а потпуна униформност приликом лабелирања је готово немогућа.

С тим у вези, иако ће се приказати евалуациони резултати добијени приликом тренирања, ово поглавље ће бити више посвећено упоређивању овог приступа са класичним претрагама докумената коришћењем *elastic search-a*. Што се тиче евалуационих резултата модел евалуиран коришћењем различитих метричких приступа. Након тренирања модел је показивао тачност од 0.843 у

косинусној сличности, 0.774 у еуклидској и 0.763 у менхетенској. Међутим, најбоља евалуација оваквих модела је тестирање и упоређивање са системима који користе другачији приступ за претрагу докумената.

Систем је упоређиван са класичним системом за претрагу докумената користећи упите различитих дужина. Истраживање је показало да се класични системи показују боље само када су упити доста кратки и када немамо довољно контекста за семантичку претрагу. Тако да ситуације у којима ће класични системи боље радити јесу претраге по кључним речима или по одређеним фразама. Повећавањем упита и проширивањем контекста систем базиран на трансформер моделу показује боље резултате док највећу предност показује у ситуацијама када се упит састоји од речи које не постоје у документу већ су само семантички сличне. У таквим ситуацијама класични системи постају готово неупотребљиви.

7. ЗАКЉУЧАК

У овом раду је представљен један начин имплементације виртуелног асистента специфицираног на одређен домен знања (у овом случају информације се примарно добављају са сајта Факултета техничких наука у Новом Саду). Циљ оваквог система је приближавање информација факултета техничких наука корисницима а све у циљу повећања доступности информација о самом факултету. Имплементација решења је обухватала различите кораке и делове система који се могу посматрати независно.

Први корак у имплементацији је прикупљање базе знања. За овај конкретан случај, документи за базу су прикупљени са сајта методом *web scraping*-а.

Други део система је систем за претрагу докумената базиран на трасформер моделима. Конкретно за овај систем је коришћен претраниран модел *sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2* који је дотрениран подацима на српском језику.

Трећи део јесте генерисање одговора на основу упита и резултата. Због недостатка података за тренирање генеративних модела за одговарање, коришћен је *OpenAI* интерфејс којем су прослеђивани резултати добављени у претходној фази.

Што се тиче корисничког дела, апликација се може користити као виртуелни асистент у оквиру сајта факултета, као посебна апликација за дописивање и постављање питања али и кроз административни режим за лаку манипулацију подацима из базе података.

Када говоримо о начинима надоградње система, могућности су разне. Полазећи од добављања података за базу који се могу аутоматизовати и чији извор може бити, како сајт, тако и приватан архив факултета. Са повећањем докумената у бази повећава се и спектар могућих питања и одговора са којима систем може да ради.

Трансформер модели се развијају свакодневно тако да ће вероватно у скорије време бити унапређени и вишејезични модели који обухватају и српски језик пошто у овом тренутку само мало број модела ради са српским језиком а и скупови података за тренирање готово да ни не постоје на српском језику. Унапређењем овог дела би се унапредила и претрага докумената тако да би увек имали најрелевантније документе припремљене за наредну фазу.

Последња фаза уједно оставља и највише простора за напредовањем где би циљ био креирање сопственог модела за одговарање на питање у слободној форми. Такође, модел би морао да буде способан за рад са документима на српском језику.

8. LITERATURA

[1] Chowdhary, K.R. (2020). Natural Language Processing. In: Fundamentals of Artificial Intelligence. Springer, New Delhi. https://doi.org/10.1007/978-81 322-3972-7_19

[2] A. Gillioz, J. Casas, E. Mugellini and O. A. Khaled, "Overview of the Transformer-based Models for NLP Tasks," 2020 15th Conference on Computer Science and Information Systems (FedCSIS), Sofia, Bulgaria, 2020, pp. 179-183, doi: 10.15439/2020F20.

[3] An, J., W. Ding, and C. Lin. "ChatGPT." tackle the growing carbon footprint of generative AI 615 (2023): 586.

[4] Han, Yikun, Chunjiang Liu, and Pengfei Wang. "A comprehensive survey on vector database: Storage and retrieval technique, challenge." arXiv preprint arXiv:2310.11703 (2023).

[5] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

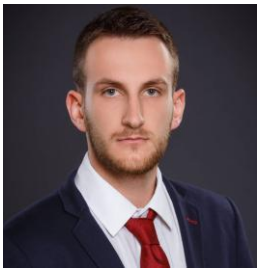
[6] Niu, Zhaoyang, Guoqiang Zhong, and Hui Yu. "A review on the attention mechanism of deep learning." Neurocomputing 452 (2021): 48-62.

[7] Tunstall, Lewis, Leandro Von Werra, and Thomas Wolf. Natural language processing with transformers. " O'Reilly Media, Inc.", 2022.

[8] Cvetanović, Aleksa, and Predrag Tadić. "Synthetic Dataset Creation and Fine-Tuning of Transformer Models for Question Answering in Serbian." 2023 31st Telecommunications Forum (TELFOR). IEEE, 2023.

[9] Carrino, Casimiro Pio, Marta R. Costa-Jussà, and José AR Fonollosa. "Automatic spanish translation of the squad dataset for multilingual question answering." *arXiv preprint arXiv:1912.05200* (2019).

Кратка биографија:



Његош Благојевић рођен је 29. 04. 1999. у Зворнику. Смер софтверско инжењерство и информационе технологиј на Факултету техничких наука у Новом Саду уписао је 2018. године. Основне студије је завршио у септембру 2022.године и од октобра 2022. године студира на мастер студијама Факултета техничких наука.

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.41
DOI: https://doi.org/10.24867/30BE04Cosovic

CLOUD АПЛИКАЦИЈА ЗА ПРЕТРАГУ ФИЛМОВА БАЗИРАНА НА OPENSEARCH-У

CLOUD BASED APPLICATION FOR MOVIE SEARCH USING OPENSEARCH

Матеја Ћосовић, *Факултет техничких наука, Нови Сад*

Област – ПРИМЕЊЕНЕ РАЧУНАРСКЕ НАУКЕ И ИНФОРМАТИКА

Кратак садржај – *Рад се бави развојем апликације за претрагу филмова користећи AWS сервисе од којих је најзначајнији OpenSearch Service. У раду је дат теоријски опис свих коришћених AWS сервиси, програмских језика, радних оквира и алата. Такође, детаљно је описана архитектура, дизајн и модел података апликације, као и испуњени функционални и нефункционални захтеви. На крају су приказана три експеримента над апликацијом како би се измерile перформансе.*

Кључне речи: *AWS, cloud, филмови, serverless, OpenSearch, AWS CDK, Lambda*

Abstract – *The paper focuses on the development of a movie search application using AWS services, with the most significant being OpenSearch Service. The paper provides a theoretical description of all AWS services, programming languages, frameworks, and tools used during development. Additionally, the architecture, design, and data model of the application are described in detail, along with the fulfilled functional and non-functional requirements. Finally, three experiments on the application are presented to measure its performance.*

Keywords: *AWS, cloud, movies, serverless, OpenSearch, AWS CDK, Lambda*

1. УВОД

Љубав према филмовима је универзална и спаја људе широм света. Први филмови настали су пре више од сто година и снимани су широм земаљске кугле [1]. Број филмова експоненцијално расте из године у годину и гледаоцима је веома тешко пронаћи баш онај филм који им одговара у датом тренутку. Решавање овог проблема представља мотивацију која стоји иза овог рада.

Cloud і *serverless* технологије постају све популарније због својих предности у односу на традиционалне технологије које користе локалне ресурсе [2]. Пружају већу скалабилност, смањују трошкове одржавања и омогућавају брже развијање и имплементацију апликација. Амазонов AWS (*Amazon Web Services*) као лидер у клауд технологијама [3] нуди широк спектар *serverless* услуга које олакшавају развој апликација.

У раду ће детаљно бити описани кораци развоја апликације, укључујући дизајн система, избор технологија и имплементација, као и анализу перформанси решења.

Циљ је да се покаже како се *serverless* технологија може искористити за имплементацију ефикасног система за претрагу филмова.

2. ТЕОРИЈСКЕ ОСНОВЕ

2.1. Увод у *cloud* технологије

Клауд (енгл. *cloud*) технологије, или технологије у облаку, представљају приступ рачунарству где се ресурси као што су складиште података, процесорска снага и апликације не налазе локално на рачунару корисника, већ су доступни путем интернета.

Овај приступ омогућава корисницима да приступају ресурсима са било ког уређаја повезаног на интернет, чиме се побољшава флексибилност, скалабилност и ефикасност пословања [4].

Серверлес (енгл. *serverless*) технологије су технологије где пружалац клауд услуга аутоматски управља инфраструктуром потребном за покретање апликација, тако да корисници могу да се фокусирају на писање и имплементацију кода без бриге о управљању инфраструктуром.

Коришћење клауд и серверлес услуга доноси бројне предности које значајно унапређују ефикасност и конкурентност организација [5]. Првенствено, смањују се трошкови јер корисници плаћају само за ресурсе које користе, без потребе за великим почетним улагањима у хардвер и софтвер.

Коришћење клауд платформи омогућава брзу имплементацију нових технологија и сервиса, подржавајући иновације. Поред тога, клауд провајдери често користе енергетски ефикасне дата центре, чиме доприносе еколошкој одрживости [6].

2.2. AWS платформа

Amazon Web Services (AWS) је водећа платформа за клауд рачунарство, која пружа широк спектар услуга као што су процесирање, базе података, аналитику и многе друге. Основан 2006. године од стране Amazona, AWS омогућава организацијама свих величина да ефикасно управљају својим IT ресурсима због његове скалабилности, флексибилности и поузданости и постаје кључни алат за компаније [7].

2.3. Amazon Lambda

Amazon Lambda је серверлес рачунска услуга која омогућава корисницима да покрећу код као одговор на догађаје, без потребе за управљањем серверима.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Драган Ивановић, ред. проф.

Ова услуга омогућава програмерима да се фокусирају на писање и оптимизацију кода, док се Amazon Lambda брине о основној инфраструктури. Код се извршава у оквиру високо доступног окружења, које аутоматски управља ресурсима потребним за покретање функција [8].
Једна од главних предности Lambda-е је способност да аутоматски скалира функције на основу броја долазних догађаја. Када дође до повећања оптерећења, Lambda аутоматски креира додатне инстанце како би обрадила повећани број захтева, а када оптерећење опадне, смањује се број инстанци.
Amazon Lambda представља моћан алат за модерну развојну праксу, омогућавајући брзу имплементацију, флексибилност и значајне уштеде. Њена способност аутоматског скалирања, интеграција са другим AWS услугама и робусна сигурносна архитектура чине је идеалним избором за широк спектар апликација, од једноставних до комплексних.

2.4. Amazon OpenSearch Service

Amazon OpenSearch Service је сервис који омогућава постављање, управљање и скалирање ElasticSearch тј. OpenSearch кластера на AWS инфраструктури. Намењен је за кориснике који желе моћне могућности претраге и анализе података, али желе да избегну сложености ручног управљања инфраструктурама и хостовања кластера самостално [9].
Amazon OpenSearch Service је дизајниран за високе перформансе, омогућавајући брзе претраге и анализе, чак и на великим скуповима података. Поред тога, модел наплате је заснован на стварној употреби ресурса, што омогућава оптимизацију трошкова. Корисници плаћају само за ресурсе које користе, укључујући инстанце, складиште и улазно-излазне операције, чиме се избегавају непотребни трошкови за неискоришћене ресурсе.

2.5. Amazon CloudFormation

Amazon CloudFormation је сервис за управљање инфраструктурама преко кода (Infrastructure as Code, IaC) који омогућава корисницима да дефинишу и аутоматски проводе ресурсе потребне за своје апликације на AWS-у. Коришћењем једноставних текстуалних датотека у JSON или YAML формату, корисници могу описати све AWS ресурсе које им апликација захтева.
CloudFormation аутоматски управља заузимањем и конфигурисањем ресурса, што смањује сложеност и ризик од људских грешака.
Једна од главних предности CloudFormation-а јесте конзистентност и поновљивост у постављању инфраструктуре. Када је инфраструктура дефинисана као код, лако се може делити, прегледати и контролисати верзије, што омогућава развојним тимовима да сарађују ефикасније [11].
Поред тога, промена инфраструктуре постаје једноставнија и мање ризична, јер се све промене могу тестирати и валидирати пре него што се примене у производном окружењу.
Ово доводи до бржег циклуса развоја и веће поузданости апликација.

2.6. Amazon CDK

AWS Cloud Development Kit (CDK) је *open-source* радни оквир који омогућава програмерима да дефинишу и заузму своју AWS инфраструктуру користећи познате програмске језике [12]. CDK омогућава писање инфраструктурног кода на исти начин као и апликативног кода, чиме се поједностављује процес развоја и управљања ресурсима у клауду.

3. СПЕЦИФИКАЦИЈА АПЛИКАЦИЈЕ

3.1. Функционални захтеви

Функционални захтеви за апликацију су специфичне карактеристике и функције које систем мора да испуњава како би задовољио потребе корисника и пословне циљеве. Ови захтеви дефинишу шта систем треба да ради, укључујући корисничке интеракције, обраду података, управљање подацима и друге оперативне функције [13].
Главна функционалност апликације за претрагу филмова је, може се наслутити, напредна претрага и филтрирање филмова над великим скупом података. Такође, постоје додатне функционалности попут напредне статистике, уноса појединачног филма и великог броја филмова, логовања и регистрација, као и брисања филмова.

3.2. Нефункционални захтеви

Нефункционални захтеви за апликацију су критеријуми који дефинишу како систем треба да функционише, а не шта систем треба да ради. Ови захтеви обухватају аспекте као што су перформансе, безбедност, употребљивост, поузданост, скалабилност и одрживост система. Одабрани нефункционални захтеви за апликацију су перформансе, безбедност и скалабилност.

3.3. Архитектура система

Архитектура апликације састоји се од клијентског и серверског дела. У оба дела коришћени су различити AWS сервиси. За израду клијентског дела коришћени су Amazon CloudFront и Amazon S3 и React радни оквир, док су за израду серверског дела коришћени Amazon Lambda, Amazon API Gateway, Amazon DynamoDB, Amazon SQS, Amazon S3, Amazon OpenSearch Service и Python програмски језик. Цела архитектура је закупљена уз помоћ AWS CDK радног оквира. На слици 3.2. приказана је архитектура система.

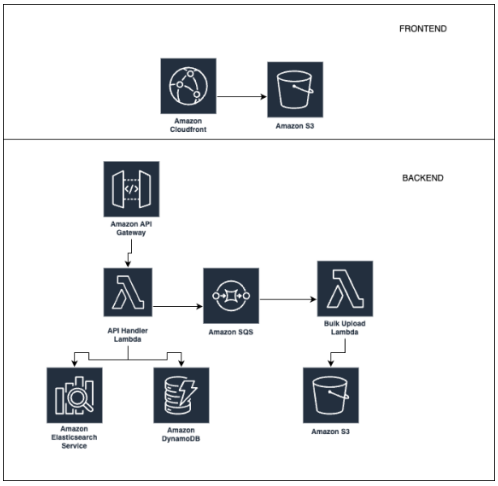
3.4. Дизајн система

За израду визуелно привлачног и респонзивног корисничког интерфејса коришћена је популарна MUI Material библиотека компоненти за React која пружа предефинисане стилске компоненте.
Апликација се састоји из следећих страница: Login, Registration, Search, Statistics и Upload.

3.5. Модел података

Модел података направљен је на основу случајева коришћења и представља структуру података који се

налазе у DynamoDB табели и OpenSearch домену. Ентитети система су корисник и филм.



Слика 3.2. Архитектура система

Како би што лакше унели велики број филмова у систем, а да притом филмови имају стварне вредности, одлучено је да се користи неки постојећи скуп података о филмовима са интернета. Изабран је *Full TMDB Movies Dataset 2024 (1M Movies)* са *Kaggle* интернет платформе.

4. ИМПЛЕМЕНТАЦИЈА РЕШЕЊА

4.1. Имплементација серверског дела апликације

Серверски део апликације написан је у Python програмском језику, конкретно користећи FastAPI радни оквир.

За организовање виртуалног окружења, изградњу пројекта за дистрибуцију и инсталацију пакета и зависности коришћен је poetry алат. Овај алат омогућава једноставно инсталирање свих потребних пакета, као и изградњу пројекта кроз покретање команди у терминалу и веома је лак за коришћење.

Апликација је организована по модулима, где је сваки модул задужен за одређене делове функционалности система и овај приступ организовања кода је традиционално виђен у пројектима заснованим у Python-у.

4.2. Имплементација клијентског дела апликације

Клијентска страна апликације развијена је користећи популарни React радни оквир. Према основном принципу React-а, примењивана је компонентна архитектура, односно апликација је подељена на компоненте како би мањи делови апликације били издвојени и лаки за одржавање.

4.3. Имплементација инфраструктуре апликације

За имплементацију инфраструктуре апликације је коришћен AWS Cloud Development Kit (CDK), моћан алат који омогућава програмерима да дефинишу cloud инфраструктуру користећи програмске језике.

У нашем случају, коришћен је Python програмски језик.

5. ЕКСПЕРИМЕНТИ И ПРИКАЗ РЕЗУЛТАТА

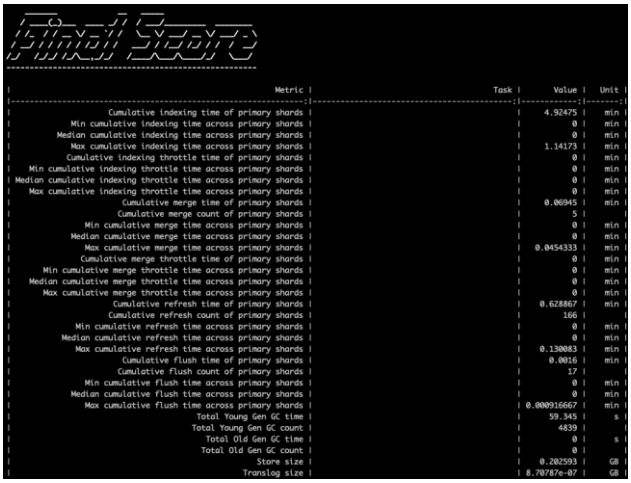
5.1. Први експеримент

У оквиру првог експеримента предложено је да се тестирају перформансе над различитим типовима инстанци и различитим бројем дата чворова. Конкретно, у раду су коришћене комбинације t3.small.search и t3.medium.search инстанци респективно са једним или пет чворова. Над сваком комбинацијом биће испробано индексирање од двеста хиљада докумената, затим претрага над њима, као и набављање детаљне статистике.

Забележени резултати слажу се са генералним претпоставкама о OpenSearch-у, броју дата чворова и типовима инстанци. Из резултата се може јасно видети да је претрага најбржа у случају 5 дата чворова t3.medium.search типа и просечно време извршавања износи 1.1 секунди, док је претрага најспорија случају једног чвора t3.small.search типа где просечно време износи 2.4 секунде.

5.2. Други експеримент

У оквиру другог експеримента предложено је да се над OpenSearch доменом покрене алат под именом OpenSearch Benchmark. Овај алат извршава тестирање над постојећим доменом са предефинисаним скупом података и прикупља детаљне информације о домену и његовим перформансама. На слици 5.1. може се видети резултат OpenSearch Benchmark-а.



Metric	Task	Value	Unit
Cumulative indexing time of primary shards		4.92475	min
Min cumulative indexing time across primary shards		0	min
Median cumulative indexing time across primary shards		0	min
Max cumulative indexing time across primary shards		1.14173	min
Cumulative indexing throttle time of primary shards		0	min
Min cumulative indexing throttle time across primary shards		0	min
Median cumulative indexing throttle time across primary shards		0	min
Max cumulative indexing throttle time across primary shards		0	min
Cumulative merge time of primary shards		0.00945	min
Cumulative merge count of primary shards		5	
Min cumulative merge time across primary shards		0	min
Median cumulative merge time across primary shards		0	min
Max cumulative merge time across primary shards		0.0454333	min
Cumulative merge throttle time of primary shards		0	min
Min cumulative merge throttle time across primary shards		0	min
Median cumulative merge throttle time across primary shards		0	min
Max cumulative merge throttle time across primary shards		0	min
Cumulative refresh time of primary shards		0.628867	min
Cumulative refresh count of primary shards		166	
Min cumulative refresh time across primary shards		0	min
Median cumulative refresh time across primary shards		0	min
Max cumulative refresh time across primary shards		0.130083	min
Cumulative flush time of primary shards		0.0016	min
Cumulative flush count of primary shards		17	
Min cumulative flush time across primary shards		0	min
Median cumulative flush time across primary shards		0	min
Max cumulative flush time across primary shards		0.000916667	min
Total Young Gen GC time		59.345	s
Total Young Gen GC count		4839	
Total Old Gen GC time		0	s
Total Old Gen GC count		0	
Store size		0.202593	GB
Translog size		8.70787e-07	GB

Слика 5.1. Резултат OpenSearch Benchmark-а

5.3. Трећи експеримент

У оквиру трећег експеримента предложено је да се прикаже како количина података утиче на перформансе. На табели 5.1. могу се видети резултати трећег експеримента.

Резултати забележени у трећем експерименту показују да што више података има у домену, то је спорије време извршавања упита, што је била и претпоставка пре извршавања експеримента.

Табела 5.1. Резултати трећег експеримента

Број докумената	**Претрага**	**Детаљна статистика**
10000 докумената	0.84 секунди	0.63 секунди
20000 докумената	0.78 секунди	0.59 секунди
50000 докумената	1.29 секунди	0.5 секунди
100000 докумената	2.3 секунди	0.62 секунди

6. ЗАКЉУЧАК

У раду је представљена cloud апликација за претрагу филмова која користи OpenSearch. Теоријски су описани све коришћене технологије и сервиси како би се разумело функционисање апликације.
Представљени су функционални и нефункционални захтеви које је апликације требало да задовољи и описани су имплементациони детаљи свих делова апликације. На крају је извршено три експеримента како би се упоредиле перформансе могућих верзија апликације.
Спроведени експерименти показују да је апликација успешно имплементирана, и осликавају њене перформансе. Такође, апликација је лако испоручива у било којем тренутку, то јест, може се наћи на интернету у веома кратком временском периоду. Једна од ствари о којој би се требало побринути у случају надоградње апликације је резервисање интернет домена путем AWS-овог Route53 сервиса. Додатно, скуп података би се могао проширити да укључује више од милион филмова, и претрага по више параметара него тренутно би могла бити омогућена. На крају, перформансе система би се могле побољшати тако што бисмо вертикално скалирали апликацију.

7. ЛИТЕРАТУРА

[1] Ishita Babbar. International Journal for Multidisciplinary Research (IJFMR) (2024). *Evolution of Cinema*, https://www.ijfmr.com/papers/2024/2/17578.pdf

[2] Zhang, Q., Cheng, L. & Boutaba, R. *Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl* **1**, 7–18 (2010). https://doi.org/10.1007/s13174-010-0007-6

[3] Bandaru, Avinash. (2020). *Amazon Web Service*, https://www.researchgate.net/publication/347442916_AMAZON_WEB_SERVICES

[4] Rafia Islam, Vardhan Patamsetti, Aparna Gadhi, Ragha Madhavi Gondu, Chinna Manikanta Bandaru, Sai Chaitanya Kesani, Olatunde Abiona
Department of Computer Information Systems, Indiana University Northwest, (2023), *The Future of Cloud Computing: Benefits and Challenges* 10.4236/ijcns.2023.164004

[5] Sether, Ayob. (2016). *Cloud Computing Benefits* 10.13140/RG.2.1.1776.0880.

[6] Mittal, Sparsh. (2014). *Power Management Techniques for Data Centers: A Survey*

[7] Mukherjee, Sourav. (2019). *Benefits of AWS in Modern Cloud*. 10.5281/zenodo.2587217.

[8] Shashank Srivastava, et al. (2023). Execution of Serverless Functions Lambda in AWS Serverless Environment. *International Journal on Recent and Innovation Trends in Computing and Communication*, *11*(9), 1081–1086. https://doi.org/10.17762/ijritcc.v11i9.9014

[9] "What is Amazon OpenSearch Service", AWS Documentation, https://docs.aws.amazon.com/opensearch-service/latest/developerguide/what-is.html

[10] Chauhan, Sidhartha & Cuthbert, Dave & Devine, James & Halachmi, Alan & Lehwess, Matt & Matthews, Nick & Morad, Steve & Seymour, Steve & Walker, Dave. (2018). Amazon CloudFront. 10.1002/9781119549000.ch7.

[11] "5 Benefits of Infrastructure as Code (IaC) for Modern Businesses in the Cloud", Technology Advisors, https://www.techadv.com/blog/5-benefits-infrastructure-code-iac-modern-businesses-cloud

[12] "What is the AWS CDK?", AWS Documentation, https://docs.aws.amazon.com/cdk/v2/guide/home.html

[13] Malan, Ruth & Bredemeyer, Dana. (2001). Functional Requirements and Use Cases.

Кратка биографија:

Матеја Ћосовић рођен је 1999. године у Сомбору. У школској 2018/19. уписује студијски програм Софтверско инжењерство и информационе технологије на Факултету техничких наука у Новом Саду. Основне студије завршио је 2022. године. Након завршених основних студија, уписује мастер студије на истом студијском програму.
контакт:
matejacosovic@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 7.005
DOI: https://doi.org/10.24867/30BE05Volaric

KOLABORATIVNA PLATFORMA ZA RAZVOJ REACT APLIKACIJA

COLLABORATIVE PLATFORM FOR DEVELOPING REACT APPLICATIONS

Filip Volarić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Cilj rada jeste istraživanje, projektovanje i razvijanje veb alata koji će omogućiti modelovanje i generisanje React koda za klijentske aplikacije. Kreiran je alat koji omogućuje korisnicima da u timovima razvijaju modele za React projekte, i da na osnovu modela generišu funkcionalne projekte. Osnovna gradivna jedinica React projekta je komponenta, pa je za njihovo modelovanje pružena najveća podrška. Generisani projekat se može pokrenuti pomoću Vite alata za izgradnju, dok se klijentske komponente mogu koristiti i u okviru Next.js radnog okvira.*

Ključne reči: *React, model, generator koda, Vite, komponente*

Abstract – *The work aims to research, design, and develop a web tool for modeling React applications, and generating the code for client applications. The tool enables users to work in teams when developing models for React projects. The basic building block of React application is a component. Therefore, their modeling offers the greatest support. Generated project can be ran with the Vite build tool, but also, generated components can be used within Next.js framework.*

Keywords: *React, model, code generator, Vite, components*

1. UVOD

Motivacija za razvoj aplikacije opisane u radu leži u rastućem broju tehnologija i radnih okvira za razvoj klijentskih aplikacija. Okviri često sadrže mnogo *boilerplate* koda što otežava razvojnim timovima da se fokusiraju na rešavanje domenskih problema. *Loosely-coupled* komponente omogućuju bolju ponovnu upotrebu koda i ubrzavaju razvoj novih funkcionalnosti [1]. Cilj opisanog alata je da korisnicima omogući vizualizaciju sistema, ili njegovih delova na klijentskoj strani, kako bi u ranoj fazi razvoja identifikovali potencijalne probleme u dizajnu.

Rad opisuje rešenje koje pokušava da prevaziđe ove probleme, i pruža integrisani generator koda koji definiše uniformnu strukturu komponenti i drugih resursa kroz celu aplikaciju. Vizualni editor pomaže korisnicima da identifikuju previše povezane komponente i bolje organizuju kod.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila prof. dr Gordana Milosavljević.

Sistem je razvijen kao veb aplikacija, kako bi bio dostupan širokom broju korisnika, bez potrebe za dodatnim zavisnostima. Funkcionalnost generisanja koda je omogućena pomoću obrađivača šablona, koji kombinuje šablone sa podacima kako bi se proizvele izlazne datoteke. Alat takođe uključuje izbor tehnologije za stilizovanje i integraciju *Redux*-a.

2. REACT

React [2] je prvi put predstavljen javnosti 2013. godine od strane kompanije *Meta* (tada *Facebook*). Iako je bio dostupan, značajan porast u popularnosti zabeležen je nekoliko godina kasnije. Važna prekretnica u razvoju *React*-a bila je uvođenje funkcijskih komponenti i *hook*-ova u verziji 16.8.

Jedan od ključnih koncepata u *React*-u je deklarativno programiranje, koje se primenjuje kroz organizaciju koda u komponente, *props* i *callback* funkcije. U *React* aplikacijama, komponente se posmatraju kao osnovne gradivne jedinice korisničkih interfejsa. *React* elementi, sa druge strane predstavljaju nepromenljive opise onoga što treba da se iscrta na stranici [1].

Kao relativno mlada tehnologija, *React* nema visok nivo standardizacije, i postoji mnogo načina da se implementira identična funkcionalnost. Ovo doprinosi problemu teškog održavanja koda, posebno u okruženjima gde se članovi tima menjaju i gde baza koda brzo raste. Ipak, postoje neke smernice, kao i noviji dizajn obrasci koji pomažu da kod postane lakši za održavanje i podstiču pravu namenu *React*-a kao tehnologije, a to su ponovno iskoristive komponente (*reusable components*).

Ponovna iskoristivost se postiže pomoću parametara komponente (*props*). U *React*-u postoji poseban *prop* koji se može vezati za bilo koju komponentu, a to je *children*. Taj *prop* je zapravo *JSX* koji se može propagirati u dečiju komponentu i prikazati na proizvoljnom nivou *JSX*-a dečije komponente. Posebno je koristan za podelu komponenti na logičke celine, jer nekad nije poželjno kroz parametre deliti informacije koje semantički nemaju smisla da postoje u komponenti.

Kada komponenta prosledi *props* drugoj komponenti, ona se naziva vlasnikom, bez obzira na njihovu relaciju roditelj-dete [1]. U nastavku teksta, vlasnik predstavlja komponentu koja prosleđuje *prop* dok je roditelj direktni čvor *DOM*-a. Principi koji se vezuju za paradigmu funkcionalnog programiranja, poput nepromenljivosti čistih funkcija i funkcija višeg reda pomažu u pisanju koda koji je lakši za održavanje i testiranje.

3. DIZAJN OBRASCI

3.1. Container presentational

Komponente u *React*-u tipično sadrže i logiku i prezentaciju. Pod logikom se podrazumeva sve što je nevezano za *UI*, poput *API* poziva, manipulacije podataka, obrađivača događaja i sl. Sa takvim komponentama, vrlo često minimalna promena zahteva da se logika proširi dodatnim uslovima i parametrima. Jedan način za kreiranje jasne granice između logike i prezentacije je poznat kao *container and presentational*. Ideja je da se složena komponenta, podeli na dve datoteke. Prva datoteka, koja predstavlja prezentacioni deo, ostaje sa istim imenom, dok se na novu datoteku dodaje sufiks *Container*.

Na ovaj način, ne samo da se preventivno sprečava prezasićenje komponente, već se i olakšava testiranje. Nova komponenta koja se koristi za prezentaciju samo prima i iscrtava podatke. Ovaj obrazac može da napravi veliku razliku kada je u pitanju održivost projekta [1]. Alat ovaj obrazac podržava, ali ga ne nameće korisniku. Korisnik ovaj obrazac može da primeni u alatu, tako što će definisati dve zasebne komponente, i potencijalno ih imenovati po konvenciji. Na ovaj način, prezentaciona komponenta može da se kreira vrlo brzo, ukoliko je markap za komponentu već definisan.

3.2. Komponente višeg reda

Primena koncepta funkcija višeg reda na *React* rezultuje komponentama višeg reda (*Higher-Order Components - HOCs*). *HOC* je funkcija koja prima komponentu kao ulaz i vraća proširenu komponentu kao izlaz. Ideja je proširiti ponašanje komponente bez narušavanja njene izolovane logike. Alat podržava kreiranje komponenti višeg reda, bez ograničenja.

3.3. React hooks

Namena *hooks* jeste da olakša način praćenja stanja unutar jedne komponente, kao i da pruži intuitivniji interfejs za baratanje životnim ciklusom komponente. *React* dolazi sa skupom ugrađenih *hook*-ova koji ovo podržavaju i to su *useState* i *useEffect*. Alat se fokusira na pravilnu primenu ova dva *hook*-a, jer su često dovoljni za interfejse proizvoljne složenosti.

Ugrađeni *React hook*-ovi olakšavaju razvoj novih komponenti, ali korisnik ima opciju da kreira sopstveni. Alat ne podržava direktno kreiranje *hook*-ova, jer se njihovo korišćenje u drugim komponentama razlikuje od uključivanja komponenti.

3.4. Function as a child

Function as a child je obrazac u *React*-u koji se koristi za dinamičko renderovanje sadržaja i omogućava komponentama da kontrolišu način na koji deca bivaju renderovana. U ovom obrascu, komponenta koristi funkciju kao dete, umesto *JSX*-a, što omogućava fleksibilnost u prenošenju podataka između komponenti. Funkcija obično prima parametre koji dolaze iz komponente vlasnika. Primer jeste komponenta koja upravlja stanjem, ili dobavlja podatke preko zahteva ka *API*-ju. Takva komponenta može da koristi ovaj obrazac, i omogući svojim potrošačima da odluče kako će ti podaci biti prikazani.

3.5. Redux

Redux je kontejner stanja za *JavaScript* aplikacije i služi kao šablon i biblioteka za upravljanje stanjima unutar aplikacije [3]. Ovaj mehanizam se realizuje upotrebom događaja, odnosno akcija. *Redux* omogućava upravljanje globalnim stanjima, što smanjuje potrebu za prosleđivanjem podataka među komponentama. Ovo dovodi do čitljivijeg koda, jasnije separacije logike i veće granularnosti komponenti.

Stanje aplikacije se čuva u globalnom objektu koji se naziva *store*. Kreator akcija je jednostavna funkcija koja vraća objekat sa opisom akcije i opciono propratnim sadržajem (*payload*). *Reducer* je funkcija koja prima trenutno stanje aplikacije i objekat akcije, odlučujući kako da ažurira trenutno stanje. *Reducer* funkcije predstavljaju primer čistih funkcija, što je važan aspekt funkcionalnog programiranja. *Reducer* se može posmatrati kao osluškivač koji obrađuje događaje u zavisnosti od tipa akcije. Standardni tok podataka u *Redux* arhitekturi obuhvata sledeće korake:

1. Registrovanje događaja. Može biti bilo koja akcija nad elementom korisničkog interfejsa, kao što je klik na dugme ili promena vrednosti u polju za unos.
2. *Dispatch* akcije. *Event handler* šalje odgovarajuću akciju ka *store*-u.
3. *Reducer* obrađuje akciju. *Reducer* prima akciju zajedno sa trenutnim stanjem i, u zavisnosti od tipa akcije, redefiniše postojeće stanje.
4. Osvežavanje interfejsa. Korisnički interfejs se ažurira novim stanjem.

Glavni problem koji se na ovaj način rešava je *prop-drilling*. *Prop-drilling* se javlja kada komponenta prosleđuje *props* detetu, koje ih zatim bez internog korišćenja prosleđuje svom detetu, što čini kod redundantnim i kompleksnim za održavanje. *Redux* i *useReducer* omogućavaju da svaka komponenta može direktno da pristupi globalnom stanju, bez potrebe za *prop-drilling*-om, čime se rešava problem komunikacije između tzv. *sibling* komponenti.

Za čitanje podataka iz *store*-a koriste se *selector* funkcije, koje se obično definišu u istom fajlu gde se definiše *slice*. *Selector* funkcije omogućavaju komponentama da pristupe delovima globalnog stanja na jednostavan način. *Hook*-ovi *useDispatch* i *useSelector* omogućavaju komponentama da šalju akcije i pristupaju stanju, čime se olakšava rad sa *Redux*-om u *React* aplikacijama.

4. KONFIGURACIJA PROJEKTA

4.1. Create React app

Do 2023. godine standardni način za kreiranje *React* aplikacije je bio *create-react-app (CRA)* alat [4]. Alat je vrlo jednostavan za korišćenje i omogućavao je korisniku da brzo započne sa razvojem svoje klijentske aplikacije. Njegova minimalna postavka i jednostavan proces kreiranja aplikacije je upravo razlog zašto je bio *de facto* standard. Iako nije pružao automatsku podršku za stvari koje su tipične za klijentske aplikacije, poput rutiranja i slanja zahteva ka *API*-ju, dolazio je sa konfigurisanim *webpack*-om [5] i *Babel*-om [6]. Otuda, iako na prvi pogled konfiguracija deluje jednostavno, dolazio je sa dosta zavisnosti, i malim prostorom za unapređenje.

Performanse su predstavljale najveći problem, jer su moderniji alati poput *Vite* [7] i radni okviri poput *Next.js* [8], dolazili sa malo više koraka u procesu kreiranja projekta, ali su pružali drastično bolje performanse [1].

4.2. Vite

Vite ima brže vreme pokretanja servera jer dolazi sa sopstvenim bundlerom i ne koristi *webpack* kao *CRA* [7]. Pruža direktnu podršku za rad sa *TypeScript*-om i bolju podršku za upravljanje zavisnostima (*dependency management*). Pored toga, dimenzije projekta kreiranog pomoću *Vite* zahtevaju 80% manje memorije od onih koji su kreirani sa *CRA* [7]. *Vite* prilikom pokretanja izvršava *dependency pre-building* i za to koristi *esbuild* koji je napisan u *Go* programskom jeziku i bundluje zavisnosti od deset do sto puta brže od *JavaScript* bundlera [7].

4.3. Stilizovanje

Od modernih aplikacija se očekuje mnogo kada je reč o stilovima i izgledu. Posledica toga je veliki broj opcija kada je reč o stilovima. Iako rezultantna aplikacija gore navedenih alata, podrazumevano podržava *CSS* (*Cascading Style Sheets*), vrlo često se timovi odlučuju za neku alternativu.

Neke od popularnijih varijanti u React aplikacijama su *TailwindCSS*, *SASS*, i *styled-components*. Prednosti koje se dobijaju nekim od pomenutih varijanti su lokalni stilovi, koji su lakši za održavanje od globalnih. *TailwindCSS* funkcioniše po principu atomskih pravila, gde svaka klasa predstavlja jedno pravilo. Svaka od ovih varijanti zahteva dodatnu konfiguraciju, što alat automatski podržava, tako da se izbor i podrazumevana konfiguracija dobijaju jednim klikom.

4.4. Zavisnosti

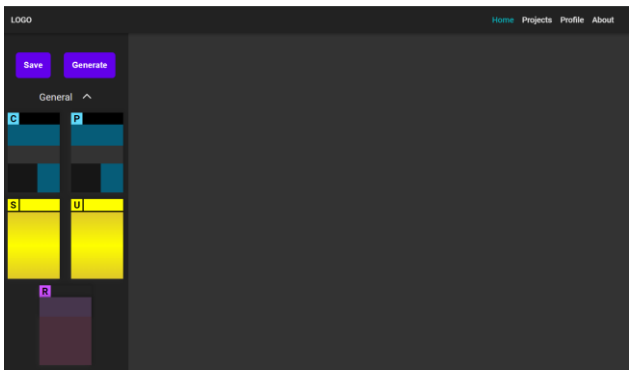
Datoteka koja sadrži meta-podatke o projektu, u *JavaScript* okruženjima je *package.json*. U njemu se nalaze informacije poput autora, verzije, licence, ali i zavisnostima projekta. Posao *package manager*-a je da te zavisnosti nabavi sa odgovarajućeg *npm* registra i smesti ih u specifičan folder *node_modules*.

Alat pored opcije za izbor *package manager*-a pruža korisniku opciju da odmah odabere neke zavisnosti koje planira da koristi u svom projektu. Slično vrlo popularnom *SpringInitializer*-u [9], u sekciji prikazanoj na slici 7. se korisniku nudi opcija pretrage. Sa *NPM* repozitorijuma se dodaju zavisnosti koje odgovaraju korisničkom unosu, i na taj način se smanjuje broj potrebnih koraka kako bi se nova zavisnost dodala u sam projekat.

5. OPIS REŠENJA

5.1. Editor

Grafički editor je implementiran kao *React* aplikacija pomoću *Vite* alata, sa funkcionalnostima poput zumiranja, pomeranja, prevlačenja elemenata, promene dimenzija i interaktivnih dijaloga. Nakon kreiranja novog projekta, korisniku se prikazuje interfejs sa mogućnostima za dodavanje elemenata na radnu površinu.
Klikom na element otvara se komponenta "fioka" (*Drawer*) koja omogućava detaljan opis elementa u zavisnosti od njegovog tipa. Na slici 5.1. prikazana je prazna radna površina editora.



Slika 5.1. Radna površina editora

Komponente, osnovne jedinice React aplikacija, korisnik može detaljno opisati kroz sedam koraka: definisanje imena i putanje, *props*-a, stanja, efekata, akcija, markapa (*JSX*) i vezama sa drugim komponentama. U okviru alata, *props*-i se unose uz validaciju da su jedinstveni i ispravno imenovani. Stanje komponente se definiše pomoću *useState hook*-a, i alat forsira konvencije imenovanja.
Efekti se definišu pomoću *useEffect hook*-a, omogućujući korisniku da postavi akcije koje se izvršavaju tokom životnog ciklusa komponente. Alat pomaže korisniku da automatski kreira i memoizuje *callback* funkcije pomoću *useCallback hook*-a. Akcije predstavljaju funkcije koje se pozivaju unutar *lifecycle hook*-ova ili kao obradivači događaja.
Na kraju, korisnik može definisati markap (*JSX*) koji je povratna vrednost komponente, što omogućava vizuelni prikaz na stranici. Takođe, alat pruža podršku za povezivanje komponente sa drugim delovima sistema, omogućavajući kreiranje složenih i modularnih aplikacija.
Pored komponenti, korisnik može da definiše i stranice, servise, *util* datoteke i *Redux slice*. Stranica predstavlja resurs koji je skoro identičan komponenti. Razlikuje se po tome što će se stranica automatski dodati u ruter *React SPA*, i nakon pokretanja generisanog koda, biti dostupna na definisanoj ruti.
Servisne datoteke sadrže funkcije koje izvršavaju *HTTP* zahtev pomoću *axios* biblioteke i vraćaju *HTTP odgovor*. Funkcija u servisnoj datoteci je klasična *JavaScript* funkcija koju definišu njen naziv i parametri. *Util* funkcije su takođe klasične *JavaScript* funkcije, ali im je namena drugačija od servisnih. Umesto da barataju *HTTP* zahtevima *util* funkcije su obično čiste funkcije, koje izvršavaju logiku koja nije vezana za pojedinačnu komponentu.
Poslednji element koji korisnik može da doda na radnu površinu predstavlja *Redux slice*. Biblioteke koje se automatski dodaju u projekat prilikom generisanja koda, ukoliko se koristi *Redux* slice, su *react-redux* i *reduxjs/toolkit*. Ove biblioteke olakšavaju implementaciju *Redux* toka podataka u aplikaciji koristeći *hook*-ove, što se uklapa u ostatak *React* ekosistema. *Redux Toolkit* je posebno značajan jer nudi unapređeni model za organizaciju koda i rešava problem prevelike količine *boilerplate* koda prilikom konfigurisanja *Redux*-a. Iako je *Redux* izuzetno moćan, jedna od njegovih mana jeste da korišćenje *react-redux* biblioteke ne pruža elegantan način za definisanje *slice*-a, tako da je logika centralizovana u jednom fajlu. *Hook*-ovi koji se koriste za

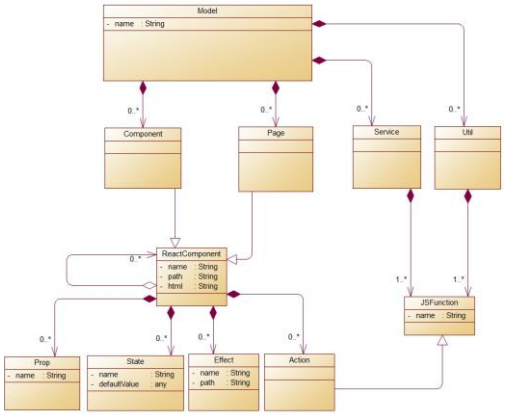
čitanje iz *store*-a i *dispatch*-ovanje akcija, se u komponentama uvoze direktno iz *react-redux* biblioteke.

5.2. Server

Server je implementiran korišćenjem *Express* [10] radnog okvira za *Node.js* [11], a kao baza podataka koristi se *MongoDB* [12]. *MongoDB* je vodeća *NoSQL* baza podataka koja čuva podatke u obliku *JSON* dokumenata, i omogućuje fleksibilnost i horizontalnu skalabilnost. *Express* omogućava postavljanje *RESTful API*-ja koji služi za komunikaciju sa klijentskom aplikacijom (editorom) i bazom podataka. Na ovaj način server obezbeđuje sve potrebne podatke editoru, kao i mogućnost skladištenja i ažuriranja modela projekata.

5.3. Generator koda

Generator koda je ključna komponenta koja koristi *template engine* za generisanje izlaznih datoteka na osnovu definisanih modela. Kao *template engine* koristi se *Nunjucks* [13], koji omogućava procesiranje unapred definisanih šablona sa podacima iz modela. *Nunjucks* pruža fleksibilnost u dizajniranju šablona, omogućavajući jednostavno uključivanje dinamičkih podataka u generisani kod. Generator prihvata šablone i podatke, obrađuje ih pomoću *Nunjucks engine*-a i proizvodi izlazne datoteke koje su deo gotovog projekta. Jedan od problema kod dizajniranja generatora koda jeste pronalaženje minimalnog skupa podataka takvog da se obezbedi funkcionalna softverska komponenta koja se može izmeniti, ali takođe direktno iskoristiti. Jedna od prednosti upotrebe *MongoDB* baze podataka je što se u klijentskoj aplikaciji proširenja mogu direktno dodati, i prikazati prilikom učitavanja modela, bez primene migracija. Na slici 5.2. je prikazan klasni dijagram za meta-model na osnovu kojeg radi generator.



Slika 5.2. Dijagram klasa za meta-model izlaznog projekta

6. ZAKLJUČAK

U radu je predstavljen sistem za modelovanje i generisanje *React* projekta. Sistem je sastavljen od serverske strane aplikacije implementirane u *Node*-u, klijentske strane aplikacije implementirane u *React*-y i *Nunjucks* obrađivača šablona za generisanje koda. Opisane su korišćene tehnike i tehnologije, kao i na koji način alat podržava različita okruženja i dizajn obrasce. Korisniku je omogućeno da na osnovu modela dobije izlazni projekat koji se može proširiti, ali i pokrenuti odmah po preuzimanju. Pravci budućeg razvoja:

- Mogućnost proširenja grafičkog editora dodavanjem novih elemenata kao što su *hook*-ovi, koje korisnik može koristiti u procesu modelovanja..
- Uvođenje funkcionalnosti za praćenje promena u generisanom kodu i automatsko ažuriranje formi elemenata u editoru, čime bi se omogućilo kontinuirano generisanje koda bez potrebe za ručnim unosom promena.
- Prikaz finalnog izgleda komponente u fazi modelovanja, kako bi se smanjilo vreme potrebno za izmenu.
- Uvođenje *LLM*-ova u okruženje koji bi mogli pomoći korisnicima da generišu određene servisne, util ili handler funkcije na osnovu slobodnog teksta, uz mogućnost pregleda i odlučivanja o zadržavanju generisanog koda.

7. LITERATURA

[1] Carlos Santana Roldán. React 18 Design Patterns and Best Practices: Design, build, and deploy production-ready web applications with React by leveraging industry-best practices, 4th Edition.Packt Publishing, 2023. ISBN 978-1803233109

[2] React, https://reactjs.org/, na mreži, pristup: septembar 2024

[3] Redux, https://redux.js.org/ , na mreži, pristup: septembar 2024

[4] Create React app, https://reactjs.org/docs/create-a-new-react-app.html, na mreži, pristup: septembar 2024

[5] Webpack, https://webpack.js.org/, na mreži, pristup: septembar 2024

[6] Babel, https://babeljs.io/, na mreži, pristup: septembar 2024

[7] Vite, https://vitejs.dev/, na mreži, pristup: septembar 2024

[8] Next.js, https://nextjs.org/, na mreži, pristup: septembar 2024

[9] SpringInitializr, https://start.spring.io/, na mreži, pristup: septembar 2024

[10] Express, https://expressjs.com/, na mreži, pristup: septembar 2024

[11] Node.js, https://nodejs.org/en, na mreži, pristup: septembar 2024

[12] MongoDB, https://www.mongodb.com/, na mreži, pristup: septembar 2024

[13] Nunjucks, https://mozilla.github.io/nunjucks/, na mreži, pristup: septembar 2024

Kratka biografija:

Filip Volarić rođen je 25. septembra 1999. godine u Rumi. Pohađao je OŠ "Veljko Dugošević" koju je završio 2014. godine kao nosilac diplome "Vuk Karadžić". Iste godine upisuje Mitrovačku gimnaziju, smer za obdarene učenike u računarskoj gimnaziji. Gimnaziju je završio 2018. godine. Iste godine upisuje Fakultet tehničkih nauka, smer Softversko inženjerstvo i informacione tehnologije. Diplomirao je 2022. godine i stekao zvanje diplomirani inženjer elektrotehnike i računarstva.

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.9
DOI: https://doi.org/10.24867/30BE06Njegic

CLOUDCAST – ПЕРСОНАЛИЗОВАНЕ НОТИФИКАЦИЈЕ ЗА ПРОГНОЗУ ВРЕМЕНА ЗАСНОВАНЕ НА *AWS SERVERLESS* ИНФРАСТРУКТУРИ

CloudCast – PERSONALIZED WEATHER FORECAST NOTIFICATIONS BASED ON AWS SERVERLESS INFRASTRACTURE

Ђорђе Његић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХИНЧКО И РАЧУНАРСКО ИНЖЕЊЕРСТВО

Кратак садржај – *У овом раду описан је CloudCast, сервис заснован на претплати који пружа персонализоване временске нотификације путем е-поште или СМС-а. Корисници конфигуришу свој налог на основу чега добијају персонализоване нотификације на дневном нивоу са информацијама о тренутним временским условима и прогнозама. Сервис је у потпуности изграђен на AWS инфраструктури користећи serverless архитектуру.*

Кључне речи: *Претплатничке услуге, Serverless архитектура, AWS инфраструктура*

Abstract – *This paper describes CloudCast, a subscription-based service that provides personalized weather notifications via email or SMS. Users configure their accounts to receive daily personalized notifications with information about current weather conditions and forecasts. The service is entirely built on AWS infrastructure using a serverless architecture.*

Keywords: *Subscription Services, Serverless Architecture, AWS Infrastructure*

1. УВОД

Данас постоји велики број сервиса који пружају информације о временским условима. Ови сервиси су углавном доступни путем мобилних и веб апликација, омогућавајући корисницима да брзо и лако добију прогнозу времена. Међутим, већина ових сервиса пружа само основне податке и то на кориснички захтев, што може бити непрактично за оне који су у покрету или имају ограничен приступ интернету.

Оно што разликује сервис *CloudCast* који ће бити представљен у овом раду јесте начин информисања корисника путем претплате и дневних нотификација. Корисници овог сервиса добијају персонализоване нотификације које су прилагођене њиховим потребама у погледу времена када их добијају, садржаја нотификације, као и медијума путем којег се обавештавају (е-маил или СМС).

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Мирослав Зарић, ред. проф.

2. КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ

2.1. Terraform

Terraform [1] је алат за инфраструктурно кодирање (IaC) који омогућава дефинисање и управљање инфраструктуром кроз декларативне конфигурационе фајлове. Користи се за аутоматизацију креирања и управљања ресурсима у cloud окружењима као што су AWS, Azure, Google Cloud и други. За разликзу од IaC алата које нуде сами cloud-provider-и, Terraform је независан од конкретног окружења и омогућава дефинисање инфраструктуре јединственом синтаксом за све. Такође омогућава верзионисање инфраструктуре, што олакшава праћење промена и враћање на претходне верзије уколико је потребно.

2.2. AWS (Amazon Web Services)

AWS [2] је водећи глобални понуђач услуга у облаку основан 2006. гоидине. Нуди широк спектар услуга у оквиру више од 200 различитих сервиса. Неке од функционалности које овај сервис нуди су креирање и управљање виртуелним серверима, складиштење и обрада података, базе података, мониторинг, аналитика, машинско учење и друго. AWS омогућава организацијама да брзо и ефикасно развијају, имплементирају и скалирају апликације без потребе за управљањем инфраструктуром.

2.2.1 AWS API Gateway

AWS API Gateway [3] је сервис који омогућава креирање, одржавање и заштиту API-ја. Омогућава дефинисање различитих типова API-ја, пружајући могућности аутентификације, ауторизације, ограничавања протока и праћења перформанси, скалирање у складу са количином саобраћаја, кеширање, и друго. Интеграција са AWS Lambda функцијама омогућава *serverless* архитектуру, где се пословна логика најчешће извршава у Lambda функцијама као одговор на захтеве који пристигну на API Gateway. Ово представља уобичајен шаблон у дефинисању *serverless* архитектура.

2.2.2 AWS Cognito

AWS Cognito [4] је сервис за управљање идентитетима и аутентификацијом корисника који омогућава једноставну интеграцију са другим AWS сервисима попут API Gateway-а. Путем овог сервиса корисници могу да креирају налоге, пријављују се, ресетују лозинке и управљају својим корисничким профилима.

2.2.3 AWS Lambda

AWS Lambda [5] је сервис који представља основ serverless архитектуре на AWS-у. Омогућава извршавање кода као одговор на догађаје без потребе за управљањем инстанцама сервера. Lambda функције се могу покренути као одговор на различите догађаје, као што су промене у S3 bucket-има, упити ка API Gateway-у, или поруке у SNS темама. Lambda омогућава аутоматско скалирање и плаћање само за утрошено време извршавања кода, што може значајно смањити трошкове у поређењу са инфраструктуром базираном на серверима. Интеграција са другим AWS сервисима омогућава изградњу комплексних апликација без потребе за управљањем инфраструктуром.

2.2.4 AWS DynamoDB

AWS DynamoDB [6] представља serverless NoSQL базу података којом у потпуности управља AWS, захваљујући чему гарантује брзе и предвидиве перформансе са аутоматским скалирањем. Такође, гарантује високу доступност и отпорност на грешке, што је кључно за апликације које увек морају бити доступне. Подржава сложене упите и индексирање, што олакшава брзо претраживање и филтрирање података. Представља један од стандардних компоненти serverless шаблона на AWS-у.

2.2.5 AWS S3 (Simple Storage Service)

AWS S3 [7] је сервис за складиштење објеката који пружа скалабилно, трајно и сигурно складиштење података. Омогућава једноставно управљање складиштењем података, са могућностима верзионисања, енкрипције и контроле приступа. Често се користи за складиштење статичких фајлова, као што су слике, видео записи, резервне копије и слично. Интеграција са другим AWS сервисима омогућава аутоматизацију процеса, као што је покретање Lambda функција као одговор на специфициране догађаје у неком S3 bucket-у.

2.2.6 AWS SES (Simple Email Service)

AWS SES [8] је сервис који омогућава слање велики број различитих типова мејлова. Омогућава једноставну интеграцију са другим AWS сервисима, као и аутоматизацију процеса слања мејлова, могућност дефинисања шаблона за мејлове и друго. SES се уобичајено користи за различите верификације, слање нотификација, потврда, *newsletter*-а и других врста email комуникације.

2.2.7 AWS SNS (Simple Notification Service)

AWS SNS [9] је сервис за слање порука и нотификација у реалном времену. Омогућава слање порука путем различитих медијума, као што су SMS, email и HTTP/HTTPS комуникација, што пружа флексибилност у начину испоруке обавештења.

Често се користи за слање нотификација о догађајима, алармима, и другим врстама порука, и омогућава једноставну интеграцију са другим AWS сервисима за аутоматизацију процеса.

2.2.8 Amazon EventBridge

Amazon EventBridge [10] је сервис за управљање догађајима који омогућава међусобно увезивање апликација и сервиса, заказивање извршавања одређених акција и слично. EventBridge омогућава креирање правила за филтрирање и усмеравање догађаја, што олакшава изградњу реактивних и скалабилних апликација, и често се користи за оркестрацију догађаја између различитих сервиса и апликација

2.2.9 AWS CloudWatch

AWS CloudWatch [11] је сервис за праћење, надгледање и управљање ресурсима и апликацијама на AWS-у. Омогућава конфигурацију, прикупљање и праћење различитих врста логова, метрика и догађаја из AWS сервиса. Такође, омогућава и креирање аларма и визуелних приказа за праћење перформанси и стања апликације, што олакшава идентификовање и решавање евентуалних проблема. Нуди и могућност аутоматизованих одговора на одређене догађаје, што на пример може бити покретање Lambda функција као одговор на одређене метрике. CloudWatch представља основни сервис за надгледање употребе ресурса и перформанси апликација на AWS-у.

2.2.10 AWS IAM (Identity and Access Management)

AWS IAM [12] је сервис за контролу приступа AWS ресурсима. Омогућава дефинисање полиса и њихово додељивање ролама, на основу којих се одређује који сервиси и под којим условима имају право приступа другим ресурсима и сервисима у оквиру AWS-а. Омогућава грануларну контролу приступа, а користи се и за управљање корисничким дозволама, креирање и управљање корисничким налозима и групама, и генерално обезбеђивање сигурног приступа ресурсима

2.3. Node.js

Node.js [13] је радно окружење за извршавање *JavaScript* кода на серверској страни, односно ван веб прегледача. Базиран је на V8 JavaScript engine-у који користи Google Chrome претраживач. Захваљујући својим асинхроним и догађајима управљаним (*event-driven*) карактеристикама, погодан је за изградњу апликација које захтевају паралелно извршавања или real-time обраду података. Још једна важна карактеристика је lightweight природа Node.js-а, која га чини идеалним избором за serverless архитектуру и коришћење у AWS Lambda функцијама.

2.4. Angular

Angular [14] је компонентно оријентисан радни оквир за развој скалабилних веб апликација. Комплетно је написан у TypeScript-у. Користи се за изградњу клијентског дела апликација користећи HTML [26] и TypeScript. Омогућава креирање динамичких, једностраних апликација (Single Page Applications - SPA) које су брзе, интерактивне и лаке за одржавање.

3. СПЕЦИФИКАЦИЈА ПРОЈЕКТА

3.1. Преглед пројекта

CloudCast представља сервис базиран на принципу претплате који корисницима омогућава да добијају персонализована обавештења о временским прилика-кама.

Након што се успешно региструје, а самим тим и претплати на услуге сервиса, корисник ће добијати дневна обавештења у складу са конфигурацијом налога. Осим тога, има могућност пријаве на сам сервис путем свог корисничког налога у оквиру ког може да промени конфигурацију претплате, евентуално се одјави са претплате, као и да прегледа тренутне временске услове и прогнозу за наредних пет дана за град који је специфицирао.

Поред крајњих корисника система, препозната је и улога администратора. Администратор на недељном нивоу путем мејла добија дијаграме који представљају информације извучене из тренутно активних претплата.

Ови дијаграми представљају тренутне односе различитих типова претплата, градова које су корисници специфицирали у оквиру претплате и слично. Такође, све ове информације се чувају и у оквиру система како би администратор могао да прегледа историјске податке и анализира тренд промена у подацима

3.2. Функционални захтеви

Систем препознајте три врсте корисника, и следеће функционалне захтеве у оквиру сваке категорије корисника:

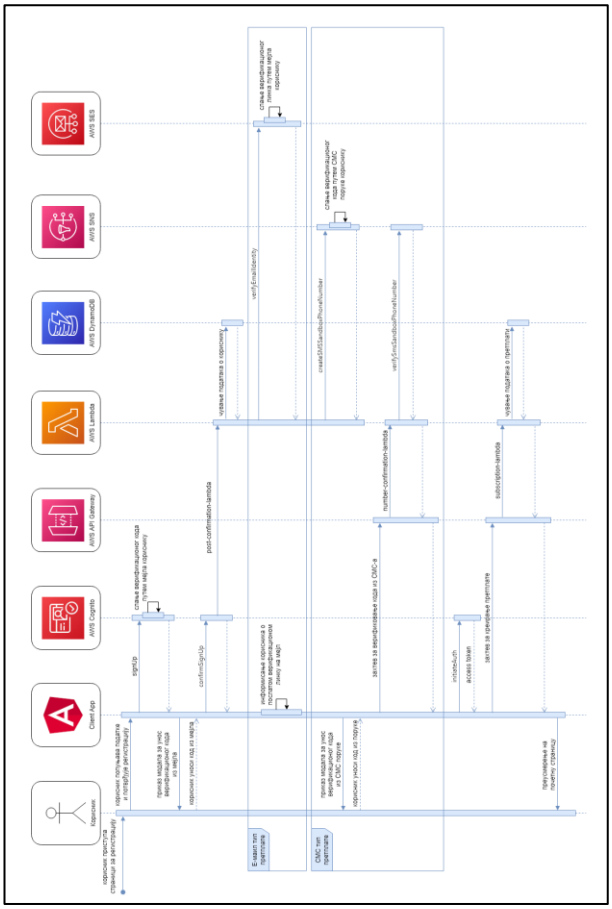
- Неулоговани корисник
  - Пријава на систем
  - Регистрација
- Регуларни корисник
  - Добијање дневних обавештења о временској прогнози у складу са конфигурацијом претплате
  - Промена статуса претплате
  - Промена конфигурације претплате
  - Преглед тренутних временских услова и прогнозе за наредних 5 дана
- Администратор
  - Добијање недељних извештаја о броју активних претплата као и дијаграма који приказују расподеле претплата по различитим конфигурационим параметрима
  - Преглед историјских података

3.3. Архитектура система

Архитектура сервиса CloudCast заснована је на serverless приступу и дефинисана је као инфраструктура као код (IaC) користећи Terraform. Овај приступ омогућава аутоматско креирање и управљање ресурсима у AWS окружењу. Клијентска апликација је развијена помоћу Angular-a, пословна логика апликације у Node.js-у.

3.4. Дијаграми

Неке од комплекснијих функционалности у систему представљени су дијаграмима секвенци ради лакшег разумевања тока догађаја. У наставку представљен је дијаграм секвенци за процес регистрације (слика 1).

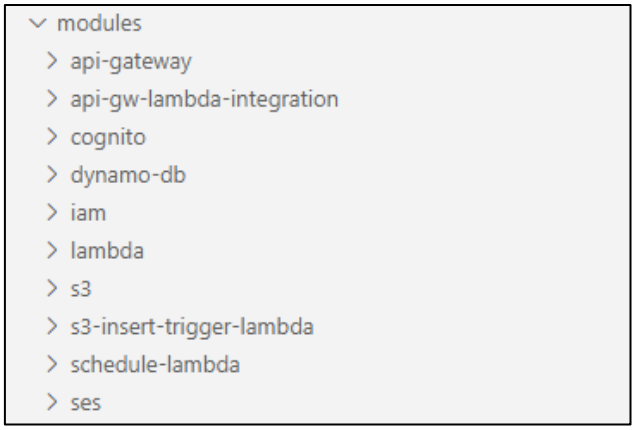


Слика 1. *Дијаграм секвенци за функционалност регистрације*

4. ИМПЛЕМЕНТАЦИЈА СИСТЕМА

4.1. Инфраструктура

Комплетна инфраструктура налази се на AWS-у, а за подизање инфраструктуре коришћен је Terraform IaC алат. Код инфраструктуре организован је у модуле који служе за груписање повезаних ресурса и омогућују једноставан начин за њихову вишеструку употребу и одржавање.



Слика 2. *Преглед свих модула у пројекту*

Сваки модул састоји се из три фајла: *main.tf* у оквиру ког се ддекларишу terraform ресурси, а на основу којих се креирају одговарајући ресурси у оквиру инфраструктуре, *outputs.tf* фајл у оквиру ког се декларишу output terraform параметри и *variables.tf* у оквиру ког се декларишу variable terraform параметри, помоћу којих је могуће конфигурисати одређене делове модула на месту њихове декларације.

4.2. Бизнис логика апликације

Целокупна бизнис логика сервиса налази је организована у осам lambda функција.

4.3. Клијентска апликација

За имплементацију клијентске апликације коришћен је радни оквир *Angular*. Клијентска апликација састоји се од 6 компоненти и *servis* директоријума. Од ових 6 компоненти, при чему прве 4 у наставку представљају читаве странице, а уједно и све странице које постоје у оквиру клијентске апликације, док преостале две представљају модалне прозоре који се појављују у оквиру тих страница

5. ЗАКЉУЧАК

CloudCast представља сервис за слање обавештења о временским приликама за одређени град. Идеја за овај рад настала је услед недостатка сличних решења на тржишту. Предност овог сервиса у односу на већ постојећа, слична решења је претплатнички механизам, као и могућност примања обавештења путем СМС порука. У претходним поглављима поменути су најпопуларнија слична решења на тржишту, наведене су технологије коришћене у имплементацији сервиса, спецификација система укључујући и UML дијаграме, детаљи имплементације и на крају демонстрација рада сервиса.

Када је реч о даљем развоју функционалности сервиса, један од могућих праваца могао би бити проширење информација о временским приликама које се шаљу кориснику. Такође, потенцијална нова функционалност је неки вид аларма који би послао упозорење свим преплатницима одређеног града уколико је велика временска непогода најављена за ту регију. Са друге стране, што се тиче инфраструктурног дела система и техничке имплементације, потенцијална унапређења могла би се тражити у проналажењу serverless решења и за клијентску апликацију, како би била у складу са остатком инфраструктуре.

6. ЛИТЕРАТУРА

[1] What is Terraform | Terraform | HashiCorp Developer. [Online], Приступљено датума: 22.8.2024. https://developer.hashicorp.com/terraform/intro

[2] About AWS. [Online], Приступљено датума: 22.8.2024. https://aws.amazon.com/about-aws/

[3] Amazon API Gateway. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

[4] Amazon Cognito. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html

[5] AWS Lambda. [Online], Приступљено датума: 23.8.2024. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[6] Amazon DynamoDB. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

[7] Amazon Simple Storage Service. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[8] Amazon Simple Email Service. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/ses/latest/dg/Welcome.html

[9] Amazon Simple Notification Service. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/sns/latest/dg/welcome.html

[10] Amazon EventBridge. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html

[11] Amazon CloudWatch. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[12] AWS Identity and Access Management. [Online], Приступљено датума: 22.8.2024. https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

[13] Node.js — About Node.js. [Online], Приступљено датума: 22.8.2024. https://nodejs.org/en/about

[14] Angular - What is Angular? [Online], Приступљено датума: 22.8.2024. https://v17.angular.io/guide/what-is-angular

Кратка биографија:

Ђорђе Његић рођен је 5. септембра 1999. године у Суботици где је стекао основно и средње образовање. Школске 2018/2019 године уписује се на Факултет техничких наука Универзитета у Новом Саду, смер Софтверско инжењерство и информационе технологије. Основне академске студије завршио је 2022. године и исте године уписује мастер академске студије на истом студијском програму. Контакт: djnjegic@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/30BE07Milijevic

PREDIKCIJA DOBITNIKA FILMSKE NAGRADE OSKAR UPOTREBOM MAŠINSKOG UČENJA

USING MACHINE LEARNING TO PREDICT OSCAR WINNERS

Jelena Milijević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *У раду је вршена предикција добитника награде Оскар за најбољи филм и за глумачко остварење. Изтраживање ове теме проистиче из њеног значаја за filmsku industriju. Korišćeni algoritmi su: Logistička Regresija, SVM, Random Forest, Bagging, XGBoost і Неуронска Мрежа. За евалуацију модела коришћена је 10-унакрсна валидација. У првој предикцији најбоље се показао Random Forest са тачношћу 91,59%, а у другој XGBoost са 84,39%.*

Ključne reči: *Mašinsko učenje, Logistička regresija, SVM, Random Forest, Bagging, XGBoost, Neuronska mreža*

Abstract – *The study focused on predicting the winners of the Oscar award for Best Picture and for acting achievements. This research is motivated by its significance to the film industry. The algorithms used include: Logistic Regression, SVM, Random Forest, Bagging, XGBoost, and Neural Networks. Model evaluation was conducted using 10-fold cross- validation. In the first prediction, Random Forest demonstrated the best performance with an accuracy of 91.59%, while in the second prediction, XGBoost achieved an accuracy of 84.39%.*

Keywords: *Machine Learning, Logistic Regression, SVM, Random Forest, Bagging, XGBoost, Neural Network*

1. UVOD

U dinamičnom svetu filmske industrije, Oskar nagrada predstavlja vrhunac priznanja za izuzetna dostignuća u kinematografiji. Efikasno predviđanje pobednika Oskara može imati značajan uticaj na filmsku industriju, unapređujući strategije produkcije, marketinga i distribucije, te zadovoljavajući očekivanja publike i stvaralaca.

Ovaj rad istražuje mogućnost primene metodologije mašinskog učenja u predviđanju pobednika Oskara u kategoriji za najbolji film, kao i za glumačko ostvarenje. Glavni izazov je kompleksnost podataka, koji obuhvataju atribute poput žanra, kritika publike, režije, i prethodnih nagrada. Raznovrsnost podataka zahteva pažljivu analizu i odabir relevantnih atributa za uspešno modelovanje.

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

U ovom radu je rađeno prikupljanje podataka, kao i eksplorativna analiza tih podataka. Analizirajući obimne podatke o filmovima, uključujući žanr, režiju, kao i prethodne nominacije i osvojene nagrade, primenjeni su različiti algoritmi: Logistička regresija, Random Forest, XGBoost, Veštačka neuronska mreža, Support Vector Machine i Random Forest sa Bagging-om. Ovaj ceo postupak je urađen i za predviđanje dobitnika nagrede za glumu. Takođe su primenjeni isti ovi algoritmi na podatke kao što su godina rođenja glumca, starost, prethodne nominacije i osvajanja ove i drugih nagrada.

Kako bi se modeli evaluirali korišćena je 10-ostruka unakrsna validacija. Takođe za svaki model je računata tačnost, preciznost, odziv, F1 mera i matrica konfuzije.

2. PREGLED STANJA U OBLASTI

Jedan od prvih radova vezanih za predikciju uspeha filma bio je rad iz 2000. godine. Ovaj rad [1] nudi detaljnu analizu i metodologiju za predikciju filmskih zarada, ističući važnost različitih faktora pre i nakon izlaska filma, uključujući i uticaj Oskar nominacija. Autori su koristili regresione modele za predikciju logaritmovanih vrednosti domaćih zarada filmova.

Iain Pardoe [2] se bavio statističkom analizom predikcije dobitnika Oskara u četiri glavne kategorije (najbolji film, režija, glumac i glumica u glavnoj ulozi) od 1938. do 2004. godine. Za predikciju je korišćen Multinomial Logit Model (MNL).

U sledećem radu Iain Pardoe i Dean K. Simonton [3] se fokusiraju na korišćenje modela diskretnog izbora za predikciju pobednika Oskara u četiri glavne kategorije. Pored MNL modela koriste se i ML(Mixed Logit Model) modeli.

Rad [4] predstavlja jedan od prvih pokušaja korišćenja mašinskog učenja za predikciju dobitnika Oskara. Predikcija je takođe obuhvatala četiri glavne kategorije. Metodologije koje su korišćene za predikciju su: SVM, Logistic Regression, Random Forest, Gaussian Naive Bayes, Multinomial Naive Bayes.

Rad [5] koristi mašinsko učenje za predviđanje popularnosti filmova. Primenjene metodologije u ovom radu su: Logistic Regression, Simple Logistics, J48, Naive Bayes, Multilayer Perceptron Neural Network, PART. Metode su evaluirane koristeći 10-ostruku unakrsnu validaciju.

U radu [6] se vrši evaluacija performansi klasifikacionih tehnika mašinskog učenja za predviđanje uspešnosti filma.

Metodologije primenjene u ovom radu su: Logistic Regression, Support Vector Machine, Random Forest, Gaussian Naive Bayes, AdaBoost, Stochastic Gradient Descent, Multilayer Perceptron Neural Network. Skup podataka je sadržao 755 filmova između 2012. i 2015.

U radu [7] se vrši predviđanje pobednika nagrade Oskar za najbolji film na osnovu odabranih karakteristika. Korišćena metodologija je Binary Logistic Regression.

U radu [8] vršena je predikcija ekonomskog uspeha filma korišćenjem metodologija: Random Forest, SVM i Multilayer Perceptron Neural Network. Autori su koristili skup podataka koji obuhvata 3167 filmova iz perioda između 1980. i 2019. godine. Autori su koristili 10-ostruku unakrsnu validaciju za sve eksperimente, kao i metrike poput preciznosti i F1 ocene.

3. TEORIJSKI POJMOVI I DEFINICIJE

3.1. Logistička Regresija

Logistička regresija je nadgledani algoritam mašinskog učenja koji ispunjava zadatke binarne klasifikacije predviđanjem ishoda. Model daje binarni ishod ograničen na dva moguća ishoda: da/ne, 0/1 ili tačno/netačno.

3.2. Support Vector Machine (SVM)

SVM je jedan od najpopularnijih algoritama za nadgledano učenje, koje se koristi za probleme klasifikacije i regresije. Osnovna ideja SVM-a je pronaći hiper-ravan (ili više njih) koji najbolje razdvaja podatke u različite klase.

3.3. Random Forest i Bagging

Random Forest je popularan i snažan ansambl algoritam za klasifikaciju, regresiju, i druge zadatke, koji koristi više odluka stabala za donošenje konačne odluke.

Ansambl metoda kombinuje predikcije više modela kako bi se poboljšale performanse u poređenju sa pojedinačnim modelima. Random Forest koristi tehniku poznatu kao bagging (Bootstrap Aggregating), gde više odluka stabala radi zajedno. Bagging je metoda ansambl tehnike koja kombinuje više modela kako bi smanjila varijansu i poboljšala preciznost.

3.4. XGBoost

XGBoost je optimizovana implementacija gradient boosting algoritma, dizajnirana da bude izuzetno efikasna, fleksibilna i prenosiva. Gradient Boosting je popularan algoritam za boosting. Kod gradient boostinga, svaki prediktor ispravlja grešku svog prethodnika.

XGBoost je implementacija Gradient Boosted stabala odluke. U ovom algoritmu, stabla odluke se kreiraju u sekvencijalnom obliku. Težine igraju važnu ulogu u XGBoost-u. One se dodeljuju svim nezavisnim promenljivama koje se zatim ubacuju u stablo odluke koje predviđa rezultate.

3.5. Neuronska mreža

Neuronske mreže su osnovni deo mnogih savremenih metoda mašinskog učenja i dubokog učenja. Osnovni gradivni blokovi neuronskih mreža su neuroni. Svaki neuron prima ulaze, obrađuje ih i šalje rezultat napolje. Takođe bitna stavka kod neuronskih mreža su aktivacione funkcije. To je funkcija koja određuje izlaz neurona na osnovu njegovog ulaza.

3.6. One-hot-encoding i TF-IDF

One-Hot Encoding je tehnika za pretvaranje kategorijskih (diskretnih) varijabli u numerički format. Svaka kategorija se pretvara u binarni vektor sa samo jednim aktivnim (1) bitom, dok su svi ostali bitovi nulti (0).

TF-IDF (Term Frequency Inverse Document Frequency) je tehnika za pretvaranje tekstualnih podataka u numerički format, koja meri značaj svake reči u dokumentu u odnosu na sve druge dokumente u korpusu.

4. METODOLOGIJA

Struktura sistema korišćenog za predikciju dobitnika Oskara za najbolji film je identična za predviđanje dobitnika Oskara za glumačko ostvarenje. Sam sistem započinje prikupljanjem podataka sa različitih izvora i sajtova, čime se formira osnovni skup podataka za analizu.

Nakon što su podaci prikupljeni, sprovedena je eksplorativna analiza podataka (EDA) kako bi se identifikovali ključni obrasci, odnosi i potencijalni problemi u podacima, uključujući nedostajuće vrednosti. Kako bi se smanjio uticaj ekstremnih vrednosti i postigla bolja normalizacija podataka, primenjene su transformacije poput korenovanja i logaritmovanja na odabranim numeričkim atributima. Takođe, za pretvaranje tekstualnih podataka u numeričke vrednosti korišćene su metode One-Hot Encoding i TF-IDF, što je omogućilo efikasniju obradu i analizu podataka.

Nakon eksplorativne analize podataka izvršena je podela skupa podataka na trening i test skup u odnosu 80:20. Budući da se radi o nebalansiranom skupu podataka, gde postoji značajno veći broj nominovanih u odnosu na one koji su osvojili Oskara, posebna pažnja posvećena je stratifikaciji tokom podele.

Trening modela je sproveden koristeći šest različitih algoritama: Logistička Regresija, SVM, Random Forest, Random Forest uz Bagging, XGBoost i neuronska mreža. Ulaz u svaki model je obrađen skup podataka sa numeričkim vrednostima, a izlaz je klasifikacija koja označava da li je osvojen Oskar ili ne.

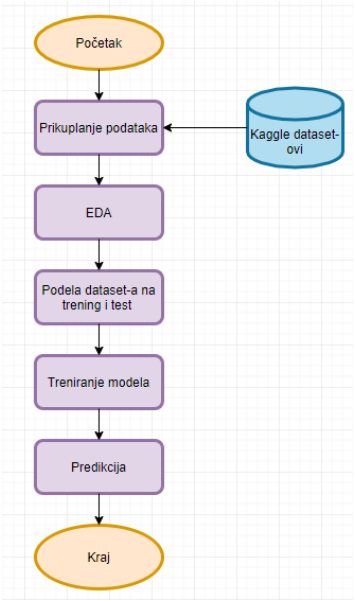
Svaki od ovih modela je obučen na trening skupu, a zatim evaluiran koristeći 10- ostruku unakrsnu validaciju, što je omogućilo robustan uvid u performanse modela. Za svaki model izračunate su ključne metrike, uključujući tačnost, preciznost, odziv, F1 meru i matricu konfuzije.

5. EKSPERIMENTI

U ovom radu su všene dve predikcije, jedna za predikciju dobitnika Oskara za glumu, a druga za glumačko ostvarenje. Korišćena su dva skupa podataka.

5.1. Skupovi podataka

Za prvi skup podataka izabrani su filmovi koji su bili nominovani od 1961. do 2021. godine. Ukupno su preuzeta tri seta podataka o nominovanim filmovima, koji su kasnije spojeni u jedan set. Set [9] je poslužio kao osnova za formiranje konačnog skupa podataka zbog raznolikosti atributa kojima raspolaže.



Slika 1. *Arhitentura sistema*

Nedostajući podaci za ceremoniju iz 2021. godine pronađeni su u okviru drugog seta podataka [9]. Spajanjem dva seta dobijen je novi set podataka od ukupno 342 uzorka. Iz trećeg preuzetog skupa podataka [10], upotrebljena je duration kolona koja sadrži informaciju o trajanju filma Jedini vid modifikacije ovog skupa podataka je preimenovanje labele original_title u Film kako bi se moglo uspešno izvršiti spajanje sa prethodnim setom po nazivu filma.
Drugi skup podataka sadrži sve nominovane glumce i glumice u glavnim i sporednim ulogoma periodu od 1961. do 2021. godine. Ukupno su preuzeta dva seta podataka sa podacima o nominovanim glumcima i glumicama, koji su kasnije spojeni u jedan set. Set [9] je poslužio kao osnova za formiranje konačnog skupa podataka zbog raznolikosti atributa kojima raspolaže. Nedostajući podaci za ceremoniju iz 2021. godine pronađeni su u okviru drugog seta podataka [9]. Spajanjem dva seta dobijen je novi set podataka od ukupno 1185 uzorka.

5.2. Eksperiment 1 – Logistička Regresija

Opisana su dva modela jedan koji se koristi za predikciju dobitnika Oskara za najbolji film a drugi za glumaćko ostvarenje.
Vrednosti hiper-parametara za prvi model su: max_iter=2000, random_state=26, C=0.1, penalty='l2', solver='liblinear', class_weight='balanced' fit_intercept= False.
Vrednosti hiper-parametara za prvi model su: max_iter=8000, random_state=26, C=100, penalty='l2', solver=lbfgs, class_weight='balanced' fit_intercept= False.

5.3. Eksperiment 2 – SVM

Za optimizaciju parametara oba modela korišćen je Grid Search. Za prvi model najbolje su se pokazali parametri C=1, gamma=0.1 i kernel='sigmoid. Dok za drugi model najbolje su se pokazali parametri C=1, gamma=0.01 i kernel='sigmoid'.

5.4. Eksperiment 3 – Random Forest

Hiper-parametri za oba modela su se podešavala ručno.
Vrednosti hiper-parametara za prvi model su: n_estimators=100, criterion='gini', max_depth=1, bootstrap=False, max_features=None, min_samples_leaf= 1, random_state=42, min_samples_split=2.
Vrednosti hiper-parametara za drugi model su: n_estimators=300, criterion='entropy', max_depth=4, bootstrap=False, max_features=0.6, min_samples_leaf= 2, random_state=42, min_samples_split=4, class_weight= 'balanced'.

5.5. Eksperiment 4 – Random Forest sa Bagging-om

Hiper-parametri za oba modela su se podešavala ručno.
Za prvi model koristio se RandomForestClassifier sa hiper-parametrima: n_estimators=10, criterion='gini', max_depth=1, min_samples_split=2, min_samples_leaf= 1, max_features=None, bootstrap=True. Zatim je RandomForestClassifier korišćen kao bazni model u BaggingClassifier-u sa 10 instanci i random_state =42.
Drugi model koristi BalancedRandomForestClassifier sa hiper-parametrima: n_estimators=200, criterion='entropy', max_depth=1, min_samples_split=2, min_samples_leaf= 1, max_features=0.6, bootstrap=True, sampling_strategy= 'auto'. Zatim je taj BalancedRandomForestClassifier korišćen kao bazni model u BaggingClassifier-u sa 10 instanci i random_state =42.

5.6. XGBoost

Hiper-parametri za oba modela su se podešavala ručno.
Vrednosti hiper-parametara za prvi model su: n_estimators=100, reg_alpha=0, random_state=42, learning_rate=0.1, random_state=42, max_depth=2, min_child_weight=1, gamma=0.1, reg_lambda=1, colsample_bytree=0.8, subsample=0.8.
Vrednosti hiper-parametara za drugi model su: n_estimators=100, reg_alpha=1, random_state=42, learning_rate=0.1, random_state=42, max_depth=4, min_child_weight=2, gamma=0.1, reg_lambda=2, colsample_bytree=0.8, subsample=0.8.

5.6. Neuronska mreža

Hiper-parametri za oba modela su se podešavala ručno
Prva model se satoji iz tri sloja. Prvi sloj ima 128 neurona, drugi 64 i poslednji 1. U prva dva sloja kao aktivaciona funkcija navedena je ReLU, a u poslednjem Sigmoid aktivaciona funkcija.
Drugi model je, s druge strane, složeniji, jer se obučava na većem skupu podataka. Sadrži četiri ključna sloja, a između svakog para slojeva umetnuti su Dropout slojevi koji smanjuju overfiting. Prvi sloj je sloj sa 512 neurona, koji koristi ReLU aktivaciju. Drugi sloj je sloj sa 256 neurona, koji takođe koristi ReLU aktivaciju i L2 regularizaciju sa parametrom 0.001. Treći sloj je sloj sa 128 neurona, koji koristi ReLU aktivaciju i L2 regularizaciju sa parametrom 0.001. Čeetvrti sloj je sloj sa jednim neuronom i sigmoid aktivacijom.
Obe neuronske mreže koriste za optimizator adam, a za funkciju gubitka binary_crossentropy.

6. REZULTATI I DISKUSIJA

Tabela 1. *Rezultati modela za najbolji film*

Modeli	10-unakrsna validacija
Logistička Regesija	87,92%
SVM	87,45%
Random Forest	**91,59%**
RF sa Bagging-om	**91,56%**
XGBoost	90,49%
NN	89,78%

Analizom dobijenih rezultata utvrđeno je da najbolje performanse za predikciju za najbolji film imaju Random Forest i Random Forest sa Bagging-om, takođe i Logistička regresija, a najgore SVM model.

Tabela 2. *Rezultati modela za glumu*

Modeli	10-unakrsna validacija
Logistička Regresija	**83,75%**
SVM	83,54%
Random Forest	82,81%
RF sa Bagging-om	81,96%
XGBoost	**84,39%**
NN	81,16%

Analizom dobijenih rezultata utvrđeno je da najbolje performanse za predikciju za glumu imaju XGBoost i Logistička regresija, takođe i SVM, a najgore neuronska mreža.

Eksplorativnom analizom podataka utvrđeno je da filmovi koji su osvojili Oskara često imaju i nominaciju za najboljeg režisera. Takođe kao što je i očekivano da najviše nominovanih filmova za nagradu Oskar i onih koji su osvojili tu nagradu pripadaju žanru drama.

Eksplorativnom analizom podataka utvrđeno je da veću šansu imaju glumci da osvoje Oskara ako je film u kojem su glumili nominovan za Oskara. Takođe kao što je i očekivano da najviše nominovanih glumaca za nagradu Oskar i onih koji su osvojili tu nagradu su glumili u filmovima koji pripadaju žanru drama. Što se tiče odnosa nagrade Oskar sa drugim nagradama, veću verovatnoću da osvoje Oskara imaju glumci ako su osvojili Zlatni globus za dramu nego za komediju.

Rad [8] fokusira se na predikciju ekonomske uspešnosti filma, dok ovaj rad predviđa osvajanje Oskara za najbolji film. Iako oba koriste slične tehnike mašinskog učenja (SVM, Random Forest, MLP), ključna razlika leži u vrsti podataka i ciljevima. Rad [8] koristi filmove iz perioda 1980–2019. i uključuje ekonomske faktore poput budžeta i prihoda, dok ovaj rad obuhvata širi vremenski period (1961–2021) i fokusira se na predviđanje nagrada. Random Forest je bio najuspešniji u oba rada, ali su MLP i SVM pokazali različite performanse u zavisnosti od prirode problema.

7. ZAKLJUČAK

U ovom radu predstavljen je sistem za predikciju dobitnika nagrade Oskar u kategorijama za najbolji film i za glumačko ostvarenje.

Jedan od ključnih zaključaka jeste da prisustvo nominacija i osvajanja drugih prestižnih filmskih nagrada, poput BAFTA, SAGA, DGA, PGA i Zlatnog globusa, značajno povećava šanse za osvajanjem Oskara, što ih čini jednim od najvažnijih faktora.

Najbolji rezultati za predikciju najboljeg filma došli su od Random Forest-a i Random Forest-a sa Bagging-om, dok je SVM bio najlošiji. Za predikciju glumačkih nagrada, najbolji su bili XGBoost i Logistička regresija, a najlošija neuronska mreža.
Kako bi se poboljšalo rešenje, potrebno je razmotriti proširenje skupova podataka, uključujući informacije o ceremonijama Oskara pre 1961. i posle 2021. godine, kao i dodavanje novih parametara poput opisa ili finansijskih informacija.

8. LITERATURA

[1] Jeffrey S. Simonoff, Ilana R. Sparrow, Predicting movie grosses: Winners and losers, blockbusters and sleepers, 2000.
[2] Iain Pardoe, "Just how predictable are the Oscars?", 2005.
[3] Iain Pardoe, Dean K. Simonton, Applying discrete choice models to predict Academy Award winners, 2008.
[4] Predicting the 85th Academy Awards: Stephen Barber, Kasey Le, Sean O'Donnell December 13, 2012
[5] Prediction of Movies popularity Using Machine Learning Techniques: Muhammad Hassan Latif, Hammad Afzal, National University of Sceinces and technology, Pakistan, August 2016.
[6] Performance evaluation of seven machine learning classification techniques for movie box office success prediction: Nahid Quader, Md. Osman Gani, Dipankar Chaki , December 2017.
[7] Predicting the "Best Picture" Oscar Award Winner: Paul Ables, 2018.
[8] Revisiting predictions of movie economic success: random Forest applied to profits: Thaís Luiza Donega e Souza, & Marislei Nishijima, Ricardo Pires, Mart 2023.
[9] https://www.kaggle.com/datasets/matevaradi/oscar-prediction-dataset(pristupljeno u januaru 2024.)
[10] https://zenodo.org/records/4244691(pristupljeno u januaru 2024.)

Kratka biografija:

Jelena Milijević rođena je u Beogradu 1998. god. Master rad na Fakultetu tehničkih nauka iz oblasti Računarstva i automatike odbranila je 2024.god.
kontakt:
jelenamilijevic98@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/30BE08Bugarski

UPOTREBLJIVOST KORISNIČKOG INTERFEJSA APLIKACIJE RAZVIJENE IONIC OKRUŽENJEM

USABILITY OF THE USER INTERFACE OF AN APPLICATION DEVELOPED IN THE IONIC FRAMEWORK

Ivana Bugarski, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *U okviru ovog rada analizirana je upotrebljivost korisničkog interfejsa aplikacije razvijene Ionic okruženjem. Uočene su razlike u performansama aplikacije u zavisnosti od programskog jezika korišćenog u Ionic okruženju pri razvijanju aplikacije.*

Ključne reči: *Ionic, React, Angular, Vue*

Abstract – *This paper analyzes the usability of the user interface of an application developed in the Ionic framework. Differences in the application's performance were observed depending on the programming language used in the Ionic framework during the application's development.*

Keywords: *Ionic, React, Angular, Vue*

1. UVOD

Porast korišćenja mobilnih telefona među korisnicima izazvao je porast u razvoju mobilnih aplikacija. Ova tendencija je podstakla tražnju za alatima koji omogućavaju kreiranje aplikacija kompatibilnih sa svim platformama, sa ciljem unapređenja kvaliteta mobilnih aplikacija. Pored univerzalnosti, javila se potreba za alatom gde će programeri implementirati delove koda uz pomoć već poznatih radnih okvira. Prethodno navedeno olakšava proces implementacije mobilnih aplikacija.

Tema istraživanja fokusira se na evaluaciju upotrebljivosti korisničkog interfejsa razvijenog uz upotrebu *Ionic* okruženja. Cilj istraživanja je analizirati performanse i korisničko iskustvo interfejsa, istražiti tehničke aspekte implementacije, kao i pružiti detaljan pregled koraka u izradi istog.

Ionic okruženje spada među pet najzastupljenijih radnih okvira za razvoj multiplatformskih aplikacija. Najzastupljeniji okvir je *Flutter*, zatim sledi *ReactNative*, potom *Kotlin*, a nakon njega dolazi *Ionic*. Jedna od ključnih prednosti *Ionic* okruženja u odnosu na prethodno navedena okruženja je njegova sposobnost da koristi poznate veb tehnologije poput *HTML*, *CSS* i *JavaScript* biblioteke, što omogućava brzu adaptaciju programera na razvoj mobilnih aplikacija i olakšava razvoj multiplatformskih aplikacija.

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

2. ARHITEKTURA IONIC OKRUŽENJA

Ionic predstavlja otvoreni okvir za razvoj hibridnih mobilnih aplikacija primenom veb tehnologija poput *HTML*, *CSS* i *JavaScript* ili *TypeScript* biblioteke. Ovaj alat omogućuje stvaranje mobilnih aplikacija koje su interoperabilne na više platformi, uključujući *Android*, *iOS* i veb. Dizajniran je da radi i da se prikazuje na svim trenutnim mobilnim uređajima i platformama onako kako je implementiran.

Sadrži gotove komponente, tipografiju i osnovnu temu prilagođenu svakoj platformi. Iako se *Ionic* aplikacije baziraju na veb tehnologijama, pružaju korisnicima iskustvo slično nativnim aplikacijama.

To znači pristup hardverskim funkcijama uređaja, brzo reagovanje kao i glatke animacije. Podržava različite *JavaScript* okvire kao što su *Angular*, *React* i *Vue*, što pruža mogućnost odabira tehnologije od strane razvojnog tima aplikacije.

Kao što je prethodno navedeno, *Ionic* okruženje može koristiti različite radne okvire za implementaciju mobilnih aplikacija, uključujući *Angular*, *React* i *Vue*, što će biti detaljnije obrađeno u nastavku teksta.

Na Slici 1 prikazane su komponente koje učestvuju u izgradnji mobilnih i veb aplikacija. Prilikom korišćenja Ionic okruženja za razvoj mobilnih i veb aplikacija, neophodno je implementirati aplikaciju koristeći neku od frontend tehnologija u sklopu Ionic okruženja.

Takođe, potrebno je uključiti *Capacitor* koji omogućava pristup nativnim funkcionalnostima uređaja i dodatno doprinosi njegovoj efikasnosti i upotrebljivosti.

Slika 1. *Prikaz komponenti za izgradnju mobilnih i veb aplikacija*

2.1. Angular

Angular je *JavaScript* okvir otvorenog koda za izgradnju veb aplikacija. Razvijen je od strane *Google* kompanije i koristi se za razvoj robusnih i skalabilnih aplikacija. *Angular* se oslanja na komponentnu arhitekturu, što znači da se aplikacija sastoji od komponenata koje se mogu ponovo koristiti i lako održavati [1].

2.2. React

React je biblioteka *JavaScript* programskog jezika koji uz predstavljanje pomoću komponenti doprinosi fleksibilnosti i učinkovitosti. Koristi se pri izradi klijentske strane veb aplikacije. Razvijen je od strane *Meta* kompanije koristeći ga za razvoj svoje aplikacije *Facebook*. Kasnije je primenjen i na *Instagram* aplikaciju, nakon čega je u upotrebi mnogih programera. Veoma je pogodan za izradu mobilnih aplikacija kao i veb sajtova, zbog dinamičkog i interaktivnog prikaza korisnicima [2].

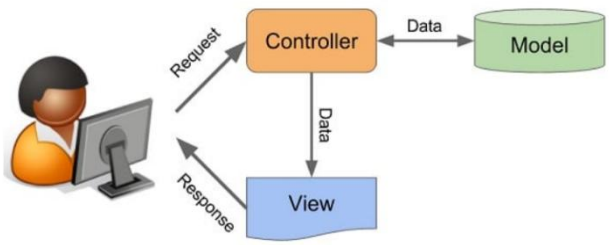
2.3. Vue

Vue je *JavaScript* okvir za izgradnju korisničkih interfejsa. Korisnički interfejs implementira uz pomoć *HTML*, *CSS* bibliotekama kao i koda implementiranog u *JavaScript* jeziku. Pruža deklarativni komponentni model programiranja sa ciljem da se dobije željeni korisnički interfejs [3].

2.4. Ionic

Ionic predstavlja moćno okruženje za razvoj hibridnih mobilnih aplikacija koji, u kombinaciji sa radnim okvirima kao što su *React*, *Angular* i *Vue*, omogućava efikasan i efektivan razvoj aplikacija. Bez obzira na odabrani radni okvir, komponente i alati *Ionic* okruženja pružaju potrebnu podršku za kreiranje aplikacija koje zadovoljavaju visoke standarde performansi i korisničkog iskustva. S obzirom na sve ove prednosti, *Ionic* se nameće kao logičan izbor za timove koji žele da razvijaju mobilne aplikacije uz minimalne troškove i maksimalnu efikasnost [4].

Ionic primenjuje *MVC* arhitekturu kroz *Angular* za formiranje jednostrane aplikacije koja je upotrebljiva za razne mobilne uređaje. Ovaj pristup razdvaja celu aplikaciju na tri dela, a to su *Model*, *View* i *Controller*. *Model* je odgovoran za održavanje podataka. *View* je odgovoran za prikazivanje podataka korisniku, odnosno deo koji se odnosi na *HTML* i *CSS*. *Controller* kontroliše interakcije između *Model* i *View* delova [5]. Na Slici 2 prikazana je arhitektura *MVC* modela.



Slika 2. *Prikaz MVC arhitekture*

3. IMPLEMENTACIJA

Aplikacija koja služi za ilustraciju je jednostavna aplikacija koja omogućava pregled liste korisnika kao i prikaz detalja o svakom korisniku. Takođe, pruža mogućnost ažuriranja podataka o korisniku.

Nakon konfiguracije radnog okruženja i instalacije potrebnih paketa, započinje se proces implementacije aplikacije.

Prethodno je naglašeno da je izgled aplikacije od izuzetne važnosti iz perspektive korisnika. Postoji mnogo biblioteka koje se koriste za uređivanje izgleda aplikacije, a koje se često koriste i u aplikacijama razvijenim isključivo u *React* programskom jeziku. Neke od tih biblioteka su *Material UI*, *Chakra UI* i druge. U implementiranoj aplikaciji korišćena je biblioteka *Chakra UI*, zbog svoje sposobnosti da olakša rad sa brendiranjem i temom aplikacije, kao i to što poseduje veliki broj već implementiranih komponenti. *Chakra UI* nudi različite mogućnosti za unapređenje elemenata aplikacije, što je čini pogodnom za kreiranje vizuelno privlačnih i funkcionalnih korisničkih interfejsa.

U kontekstu *Angular* programskog jezika, kod se može implementirati sa dodatnim koracima koji su karakteristični za ovaj okvir. Da bi se postiglo funkcionalno okruženje za rad aplikacije, neophodno je sprovesti određene konfiguracije, uključujući postavke verzija, rutiranje, podešavanje tema i druge aspekte. Ovi zahtevi se zadovoljavaju putem definisanja odgovarajućih parametara i postavki u više datoteka.

Struktura direktorijuma organizovana je tako da se u *src* direktorijumu nalaze datoteke koje obuhvataju konfiguraciju verzije, pokretanja, rutiranja i druge aspekte aplikacije.

U suprotnosti sa *Ionic* aplikacijom implementiranom uz *React*, kod implementacije sa *Angular* bibliotekom primetna je razlika u organizaciji komponenti. Konkretno, u *React* biblioteci se često sva tri dela *HTML*, *CSS* i funkcionalna logika integrišu unutar jedne komponente, dok su ovi aspekti kod Angular biblioteke često razdvojeni na tri odvojene komponente. Razdvajanje ovih elemenata na tri komponente može imati svoje prednosti, uključujući veću preglednost koda, poboljšanu čitljivost, kao i bržu identifikaciju željenih delova koda. Međutim, kada se svi ovi aspekti kombinuju unutar jedne komponente, kao što je slučaj u *React* biblioteci, to može doprineti bržim izmenama, kako u funkcionalnom delu tako i u stilskom.

Dodatno, u *React* biblioteci postoji prednost renderovanja koja omogućava brzo osvežavanje samo delova stranice koji su podložni promenama. Na primer, nakon izmena u kodu koji se odnosi na tabelu, samo će se tabela osvežiti i prikazaće se promene, što doprinosi bržem iterativnom razvoju aplikacije. Opisani princip renderovanja je posebno koristan zbog svoje efikasnosti u procesu implementacije aplikacija, obezbeđujući brže i intuitivnije iskustvo za razvojne timove.

Za precizno merenje performansi implementiranih aplikacija, korištena je *performance* funkcija u *JavaScript* biblioteci. Ovaj objekat omogućava pristup različitim

metodama i svojstvima koja su ključna za analizu performansi veb aplikacija. *Performance* objekat pruža detaljne informacije o vremenu izvršavanja koda, kao i o performansama mrežnih zahteva i drugih relevantnih aspekata u radu aplikacije u pretraživaču. Ključne funkcije uključuju *performance.now()*, koja se koristi za tačno merenje vremena izvršavanja, kao i svojstva poput *performance.timing()* i *performance.memory()*, koja pružaju informacije o vremenskim tačkama učitavanja resursa i upotrebi memorije aplikacije.

4. ANALIZA UPOTREBLJIVOSTI

U analizi *UI* iz korisnikovog ugla, fokus je na interakciji i doživljaju korisnika tokom korišćenja aplikacije.

Koristeći *Ionic* alat uz *Angular*, *React* ili *Vue*, korisnici mogu primetiti da je *UI* dizajniran sa naglaskom na mobilno iskustvo. Na primer, navigacija je obično jednostavna i intuitivna, sa jasno označenim elementima kao što su naslovi i dugmad. Dodatno, korisnici mogu uočiti brzinu interakcije, što može doprineti pozitivnom doživljaju. Takođe, korisnici mogu ocenjivati i faktore kao što su estetika, organizacija sadržaja, upotrebljivost ikona i intuitivnost navigacije. Dobro postavljeni elementi mogu olakšati korisnicima pronalaženje željenih funkcionalnosti, dok previše složen *UI* može izazvati konfuziju ili frustraciju.

Uzimajući u obzir sve ove faktore, analiza korisničkog interfejsa omogućava da se sagleda kako korisnici doživljavaju i interaguju sa aplikacijom, čime se dobija uvid u potencijalne oblasti za poboljšanje ili optimizaciju korisničkog iskustva.

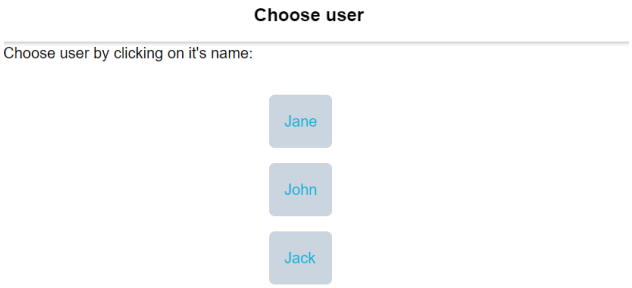
4.1. Demonstracija

Demonstracija korisničkog interfejsa implementiranog u okruženju *Ionic* uz upotrebu *React*, *Angular* i *Vue* biblioteke predstavljena je na slikama ispod. Pošto je izgled za sve tri aplikacije isti, analiziraće se performanse i korisničko iskustvo sa stanovišta korisnika. Vizuelni elementi poput nijansi boja, veličina i jačine teksta su dosledno primenjeni u svim primerima, ključne razlike se pojavljuju u performansama i odzivu aplikacija tokom njihovog rada. *React*, *Angular* i *Vue* su svi moćni alati za izradu korisničkih interfejsa u okviru *Ionic* platforme, ali svaki od njih ima svoje prednosti i mane.

Prvo je predstavljena početna stranica sa prikazom liste korisnika, gde je svaki korisnik prikazan kao dugme. Klikom na to dugme prelazi se na stranicu sa detaljima o izabranom korisniku. Na stranici sa detaljima o korisniku, omogućeno je ažuriranje prikazanih detalja. Opisani način funkcionisanja korisničkog interfejsa je intuitivan i jednostavan za upotrebu. Korisnici sa minimalnim računarskim iskustvom mogu lako da zaključe da će se detalji odabranog korisnika prikazati nakon selektovanja željenog korisnika.

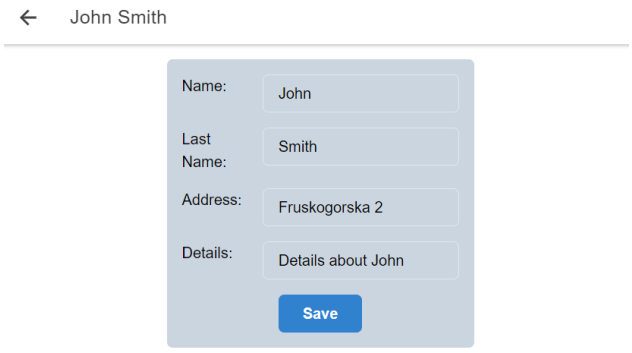
Dodatno, korisnički interfejs podstiče interakciju time što se kursor menja u oblik pokazivača prstom kada se pređe preko dugmeta sa imenom korisnika, što jasno sugeriše da je potrebno pritisnuti dugme kako bi se dobile dodatne informacije. Ovaj pristup dizajnu korisničkog interfejsa naglašava jednostavnost i intuitivnost, omogućavajući korisnicima da lako i brzo da pristupe potrebnim informacijama.

Na Slici 3 može se primetiti da je dizajn stranice *UserListPage* jednostavan. Stranica sadrži naslov, tekstualni opis i listu sa imenima korisnika. Tekstualni opis detaljno objašnjava šta treba korisnik da uradi kako bi video detalje o željenom korisniku prikazanog u listi. Lista korisnika prikazana je uz pomoć dugmadi, gde svako dugme ima istu nijansu pozadine, istu veličinu kao i isti stil slova kojima su imena korisnika predstavljena. Ukoliko bi se dugmad ili tekst razlikovali, stranica ne bi podržala konzistentnost i jednostavnost. Nije poželjno nemati boje i stil kako je to prikazano na prethodnim slikama, ali i preterana šarolikost može da dovede do nepreglednosti stranice i njenih delova.



Slika 3. *Prikaz UserListPage stranice*

Nakon što je korisnik selektovan, dobija se prelaz na stranicu sa detaljima o selektovanom korisniku. Slika 4 prikazuje stranicu *UserDetailPage* gde su prikazani detalji izabranog korisnika. Na osnovu prikazanog primer izgleda *UserDetailPage* stranice može da se zaključi da detalji prikazani na Slici 4, privlače pažnju i korisnika zadržavaju na stranici. Čine stranicu zanimljivijom i lakšom za gledanje kao i sam rad na njoj.



Slika 4. *Prikaz UserDetailPage stranice*

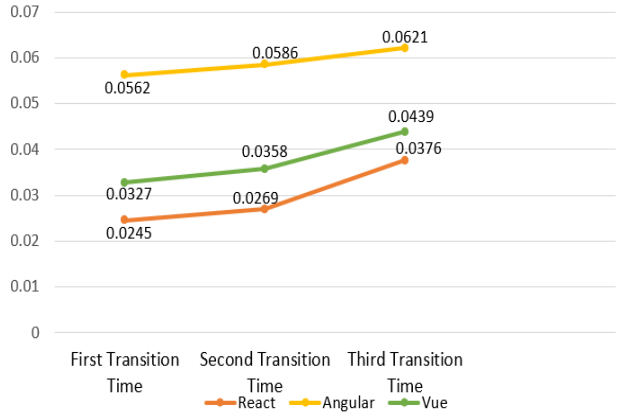
Nakon demonstracije aplikacija, može se zaključiti da izgled ostaje konzistentan bez obzira na to da li je korišćena *React*, *Angular* ili *Vue* biblioteka u *Ionic* okruženju.

Međutim, razlike se uočavaju u performansama sistema. Što se tiče brzine rada aplikacije, najbrža se pokazala aplikacija implementirana pomoću *React* biblioteke, čija se brzina nije menjala ni tokom procesa implementacije. Aplikacija implementirana *Angular* bibliotekom izdvaja se u odnosu na *React* i *Vue* po tome što ima sporije

inicijalno učitavanje zbog svoje kompleksnosti i obuhvatnosti. S druge strane, aplikacija implementirana pomoću *Vue* biblioteke zaostaje za *React* bibliotekom u pogledu brzine renderovanja i responzivnosti.

Nakon analize korišćenjem *performance* objekta, utvrđeno je da je vreme učitavanja *UserDetailPage* stranice nakon klika na dugme sa imenom korisnika bilo 0,0245 sekundi za *React*, 0,0562 sekunde za *Angular* i 0,0327 sekundi za *Vue* biblioteku. Nakon izmene podataka selektovanog korisnika vreme za koje se izvrši izmena podataka za *React* bilo je 0,0269 sekundi, za *Angular* 0,0586 i za *Vue* biblioteku bilo je 0,0358 sekundi. Vreme učitavanja *UserListPage* stranice nakon klika na dugme za vraćanje sa stranice *UserDetailPage* bilo je 0,0376 sekundi za *React*, 0,0621 sekundi za *Angular* i 0,0439 sekundi za *Vue* biblioteku. Na osnovu ovih rezultata može se zaključiti da je implementacija aplikacije u *React* biblioteci pokazala najbrže vreme učitavanja u poređenju sa *Angular* i *Vue* bibliotekama.

Na Slici 5 prikazani su podaci dobijeni korišćenjem *performance* objekta za merenje vremena prelaska između stranica za biblioteke *React*, *Angular* i *Vue*. Početne tačke na grafikonu predstavljaju vreme prelaska sa stranice *UserListPage* na stranicu *UserDetailPage*, dok krajnje tačke prikazuju vreme prelaska sa stranice *UserDetailPage* na stranicu *UserListPage*.



Slika 5. *Prikaz podataka dobijenih korišćenjem performance objekta*

Ionic se pokazao kao izuzetno učinkovit alat za izgradnju visokokvalitetnih aplikacija sa glatkim performansama i atraktivnim korisničkim interfejsima. Njegova integracija sa *Capacitor* bibliotekom, koja omogućava pristup nativnim funkcionalnostima uređaja, dodatno doprinosi njegovoj efikasnosti i upotrebljivosti. Ova fleksibilnost i sposobnost da pruži nativno iskustvo korisnicima, čini ga idealnim izborom za mnoge razvojne projekte. Upotreba *Ionic* okruženja ima smisla i sada i u budućnosti, jer kontinuirano unapređuje svoje mogućnosti i prati najnovije trendove u tehnologiji razvoja mobilnih aplikacija. Njegova popularnost je rezultat kombinacije fleksibilnosti, jednostavnosti upotrebe i moćnih funkcionalnosti, što ga izdvaja od ostalih alata na tržištu.

Kao rezultat toga, *Ionic* je popularan među programerima koji žele da kreiraju visoko performansne, multiplatformske aplikacije sa minimalnim naporom.

5. ZAKLJUČAK

Korišćenje *Ionic* okruženja kao alata za razvoj mobilnih aplikacija pruža brojne prednosti koje ga čine idealnim za savremene potrebe programera. Njegova sposobnost da omogući razvoj jedne aplikacije za više platformi, uključujući *iOS*, *Android* i veb, značajno smanjuje vreme i troškove razvoja, što je od ključnog značaja u brzorastućem svetu mobilnih tehnologija.

Nakon detaljne analize korisničkog interfejsa aplikacija razvijenih korišćenjem *React*, *Angular* i *Vue* biblioteka unutar *Ionic* okruženja, uočene su značajne razlike u performansama i brzini rada.

U zaključku, *Ionic* se izdvojio kao izuzetno korisno okruženje za razvoj multiplatformskih aplikacija, zahvaljujući svojoj fleksibilnosti, jednostavnosti upotrebe i moćnim funkcionalnostima. Njegova sposobnost da integriše *JavaScript* biblioteke i pristup nativnim funkcionalnostima uređaja čini ga idealnim izborom za mnoge razvojne projekte. Pored toga što ne uspeva da pruži odgovarajuću efikasnost za previše zahtevne aplikacije, *Ionic* pruža efikasan i ekonomičan način za razvoj visokokvalitetnih mobilnih aplikacija sa minimalnim naporom.

6. LITERATURA

[1] Angular, https://angular.dev/guide/components (pristupljeno u maju 2024.)
[2] React, https://react.dev/learn (pristupljeno u maju 2024.)
[3] Vue, https://vuejs.org/guide/introduction (pristupljeno u maju 2024.)
[4] Ionic, https://ionic.io/blog/announcing-the-ionic-react-beta (pristupljeno u junu 2024.)
[5] Priyanka Chaudhary, „International Research Journal of Engineering and Technology (IRJET)“, Student, Department of computer science, ABESIT, Ghaziabad, Volume: 5, pp. 3182, May 2018.

Kratka biografija:

Ivana Bugarski rođena je u Novom Sadu 1999. godine. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Upotrebljivost korisničkog interfejsa aplikacije razvijene Ionic okruženjem odbranila je 2024. godine.
kontakt: ivanabugarski19@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/30BE09Krsmanovic

RAZVOJ APLIKACIJE KORIŠĆENJEM MODULAR MONOLIT ARHITEKTURE I EKSTRAKCIJA MIKROSERVISA IZ MODULA

DEVELOPMENT OF APPLICATION USING MODULAR MONOLITH ARCHITECTURE AND EXTRACTION OF MICROSERVICE FROM MODULE

Đorđe Krsmanović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

Kratak sadržaj – *U okviru rada kreirana je specifikacija imaginarnog softverskog sistema, koji je razvijen upotrebom modular monolit arhitekture. Sistem je dovoljno kompleksan, kako bi se simulirao rad na realnom projektu. Zatim je izvršena ekstrakcija mikroservisa iz modula, čime se prelazi na mikroservisnu arhitekturu. Cilj rada je istražiti koliko je izazovno izvršiti tranziciju sa modular monolitne na mikorservisnu arhitekturu. U radu su predstavljene obje arhitekture sa akcentom na načine komunikacije i rad sa podacima. Opisan je proces rada i koraci koje je potrebno uraditi kako bi se softverski sistem implementirao. Zatim su putem C4 dijagrama predstavljeni rezultati rada na aplikaciji i opisane izmjene do kojih dolazi prilikom tranzicije sa modularne na mikroservisnu arhitekturu. Na samom kraju date su prednosti i mane obje arhitekture, kao i smjernice za poboljšanje rada i dalje istraživanje.*

Ključne reči: *arhitektura softvera, modularni monolit, mikroservisna arhitektura,*

Abstract – *As part of this paper, a specification for an imaginary software system is created, which was developed using a modular monolith architecture. The system is sufficiently complex to simulate work on a real-world project. Subsequently, microservice was extracted from the module, transitioning the system to a microservice architecture. The goal of the thesis is to explore the challenges involved in transitioning from a modular monolith to a microservice architecture. The paper presents both architectures, focusing on communication methods and data handling. The development process is described, outlining the necessary steps for implementing the software system. The results of the work are illustrated using C4 diagrams, showcasing the changes that occur during the transition from a modular to a microservice architecture. Finally, the advantages and disadvantages of both architectures are discussed, along with recommendations for improving the process and suggestions for further research.*

Keywords: *software architecture, modular monolith, microservices architecture*

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Nikola Luburić, docent.

1. UVOD

Arhitektura softvera je od suštinske važnosti u razvoju softvera, jer određuje osnovu za strukturu, funkcionalnost, skalabilnost i održivost sistema. Veliki broj inženjera se odlučuje za upotrebu mikroservisne arhitekture, jer određene komponente sistema koje koristi veći broj korisnika lako možemo skalirati i postići dobre performanse sistema. Međutim, mikroservisna arhitektura je sama po sebi kompleksna i troši više resursa na razvoj i testiranje softvera. Najbolji pristup bio bi kreirati softver na takav način da, kada uvidimo potrebu za skaliranjem određenog dijela aplikacije, imamo mogućnost da od njega relativno lako napravimo mikroservis. Modular monolit arhitektura upravo to omogućava.

Ovaj rad bavi se razvojem softvera i izborom pravilne arhitekture. Na početku je kreirana specifikacija jednog softverskog sistema. Zatim je taj softverski sistem implementiran upotrebom modular monolit arhitekture. Cilj rada je da se utvrdi koliko je izazovno od jednog modula aplikacije napraviti mikroservis.

U drugom poglavlju biće opisane teorijske osnove modular monolit i mikroservisne arhitekture. Naredno poglavlje baviće se opisom procesa rada na softverskom sistemu, gdje će biti navedene sprovedene aktivnosti tokom razvoja softvera. U četvrtom poglavlju biće opisani rezultati rada, uz dijagrame aplikacija, te će biti utvrđene razlike u arhitekturi sistema. Naredno poglavlje opisuje iskustvo razvoja softvera s obje arhitekture, uključujući njihove prednosti i mane. Poslednje poglavlje donosi zaključak sa sažetkom istraživanja i prijedlozima za buduća unapređenja.

2. TEORIJSKE OSNOVE

U ovom poglavlju biće razmatrane teorijske osnove modularne monolitne i mikroservisne arhitekture, sa posebnim akcentom na aspekte komunikacije i izolacije podataka.

2.1. Arhitektura modularnog monolita

Modular monolit arhitektura kombinuje paradigmu modularnog razvoja softvera i monolitne arhitekture. Modularni monolit je arhitektonski obrazac koji strukturira aplikaciju u nezavisne module ili komponente sa dobro definisanim granicama. Moduli su podijeljeni na osnovu logičkih granica, grupišući zajedno povezane funkcionalnosti, te svaki ima dobro definisan interfejs za komunikaciju. Iako moduli mogu biti organizovani u odvojene jedinice, oni su čvrsto integrisani i ne mogu

potpuno funkcionisati bez međusobne saradnje. Ova arhitektura koristi zajedničku bazu koda i skladište podataka, što pojednostavljuje razvoj i isporuku, ali istovremeno omogućava dobru organizaciju koda. Fleksibilnost je još jedna prednost, jer omogućava dodavanje i brisanje modula bez velikog uticaja na cijeli sistem, a preopterećeni moduli mogu se izdvojiti u mikroservise i horizontalno skalirati. Isporuka softvera je jednostavnija zbog postojanja samo jednog artefakta.

2.1.1. Komunikacija između modula

Jedno od osnovih pravila modularne arhitekture jeste da moduli ne mogu da referenciraju tuđe funkcionalnosti direktno, već samo mogu da im pristupe putem javnog API drugog modula.

Najjednostavniji način komunikacije je sinhrona komunikacija, sa direktnim pozivima metoda između modula. Mana ovog pristupa je čvrsta spregnutost između modula, ukoliko jedan modul nije dostupan to će uticati i na dostupnost drugog modula.

Sljedeći način komunikacije je asinhrona komunikacija putem poruka. Asinhrona komunikacija smanjuje spregnutost između modula, ali se povećava kompleksnost, jer se uvodi sistem za razmjenu poruka.

2.1.2. Izolacija podataka

Izolacija podataka između modula je ključna za održavanje njihove nezavisnosti i slabe povezanosti, pri čemu svaki modul može pristupiti samo svojim tabelama u bazi podataka i nije dozvoljeno da vrši upite nad podacima drugih modula. Postoji nekoliko tehnika za izolaciju podataka:

- odvojene tabele – sve tabele se nalaze u istoj bazi podataka, što može biti jednostavno u ranim fazama razvoja
- odvojene šeme – tabele se grupišu i logički izoluju pomoću šema u bazi, gdje svaki modul ima sopstvenu šemu
- odvojene baze – svaki modul ima svoju bazu podataka, što omogućava veću izolaciju ali dodaje kompleksnost u održavanju infrastrukture
- drugačija vrsta skladišta - koriste se različiti tipovi skladišta, poput graf baza ili drugih nerelacionih baza, za specifične poslovne potrebe modula.

2.2. Mikroservisna arhitektura

Mikroservisna arhitektura je pristup razvoju jedne aplikacije kao skupa malih usluga, gdje se svaka pokreće u svom procesu i komunicira jednostavnim mehanizmima, često pomoću HTTP-a [1]. Svaki servis implementira zaseban poslovni zahtjev i može se nezavisno pustiti u rad potpuno automatizovano. Postoji minimum centralizovanog upravljanja ovim uslugama, koje mogu biti napisane na različitim programskim jezicima i koristiti različite tehnologije za skladištenje podataka [2].

Može se reći da u mikroservisnoj arhitekturi aplikacija je sačinjena od velikog broja slabo spregnutih i nezavisno isporučenih komponenti. Mikroservisi se organizuju prema biznis logici koju trebaju da rješavaju. Pojedinačni servisi obavljaju jedan zadatak koji predstavlja jednu poslovnu funkciju cjelokupnog sistema. Prilikom razvoja mikroservisa svaki mikroservis može da ima različite tehnologije uključujući izbor baze i programskog jezika.

2.2.1. Komunikacija između mikroservisa

Servisi moraju sarađivati da bi ispunili korisničke zahtjeve. Postoje dva osnovna tipa komunikacije:

- Sinhrona komunikacija – klijent šalje zahtjev mikroservisu i čeka odgovor. Najčešće se koristi REST [3] ili gRPC [4] protokol. REST koristi HTTP glagole i URL-ove za manipulaciju resursima, dok gRPC omogućava pozivanje metoda na udaljenim serverima koristeći binarne protokole, čineći ga efikasnijim, ali zahtijeva HTTP/2.
- Asinhrona komunikacija – servisi komuniciraju slanjem poruka putem posrednika (message broker), npr. RabbitMQ [5], ActiveMQ [6], Apache Kafka [7]. Posrednik baferuje poruke i šalje ih kada primalac bude spreman. Izbor posrednika zavisi od zahtjeva aplikacije, pri čemu se balansira između latencije i pouzdanosti.

2.2.2. Rad sa podacima u mikroservisima

Većina servisa ima potrebu da sačuva svoje podatke u nekoj vrsti baze podataka. Postoje dva različita pristupa:

- Dijeljena baza podataka – koristi se jedinstvena baza podataka za više servisa. Svaki servis slobodno pristupa svojim podacima i podacima drugih servisa koristeći lokalne ACID transakcije.
- Posebna baza za svaki servis – svaki servis ima svoju bazu podataka. Podaci su privatni za servis. Transakcije koje radi servis odnose se samo na njegovu sopstvenu bazu. Podaci koji njemu ne pripadaju, dostupni su isključivo putem API-ja drugih servisa

Pristup posebna baza po servisu se najčešće koristi prilikom razvoja aplikacija korišćenjem mikroservisne arhitekture. On omogućava da servisi budu labavo spregnuti, te promjena u bazi podataka jednog servisa ne utiče na druge servise. Svaki servis može da koristi tip baze podataka koji najviše odgovara njegovim potrebama.

3. METODOLOGIJA

Metodologija rada uključuje razvoj imaginarnog sistema za zakazivanje smještaja kako bi se simulirao rad na realnom sistemu i isprobali različiti arhitekturalni pristupi. Projekat je obuhvatio specifikaciju dizajna, implementaciju modularnog monolita i ekstrakciju mikroservisa.

Specifikacija dizajna uključuje klasne i sekvencijalne dijagrame koji prikazuju statičku i dinamičku strukturu sistema. Prilikom izrade klasnog dijagrama izvršena je podijela aplikacije na tri modula:

- Accommodation
- Commerce
- User Access.

Implementacija modularnog monolita je rađena u .NET [8] i Angular [9] radnim okvirima, uz korišćenje Domain Driven Design-a [10] i čiste arhitekture [11].

Ekstrakcija mikroservisa je počela sa Accommodation modulom, jer se očekivalo da će korisnici najviše koristiti funkcionalnosti ovog module, te bi ga trebalo skalirati. Prilikom ekstrakcije, modularni monolit je podijeljen tako da je svaki mikroservis funkcionisao kao zasebna jedinica, sa sopstvenim repozitorijumom koda. Organizacija koda u više repozitorijuma je bila ključna jer je omogućila nezavisan razvoj mikroservisa bez uticaja na druge dijelove aplikacije. Zajednički dijelovi koda, takođe su ekstrahovani u zasebne repozitorijume kako bi mogli biti referencirani od više mikroservisa. Dodat je API Gateway za rutiranje, autentifikaciju i autorizaciju i predstavlja ulaznu tačku u sistem.

4. REZULTATI RADA

U ovom poglavlju biće predstavljeni rezultati rada na projektima. Fokus će biti na C4 [12] modelu dijagrama koji su nacrtani kako bi se što bolje opisala arhitektura aplikacija. C4 dijagrami su metoda za modelovanje softverske arhitekture koja koristi hijerarhijski pristup kako bi se prikazali različiti nivoi detalja sistema.

U potpoglavlju 4.1 biće prikazan i opisan dijagram komponenti koji prikazuje razvoj aplikacije koristeći modularni monolit. Naredno potpoglavlje će prikazati samo promjene do kojih dođe kada se izvrši estrakcija mikroservisa iz modula.

4.1. Modular monolit arhitektura

Dijagram komponenti na slici 4.1 pruža uvid u unutrašnju strukturu sistema. Glavna komponente sistema su klijentska aplikacija, serverska aplikacija, baza podataka i komponenta za razmjenu poruka.

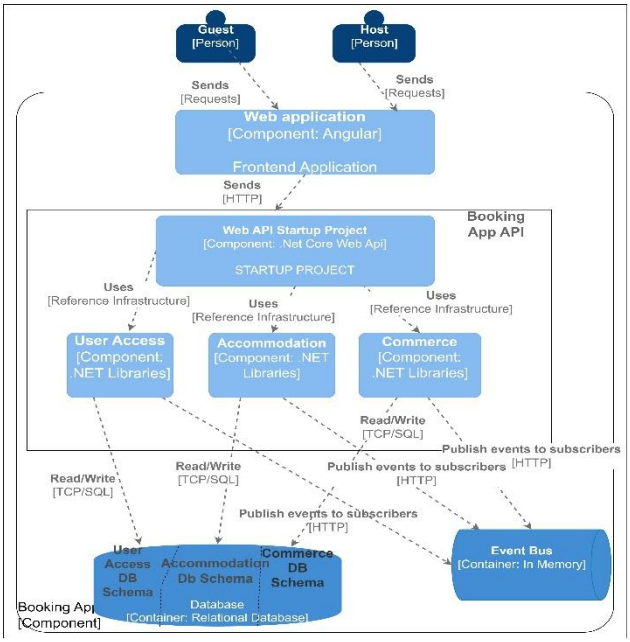
Svaki modul sastoji se od pet .NET klasnih biblioteka, koje predstavljaju prezentacioni, aplikativni, domenski i infrastrukturni sloj, dok peta sadrži tipove poruka koji se razmjenjuju između modula. Web API projekat je centralni dio koji pokreće aplikaciju i konfiguriše module.

Modularna aplikacija koristi jednu relacionu bazu podataka, gdje svaki modul ima sopstvenu šemu baze podataka. Moduli komuniciraju razmjenom poruka, što smanjuje spregnutost i olakšava prelazak na mikroservisnu arhitekturu. U modular monolit arhitekturi se koristi in-memory transport za brzu razmjenu poruka unutar jednog procesa, poboljšavajući performanse bez potrebe za dodatnom infrastrukturom u vidu korišćenja posrednika poruka.

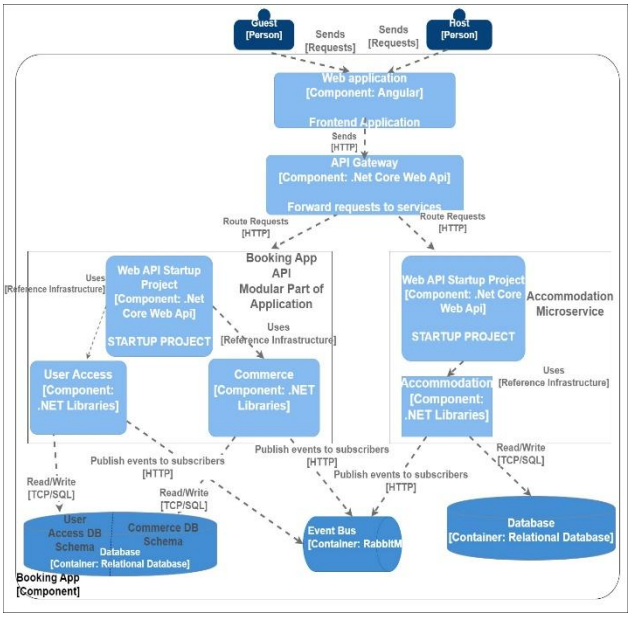
4.2. Mikroservisna arhitektura

Pogledom na dijagram komponenti na slici 4.2 mogu se vidjeti prve promjene koje su posljedica izmjene arhitekture. Samim činom ekstrakcije mikroservisa iz modula, prelazi se na mikroservisnu arhitekturu, koja zahtijeva da servis ima sopstveno skladište. Kreirana je nova baza podataka kojoj će pristup imati ekstrahovani mikroservis. Ostatak modularnog dijela aplikacije u ovom slučaju čini drugi mikroservis. Ovaj mikroservis koristi bazu podataka koja sadrži dvije šeme, gdje svakoj šemi pristupa određeni modul. Takođe dodat je novi Web Api projekat koji ima ulogu API Gateway-a.

Sa prelaskom na mikroservisnu arhitekturu, potrebno je zamijeniti in-memory mehanizam za razmjenu poruka distribuiranim rješenjem koje podržava mrežnu komunikaciju, jer mikroservisi mogu biti na različitim lokacijama. Iz tog razloga, izabran je RabbitMQ kao novi posrednik poruka.



Slika 4.1. *Dijagram komponenti sistema, modular monolit arhitektura*



Slika 4.2. *Dijagram komponenti sistema, mikroservisna arhitektura*

5. DISKUSIJA

U ovom poglavlju biće predstvljene prednosti i mane implementiranih arhitektura. Modularni monolitni pristup nije pogodan za jednostavne aplikacije bez dovoljno biznis logike, ali je odličan za složene aplikacije, jer omogućava dobru organizaciju koda i lako upravljanje složenošću. Moduli su grupisani prema skupu zajedničkih funkcionalnosti, što olakšava migraciju na mikroservise ako su ispraćene dobre prakse razvoja aplikacije. Nedostaci ovog pristupa su nemogućnost Join operacija

između podataka koji pripadaju različitim modulima. Zbog toga većina logike za agregaciju podataka treba da bude na klijentskoj strani.

Mikroservisna arhitektura je idealna za aplikacije koje zahtijevaju skalabilnost i timsku podjelu rada, uz fleksibilnost u izboru tehnologija. Mane uključuju složenost infrastrukture, testiranja, verzionisanja i upravljanja zajedničkim kodom, kao i izazove sa održavanjem konzistentnosti sistema.

6. ZAKLJUČAK

U radu su predstavljeni modularni monolit i mikroservisna arhitektura, pri čemu su razmatrane prednosti i mane obje arhitekture. Implementiran je projekat koristeći modularni monolit kao arhitektonski obrazac, nakon čega je izvršena ekstrakcija mikroservisa iz modula. Motivacija za pisanje rada bila je sprovođenje eksperimenta radi provjere koliko je izazovno preći sa monolitne arhitekture na mikroservisnu.

U današnje vrijeme često imamo monolitne aplikacije, koje je potrebno prebaciti u mikroservise radi što boljeg skaliranja i lakšeg održavanja. Na osnovu rezultata, ukoliko imamo monolitnu aplikaciju, najbolje bi bilo prvo preći na modularni monolit sa jasno definisanim modulima. Ovo bi predstavljalo jednu vrstu međukoraka, jer se suštinski aplikacija sa stanovišta arhitekture ne mijenja značajno. Potrebno je izvršiti refaktorisanje koda i formiranje modula. Nakon toga, kao što je prikazano u radu, tranzicija na mikroservise može se relativno lako izvesti.

Kao unapređenje ovog rada, predlože se analiza oba sistema sa DevOps strane. Dobro bi bilo automatizovati isporuku i testiranje modularne aplikacije i vidjeti do kojih sve promjena dolazi kada se izvrši tranzicija na mikroservisnu arhitekturu. Analizirajući ovaj aspekt razvoja i održavanja sistema, dobila bi se cjelokupnu slika koliki je zaista napor potrebno uložiti da se izmijeni arhitektura aplikacije.

7. LITERATURA

[1] Zvanična dokumentacija HTTP protokola, https://httpwg.org/specs/ [datum pristupa jul 2024]

[2] Microservices article, https://martinfowler.com/articles/microservices.html [datum pristupa jul 2024]

[3] Richardson, Leonard; Ruby, Sam (2007). RESTful Web Services. Sebastopol, California: O'Reilly Media

[4] Zvanična dokumentacija gRPC protokola, https://grpc.io/docs/ [datum pristupa jul 2024]

[5] Zvanična dokumentacija RabbitMQ, https://www.rabbitmq.com/docs [datum pristupa jul 2024]

[6] Zvanična dokumentacija ActiveMQ, https://activemq.apache.org/ [datum pristupa jul 2024]

[7] Zvanična dokumentacija Apache Kafka, https://kafka.apache.org/documentation/, [datum pristupa jul 2024]

[8] Zvanična dokumentacija .NET radnog okvira, https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet [datum pristupa jul 2024]

[9] Zvanična dokumentacija Angular radnog okvira, https://v17.angular.io/docs [datum pristupa jul 2024]

[10] Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software

[11] Robert C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design

[12] Zvanična dokumentacija C4 modela dijagrama, https://c4model.com/ [datum pristupa avgust 2024]

Kratka biografija:

Đorđe Krsmanović rođen je 14. januara 1999. godine u Foči. Osnovno obrazovanje je stekao u osnovnoj školi „Sveti Sava“ u Foči. Nakon toga upisuje gimnaziju – prirodno matematički smjer u Srednjoškolskom centru u Foči. Školske 2018/2019 godine upisuje Fakultet tehničkih nauka u Novom Sadu, odsjek Elektrotehnika i računarstvo, smjer Računarstvo i automatika. Po završetku studija u roku, upisuje master akademske studije koje završava 2024. godine.

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/30BE10Djurisic

RAZVOJ RAČUNARSKE IGRE ZASNOVANE NA KOMPLEKSNOJ PRIČI

DEVELOPMENT OF A VIDEO GAME WITH A COMPLEX STORYLINE

Milan Đurišić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *Celokupna procedura kreiranja inicijalne verzije računarske igre RPG (role playing game) žanra. Ova procedura podrazumeva ne samo kodiranje i kreiranje resursa za računarsku igru, već i pisanje i razvoj priče koja služi kao tema same igre. Analizirane su postojeće igre raznih žanrova, kako bi se što bolje donele odluke o korišćenju elemenata razvoja igre kao što su pozicija i perspektiva kamere, specijalni efekti, izgled i reakcija menija i slično.*

Ključne reči: *Unity Razvojno Okruženje, RPG, Interfejs, Mehanike Igre, Gejming*

Abstract – *Detailed procedure on the development of an RPG (role playing game) style computer game. This procedure describes not only the coding and creation of resources for the video game, but the conceptualization, writing and development of the story that the game is trying to tell. Some existing game genres are analyzed in an effort to better understand how to use certain elements of game development, such as positions and perspectives of the camera component, special effects, and look and feel of menus.*

Keywords: *Unity Development Environment, RPG, Interface, Game Mechanics, Gaming*

1. UVOD

Video-igre u osnovi predstavljaju računarske programe u kojima korisnici kontrolišu slike na ekranu. One se mogu igrati na raznim sistemima, od kojih su najčešći PC računari, i konzole raznih proizvođača. Video-igre omogućavaju korisnicima da dožive drugačije uloge, i da urade razne stvari koje ne bi mogli da urade u stvarnom životu, bez pravih posledica. Prva komercijalna igra bila je Computer Space [1] iz 1971. godine. U ovoj igri igrači upravljaju letelicom koja ispaljuje rakete na protivničke letelice. Naredne godine kompanija Atari pušta u prodaju svoju igru Pong, koja ujedno postaje prva komercijalno uspešna igra.

Igra koja se u ovom radu opisuje naziva se "Kingdom Of Souls". Igra je RPG žanra, kreirana koristeći Unity razvojno okruženje, kao i razne druge alate.

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

2. IGRA "KINGDOM OF SOULS" I INSPIRACIJA ZA RAZVOJ

"Kingdom Of Souls" je 3D RPG igra namenjena za PC računar u kojoj igrači stupaju u ulogu čarobnjaka koji koristi zle demonske magije u svoju korist. Igra je namenjena za igrače starije od 17 godina zbog određenih tema koje se istražuju u okviru igre. Inspiracija za razvoj igre najviše dolazi od igara iz serijala "The Elder Scrolls", "Fallout", i "Baldurs Gate". Ovi serijali su najviše uticali na razvoj igre zbog mogućnosti koje se nude u okviru same igre, što podrazumeva ne samo mehaničke radnje likova (od kretanja do borbe i slično), već i veliki izbor interakcije sa raznim likovima i raznih zadataka koji utiču na razvoj priče.

3. PRIČA KAO OSNOVA ZA RAZVOJ IGRE

Igre koje su inspirisale razvoj igre "Kingdom Of Souls" u osnovi imaju dobro razvijenu priču. Priče im se sastoje ne samo od glavnog junaka i njegovog puta kroz svet do nekog krajnjeg zadatka, već imaju opisanu istoriju sveta u kojem se glavni junak nalazi. Pored istorije sveta, skoro svaki lik sa kojima se igrači susreću u igri imaju neku istoriju, koja može biti opisana površno - za potrebe određenog zadatka, ili toliko detaljno da igrači počnu da idolizuju te likove. Iz ovih razloga, prvi korak u procesu razvoja ove igre bio je osmišljavanje priče.

Obzirom da se radi o igri u kojoj igrači stupaju u ulogu pojedinca - čoveka, priča mora objasniti odakle je taj čovek, odakle je mesto i svet u kom se on nalazi, i zašto je sve uopšte tu. Odgovor je u religiji sveta u kom se glavni lik nalazi i bićima koja su stvorila svet, odnosno objašnjenju zašto su ga stvorili, kao i kratkoj istoriji sveta u nekom intervalu koji se završava sa momentom u kom se nalazi glavni junak.

Bogovi stvaraju planetu Terideju (lat. Territorium Deorum - Teritorija od Bogova) i bićima na planeti daju moći i duše. Ove duše Bogovi preuzimaju nakon smrti bića i postaju jači. To su uradili jer vode konstantnu borbu protiv sila podzemlja i sila demonskog carstva, i ove bitke vode u van-realnom svetu Spiritšir (eng. Spirit Shear - razor duša). Demonske sile su saznale za nameru Bogova da preuzimaju duše smrtnika, i zbog toga su napali smrtni svet i usadili ljudima sumnju u Bogove. Ljudi koji su nevernici i ne priznaju Božanstvene sile nakon svoje smrti ostavljaju svoju dušu u podzemnom svetu iz kog energija njihovih duša ide u pomoć demonskim silama. Demonske sile su, zbog nemogućnosti da ih unište, razbacale spise koji sadrže istinu o kreaciji sveta po celom svetu, a ljudi su uspeli da pronađu samo jedan. Ovaj spis je jedina relikvija koja stvara nove vernike i doprinosi

božanstvenim moćima. Sa ovakvom osnovom priče, donekle je objašnjeno postojanje sveta u kom se dešava radnja igre, kao i neki smisao postojanja bića na planeti.

Glavni lik igre nalazi se u trenutku kada se po planeti otvaraju portali iz kojih izlaze demoni iz podzemlja i napadaju ljude na svetu. Obzirom da se glavni junak bavio magijama koje imaju veze sa demonima, on ima načine da ove demone spreči i vrati nazad u podzemni svet. Glavni problem jeste što je glavni junak baš zbog svog korišćenja zabranjenih magija odbačen od ostatka civilizacije, i sada mora da ubedi kraljicu i njene podanike da mu veruju kako bi uspešno odbranili svet od demonske najezde. Ova premisa obezbeđuje glavni tok radnje u igri, ali i otvara razne druge putanje koje igrači mogu da prate u igri nezavisno od glavne priče. Takođe, istorija glavnog lika obezbeđuje kompletno drugi skup događaja i zadataka ukoliko igrači žele da napuste glavnu priču, i reše glavni cilj zaustavljanja podzemnih sila na svoj način.

4. GEJMPLEJ I RAZVOJ IGRE

Gejmplej se može opisati kao skup aktivnosti pomoću kojih igrači mogu interagovati sa svetom igre. Gejmplej je ograničen pravilima igre koja definišu ciljeve, nagrade i kazne.

Igrači raspolažu životnim, magičnim i kondicionim bodovima za glavnog lika. Životni bodovi se troše prilikom borbe sa neprijateljima, i ako se potpuno istroše glavni lik umire i igra se završava. Drugi način da se igra završi jeste da igrači stignu do poslednjeg zadatka u igri i uspešno ga reše. Ovo su osnovna pravila i definišu kako završetak igre. Igrači koriste magične bodove da bacaju magije na neprijatelje, a bodove kondicije kako bi izvršavali razne fizičke poteze, sve u cilju olakšavanja borbe i generalnog prolaska kroz zadatke. Zadaci koje igrači izvršavaju mogu, a ne moraju, da se sastoje iz više delova, i igrači mogu imati više aktivnih zadataka koje rešavaju proizvoljnim redosledom. Igrači mogu pratiti progres i ciljeve zadataka u svom žurnalu, gde je tekstualno dat opis zadatka i šta je potrebno uraditi i kome se obratiti za kraj zadatka ili dodatne informacije.

Slika 1. *Izgled žurnala sa aktivnim zadacima*

Borba u igri se odvija u realnom vremenu i igrači mogu koristiti razna oružja ili magije kako bi savladali neprijatelje. Svako oružje ima svoje karakteristike kao i bodove "štete" koja se može naneti protivnicima u nekom intervalu. Prilikom borbe, za svaki udarac koji pogodi neprijatelja baca se kocka u pozadini koja određuje koliko bodova štete je naneto pogođenom liku. U borbi igrači mogu koristiti i magije koje nanose štetu, i šteta se proračunava po istom principu, obzirom da određene magije imaju sličan opis štete koju nanose. Ubijanjem protivnika, završavanjem zadataka, i otkrivanjem novih mesta i skrivenih mehanizama u igri igrači stiču bodove iskustva koje mogu potrošiti na svoje atribute i magije kako bi bili moćniji.

Glavni lik ima konačan skup fizičkih i psihičkih atributa, koji imaju vrednost 1 na početku igre. Povećavanjem fizičkih atributa igrači omogućavaju korišćenje novih fizičkih sposobnosti i poteza. Povećavanjem psihičkih atributa igrači otkrivaju nove magije koje mogu koristiti, i proširuju skup mogućih interakcija sa likovima u igri. Ovaj sistem omogućuje igračima ne samo da što bolje osmisle glavnog lika po svom ukusu, već i drugačije mogućnosti prilikom nekog narednog igranja iste igre.

Slika 2. *Fizička i psihička strana atributa glavnog junaka*

Magije igraju veliku ulogu u igri budući da je glavni lik čarobnjak. Magije predstavljaju najbolji način da igrači završe razne zadatke u igri kako bi je prešli. Po priči, magije su podeljene u četiri knjige, i svaka knjiga ima svoj skup magija koje igrač može koristiti. Ovaj način podele magija, na sličan način kao i podela atributa, omogućava igračima da naprave čarobnjaka u svom stilu, i da se fokusiraju ili maksimalno na jednu kategoriju magija, ili probaju po malo od svake kategorije, dodajući nove mogućnosti prilikom ponovnog igranja.

Slika 3. *Magije predstavljene kao ikonice u 4 kategorije za svaku knjigu*

Svaka magija se može pojačati trošenjem bodova iskustva. Magije počinju sa osnovnim nivoom 1, i kako se nivo magije povećava, tako joj se pojačava efekat, a neke magije dobijaju dodatne efekte na određenim nivoima. Obzirom da se igra izvršava u realnom vremenu i da igrači imaju širok skup magija na raspolaganju mogu u jednom trenutku, potrebno je napraviti takav sistem razmena i korišćenja magija da se tok igre ne usporava drastično. Određene igre ovog žanra taj problem rešavaju

"hotkey" metodom, gde je odabir magije ili oružja vezan za određeno dugme. U ovoj igri ovaj problem je rešen tako što igrači za svaku ruku glavnog igrača mogu da zabeleže nekoliko magija. U toku igranja, na određeno dugme otvara se prikaz koje magije su vezane za koju ruku, i mišem se lako može odabrati koja je trenutno aktivna magija za odabranu ruku. Prilikom otvaranja ovog prikaza, tok igre se usporava na ~50% (*slow motion* efekat), tako da igrači moraju brzo da odaberu šta žele da aktiviraju za koju ruku. Na ovaj način tok igre se usporava, ali ne toliko da igrači potpuno izgube osećaj imerzije tokom igranja.

Slika 4. "*Spell-Wheels" sistem odabira magija*

Oružje i drugi predmeti kao što su napici, eliksiri, hrana, odeća i oklop stoje u inventaru glavnog lika i mogu se menjati dokle god igrači nisu u borbi. Količina magija, različitih predmeta, kao i količina mogućih kombinacija ovih elemenata doprinose širokom skupu različitih načina na koje se priča u igri odvija, a samim tim se povećava i broj ponovnog igranja igre, i igrači na taj način dobijaju veću vrednost po ceni jedne igre.

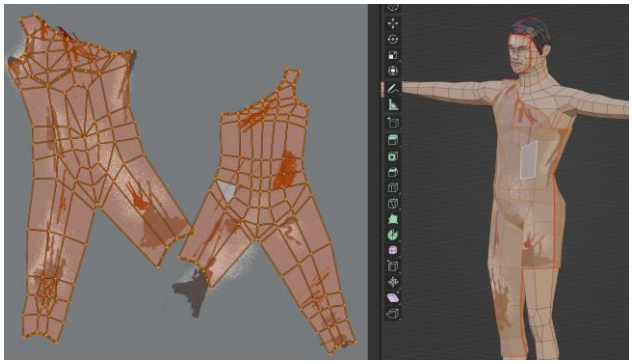
5. KORIŠĆENI ALATI PRILIKOM RAZVOJA

Glavni alat koji je korišćen u izradi igre jeste Unity pogon/okruženje za razvoj igara. Okruženja za razvoj igara su generalno napravljena za razvoj određenih žanrova ili ne podržavaju svaki stil igre, a Unity okuženje omogućava razvoj igara praktično bilo kog žanra u bilo kom stilu. Unity okruženje razvijeno je 2005. godine, sa ciljem da bude pristupačno za što više programera i timova koji žele da se bave razvojem igara [2].

Slika 5. *Izgled Unity okruženja za razvoj igara*

U zavisnosti od funkcionalnosti i mehanike koja treba da se razvije u igri, Unity ili omogućuje lak način da se te funkcionalnosti implementiraju, ili već ima implementirane mehanike koje se mogu uneti u igru u par koraka. Zbog ovih karakteristika, i zbog širokog skupa korisnika i online dokumentacije, Unity se pokazao kao idealno okruženje za razvoj jedne kompleksne igre.

Za razvoj 3D modela, pored korišćenja nekih gotovih modela preuzetih iz raznih izvora na internetu, korišćeni su Blender i Mixamo. Blender okruženje korišćeno je za konkretno modelovanje 3D likova, kao i farbanje i dodeljivanje tekstura modelima.



Slika 6. *Modelovanje i dodeljivanje teksture 3D modelu u blenderu*

Mixamo je online alat koji nudi širok skup besplatnih gotovih animacija za 3D humanoidne likove. Ovaj alat je odabran zbog lakoće korišćenja koje se svodi na upload 3D modela lika, i potom označavanja određenih tačaka u modelu koje su potrebne za generisanje animacija.

Midjourney je laboratorija koja se bavi razvojem sistema za veštačku inteligenciju, konkretno za generisanje slika na osnovu prosleđenog teksta [3]. Slike generisane sa alatom Midjourney u razvoju igre igraju ulogu privremenih resursa, da igra u fazi razvoja ne bi imala prazne resurse, odnosno osnovne resurse koji nisu detaljno definisani već postoje kao jednostavni i jednobojni oblici.

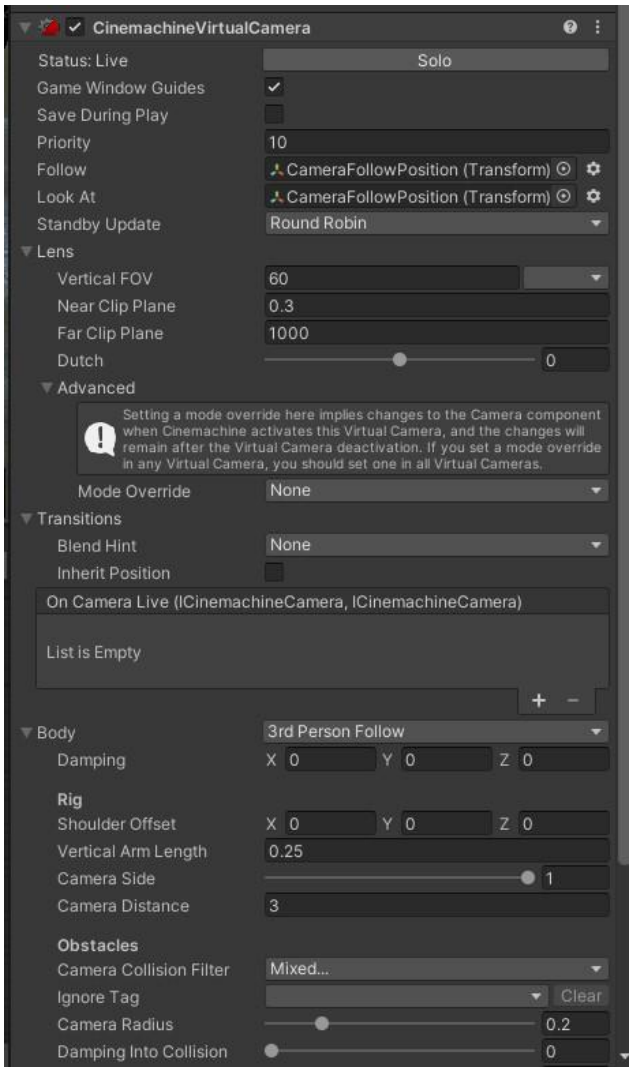
U zavisnosti od uloge u koju stupaju igrači, bitno je odabrati dobro pozicioniranje i perspektivu kamere. Neke od najčešćih perspektiva koje mogu da se nađu u igrama su:

- Izometrijska perspektiva
- Ptičja perspektiva i
- Perspektiva prvog i trećeg lica [4-5].

U igri je korišćena perspektiva trećeg lica. Kamera iz trećeg lica konstantno prati glavnog lika kroz igru, i daje pogled ne samo na svet koji lik vidi već i na samog lika. Na taj način igrači mogu videti kako se njihov lik razvija kroz svoju avanturu, kako fizički postaje širi, veći, mršaviji ili dobija ožiljke, a tako vide i razne predmete koje nose sa sobom i odeću koju imaju na sebi. Kamera je postavljena iza glavnog lika i iznad njegove glave i usmerena je u centar ekrana. U centru ekrana postavljena je bela tačka koja obeležava u šta igrači ciljaju, a oblik i boja ovog "nišana" mogu da se menjaju kroz opcije u igri. U raznim drugim igrama omogućeno je zumiranje tačke u koju se cilja (zoom-in efekat), međutim, u ovoj igri ta sposobnost nije implementirana, kako bi korišćenje magija i strela imalo veći izazov.

Implementacija kamere, zahvaljujući Unity okruženju, nije bila previše komplikovana. Najveći doprinos u implementaciji kamere imao je paket *Cinemachine,* koji

može besplatno da se prezume sa ugrađenog *Unity Package Manager*-a. Ovaj paket omogućava kreiranje komponente virtuelne kamere, koja ima veliki broj atributa za podešavanje i nameštanje idealne gamere za razne perspektive. Za običnu kameru koja se nudi u Unity okruženju, veliki broj sposobnosti se mora kodirati iznova (kao na primer: da se kamera ne sudara sa određenim objektima, ili da ne prolazi kroz zidove dok se igrači kreću po svetu). *Cinemachine* kamera, ne samo da ima ponuđene atribute koji regulišu ove i dosta drugih problema, nego su tako podešene da bez preteranog menjanja atributa već imaju ponašanje koje je potrebno za praćenje objekta u igri.



Slika 7. *Atributi Cinemachine kamere*

U razvoju igre su najbitniji bili *Follow i Look At* atributi [6]. Follow predstavlja objekat za kojim se kamera kreće, što je u ovom slučaju upravo glavni lik. Look At atribut predstavlja objekat u koji je kamera uperena, i za ovaj atribut je kreiran nevidljiv objekat koji ima svoje kordinate, postavljen tako da stoji u sredini ekrana kako bi igrači uvek gledali u sredinu ekrana tokom igre. Samo sa ova dva atributa je veliki broj funkcionalnosti kamere podešen, jer su ostali efekti kamere već nameštени. Dalji razvoj kamere sveo se na podešavanje vrednosti ostalih atributa pomoću unosa vrednosti i podešavanjem slajdera vrednosti (slika 7).

6. ZAKLJUČAK

Pored razvoja same video igre, cilj rada bio je i razvoj i konceptualizacija dobre, detaljne priče za jednu video-igru, i moguću buduću franšizu ovog žanra. Dobar deo postojećih igara započinje svoju priču od nekog trenutka koji je relevantan za ono što se dešava u samoj igri. Ovaj rad objašnjava ne samo put glavnog lika koji igrači proživaljavaju, već i njegovu prošlost kao i istoriju dešavanja u okviru sveta od samog njegovog nastanka pa do trenutnih dešavanja u životu glavnog lika.

Razvoj same igre imao je svoje praktične probleme. Međutim, Unity okruženje svodi na minimum razne problematike oko razvoja igara koje se tiču implementacije - kada se jednom naiđe na neki problem, naredno rešavanje istog ili sličnog problema može se svesti na par koraka i komandi.

U poređenju sa konkurencijom, najjači element igre "Kingdom of Souls" je njena jedinstvena priča. Gotovo nijedna mejnstrim igra nema baš toliko detaljno razvijenu priču. Određeni delovi priče su namerno ostavljeni nedovršeni ili neotkriveni, kao otvorena mogućnost za razvoj nekih drugih delova igre u ovom serijalu.

7. LITERATURA

[1]https://www.cleverism.com/gaming-industry-introduction (pristupljeno u avgustu 2024.)

[2]https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/ (pristupljeno u avgustu 2024.)

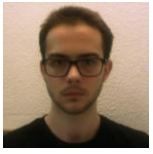
[3] https://www.midjourney.com/home (pristupljeno u avgustu 2024.)

[4]https://www.acmi.net.au/education/school-program-and-resources/game-guide-analysing-videogames-english-media/game-guide-perspective-mechanics-gameplay/ (pristupljeno u avgustu 2024.)

[5]https://kevurugames.com/blog/what-is-isometric-perspective-in-games/ (pristupljeno u avgustu 2024.)

[6]https://docs.unity3d.com/Packages/com.unity.cinemachine@2.9/manual/CinemachineVirtualCamera.html (pristupljeno u avgustu 2024.)

Kratka biografija:



Milan Đurišić rođen je u Novom Sadu 1994. god. Master rad na Fakultetu tehničkih nauka iz oblasti Proces razvoja računarskih igara odbranio je 2018.god. kontakt: milanns@live.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38
DOI: https://doi.org/10.24867/30BE12Pejic

RAZVOJ I PARALELIZACIJA ŠAHOVSKE MAŠINE SA IMPLEMENTACIJOM SERVERA I ANDROID APLIKACIJE

CHESS MACHINE DEVELOPMENT AND PARALLELIZATION WITH SERVER IMPLEMENTATION AND ANDROID APPLICATION

Dejan Pejić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu prikazan je razvoj jednostavne šahovske mašine. Opisan je Minimax algoritam za pretragu stabla svih po-teza. Izvršena je implementacija serijske i paralelne verzije algoritma s ciljem poboljšanja mašine. Mašina je integrisana u server, dok je pored toga razvijena i korisnička Android aplikacija.*

Ključne reči: *Šahovska mašina, Minimax algoritam, Paralelno programiranje, OpenMP, Klijent-server, Pthreads, Android*

Abstract – *This paper presents the development of a simple chess machine. The Minimax algorithm for searching the tree of all propositions is described. Serial and parallel versions of the algorithm were implemented with the aim of improving the machine. The machine is integrated into the server, while a user-friendly Android application has also been developed.*

Keywords: *Chess engine, Minimax algorithm, Parallel programming, OpenMP, Client-server, Pthreads, Android*

1. UVOD

Šahovske mašine su sofisticirani kompjuterski programi dizajnirani za analizu i procenu šahovskih pozicija, obezbeđujući optimalne poteze kroz napredne algoritme. Jedan od najčešće korišćenih algoritama za pretragu je Minimaks. Minimaks algoritam je fundamentalni metod u teoriji igara koji se koristi za donošenje optimalnih odluka prolaskom kroz sve moguće poteze u igri. Sa velikim dobinama pretrage, koje su poželjne radi pronalaženja boljih poteza, dolazi do potrebe za proračunom velikog stabla potencijalnih poteza, što algoritam čini računski intenzivnim. Iz tog razloga neophodne su neke optimizacije ove pretrage radi rešavanja ogromne složenosti, a jedna od mogućnosti je paralelizacija pretrage. Paralelizacija omogućava značajna poboljšanja performansi raspodelom računarskog opterećenja na više procesora

Šahovske aplikacije, koje integrišu ove mašine, omogućavaju korisnicima da igraju, analiziraju i proučavaju partije. Ove aplikacije često imaju grafičke interfejse omogućavajući korisnicima da komuniciraju sa mašinom, analiziraju različite pozicije i pregledaju igre sa uvidima iz analize mašine.

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

Moderne šahovske mašine uključuju veštačku inteligenciju i tehnike dubokog učenja kako bi dodatno poboljšali svoje performanske. Ideja ovog rada nije da se poredi sa njima, nego da se predstavi osnovni koncept razvoja i uticaj paralelizacije na performanse i snagu mašine.

U drugom poglavlju rada biće ukratko opisani osnovni koncepti u razvoju šahovske mašine sa posebnim fokusom na Minimaks algoritam i njegovu Negamaks verziju. U trećem poglavlju opisuje se paralelizacija Negamaks verzije algoritma korišćenjem OpenMP biblioteke za paralelno programiranje. U četvrtom poglavlju biće opisan razvoj servera uz pomoć Pthreads biblioteke i njemu prilagođene Android aplikacije. Na kraju, u petom poglavlju, biće predstavljeni rezultati celokupnog rada i diskusija o drugim načinima unapređenja mašine.

2. RAZVOJ MAŠINE I MINIMAX ALGORITAM

2.1. Osnovni koncepti mašine

Prilikom razvoja šahovske mašine osnova je implementacija šahovske table i figura. Šahovska tabla je reprezentovana pomoću niza kojim se modeluju 64 polja table. Slično tome i svaka figurica je predstavljena posebnim brojem, čime se omogućava njihovo razlikovanje. Nakon toga je omogućeno generisanje svih poteza, koji nije lagan proces usled raznolokih pravila kretanja figurica, a pored toga potrebno je obratiti pažnju i na specijalne poteze poput rokada, an pasana i promocije. Potez se čini 24-bitna vrednost koja predstavlja bitsko polje sa opisom raznih segmenata poteza. Što se tiče odigravanja poteza, moralo se osigurati da je svaki od njih legalan u toku partije, te omogućiti i detektovanje eventualnog ponavljanja poteza korišćenjem Zobristovog heširanja. Kako bi mašina pratila današnje standarde, implementiran je i FEN koncept.

FEN je skraćenica od Forsajt-Edvardsonove notacije i to je standardna notacija za opisivanje pozicija u šahovskoj igri. FEN je važan jer olakšava prevođenje bilo koje šahovske pozicije u jedan red teksta. Olakšava proces ponvaljanja pozicije pomoću računara i omogućava igračima da pokreću partije iz željene pozicije [1].

Pored konzolne aplikacije koja je razvijena za testiranje mašine, omogućeno je da mašina bude kompatibilna i sa UCI protokolom (engl. *Universal Chess Interface*), tako da se mašina može učitati u već razvijene šahovske aplikacije sa korisničkim interfejsom koje koriste ovaj standard.

2.1. Minimaks algoritam

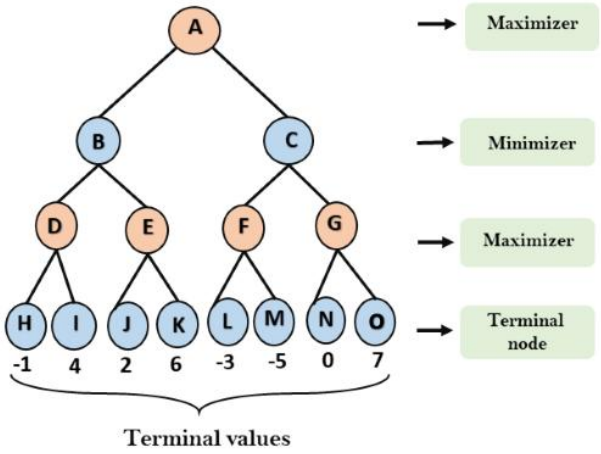
Algoritmi pretraživanja imaju tendenciju da koriste uzročno-posledični koncept – pretraga razmatra svaku moguću akciju koja joj je dostupna u datom trenutku; zatim razmatra naknadne poteze iz svakog od tih stanja, i tako redom pokušavajući da pronađe terminalna stanja koja zadovoljavaju ciljne uslove koji su joj dati. Po pronalasku terminalnog stanja ona sledi korake za koje zna da su neophodni za postizanje tog stanja.

Međutim, u takmičarskoj igri za više igrača, kada su uključeni drugi činioci koji imaju različite ciljeve na umu (obično one koji se direktno suprotstavljaju jedni drugima), stvari postaju komplikovanije. Čak i ako algoritam pretrage može da pronađe terminalno stanje, on obično ne može jednostavno da preduzme niz akcija koje će dostići to stanje, jer za svaku akciju koju algoritam preduzme ka svom cilju, protivnički igrač može preduzeti akciju koja će promeniti trenutno stanje (i to verovatno na način koji je nepovoljan za algoritam pretrage). To ne znači da je pretraga beskorisna u pronalaženju strategija za igre sa više igrača, već jednostavno zahteva dodatnu taktiku kako bi bila efektivna.

Za igre sa dva igrača, minimaks algoritam koristi činjenicu da dva igrača rade na suprotnim ciljevima i na taj način pravi pretpostavku o tome koja će buduća stanja biti dostignuta sa napretkom igre, a zatim nastavlja u skladu sa optimizovanjem svoje šanse za pobedu. Teorija iza minimaks algoritma je da njegov protivnik (engl. *minimizer*) pokušava da minimizira bilo koju vrednost koju algoritam (engl. *maximizer*) pokušava da maksimizira (shodno tome algoritam se i naziva „minimaks“). Dakle, program treba da napravi potez koji ostavlja svog protivnika u stanju da napravi najmanju štetu.

U idealnom slučaju, program istražuje svako moguće terminalno stanje igre počevši iz trenutnog stanja (kao i putanje koje je potrebno da dođe do bilo kog od njih) i svakom dodeljuje vrednosti (engl. *terminal values*). Za igre na poraz ili pobedu, kakva je i šah, postoje samo tri moguće vrednosti: pobeda, poraz ili nerešeno (kojima se često dodeljuju numeričke vrijednosti 1, -1 i 0 redom). Onda, počevši sa dna, program proračunava koji mogući ishod je najbolji za njegovog protivnika. Zatim pretpostavlja da će protvnik, ako se dostigne to stanje igre, napraviti potez koji je najbolji za njega (protivnika), a najgori za algoritam. Tako može da pretpostavi šta će njegov protivnik uraditi i da ima konkretnu predstavu o tome kakvo će biti konačno stanje igre ako se ta pretposlednja pozicija zaista i dostigne. Zatim, program može da tretira tu poziciju kao terminalnu, iako ona u suštini to nije. Ovaj postupak se može ponoviti na višem nivou, i tim redom do korena stabla. Na kraju, svakoj opciji koju algoritam trenutno ima na raspolaganju može biti dodeljena vrednost, kao da je terminalno stanje, a program jednostavno bira najvišu vrednost i preduzima tu akciju, odnosno odigrava taj potez [2].

Na slici 1 dat je primer pretrage putem minimaks algoritma. Crveni čvorovi maksimiziraju rezultat, dok je plavima čvorovima cilj da ga minimiziraju. Kada se porede svi terminali čvorovi, vidi se da je najveća vrednost 7, koja se može dostići ukoliko bi se crveni igrač pri prvom biranju odlučio za čvor C. Međutim, u tom slučaju bi se prepustio riziku da pri svom biranju plavi igrač izabere čvor F. U tom slučaju, crveni bi na kraju mogao da dođe do vrijednosti -3 ili -5, koje su nepovoljnije u odnosu na najgori izbor iz druge grupe. Zato će po minimaks algoritmu, crveni igrač izabrati čvor B, što bi ga kasnije u najgorem slučaju dovelo da bira između -1 i 4. Ako protivnik ne odigra kako se očekuje, proračun se može ponovo pokrenuti, pri čemu će rezultat biti dobar kao što je i predviđeno (ili čak i bolji).



Slika 1. *Primer pretrage po Minimaks algoritmu [3]*

Negamaks algoritam je pojednostavljena i elegantnija verzija mimaks algoritma koja koristi prednosti simetrije između maksimiziranja i minimiziranja vednosti u igrama sa dva igrača, kao što je šah. Funkcioniše pod pretpostavkom da u igri važi koncept nulte sume, odnosno da je dobitak jednog igrača gubitak drugog. Samim tim rezultat jednog igrača je negacija drugog igrača. U programu se poziva rekurzivno.

Ova verzija algoritma je iskorišćena za pretragu u šahovskoj mašini iz rada. Pošto se ne primenjuju druge optimizacije, algoritam pronalazi poteze za pristojno vreme pri najvećoj dubini pretrage jednakoj 5.

```
int negaMax( int depth ) {
    if ( depth == 0 ) return evaluate();
    int max = -oo;
    for ( all moves)  {
        score = -negaMax( depth - 1 );
        if( score > max )
            max = score;
    }
    return max;
}
```

Slika 2. *Pseudokod rekurzivnog Negamaks algoritma*

3. PARALELIZACIJA NEGAMAKS ALGORITMA

Dobra šahovska mašina ne treba samo da pronađe najbolji potez, već je poželjno da to uradi za što kraće vreme. Za dubinu pretrage jednakoj 5 prilikom testiranja rada mašine, u zavisnosti od pozicije mašini je bilo potrebno 2 – 18 sekundi da pronađe najbolji potez. To je suviše vremena, pa se iz tog razloga pretraga ograničila na konačno trajanje od 4 sekunde. Za to vreme se pri

komplikovanim pozicijama ne može pronaći adekvatan potez, pa je iz tog razloga potrebno ubrzati pretragu kako bi mašina održala svoj potencijal. S obzirom na stabloliku strukturu Negamaks algoritma, jasno je da je on odličan kandidat za paralelizaciju.

3.1. Implementacija paralelnog Negamaks algoritma upotrebom OpenMP biblioteke

Koren stabla pretrage je početna pozicija pretrage i iz te pozicije se vrši paralelizacija. Mnogi programeri posvećuju posebnu funkciju za pretraživanje korena [4], što je i ovde slučaj. Za koren pretrage implementirana je funkcija *rootNegaMax*, unutar koje se paralelno pozivaju rekurzivne funkcije *negaMax*. Pri korenu stabla formira se lista svih mogućih poteza, koji se potom do svojih terminalnih čvorova analiziraju u različitim nitima.

Paralelizacija je postignuta upotrebom modela deljene memorije pomoću OpenMP biblioteke i generalno se implementira dodavanjem *pragma* direktiva u kodu. Za paralelizaciju Negamaks pretrage iskorićena je direktiva *pragma omp parallel num_threads(tc) reduction(+: nodes) reduction(||: playableMove)*, gde parametar *tc* predstavlja broj niti koje koriste za izvršavanje pretrage, a redukcije služe da kombinuju rezultate iz više niti u jedinstvenu vrednost. Promenljiva *nodes* predstava broj posećenih terminalnih čvorova i zahteva se od strane UCI protokola, dok je *playableMoves* promenljiva tipa *bool* i daje informaciju o tome da li je tokom pretrage pronađen bar jedan legalan potez.

Prilikom paralelizacije, radi poboljšavanja performansi programa za svako jezgro napravljene su lokalne kopije podataka neophodnih za realizaciju igre i pretrage. Međutim, jedan podatak ostaje deljen, a to su informacije o najboljem potezu i njegovoj terminalnoj vrednosti. Istovremeni pristup više niti mogao bi dovesti do greške sa generisanjem poteza, pa je zato pristup ovom podatku stavljen unutar *#pragma omp critical* direktive, koja obezbeđuje da samo jedna nit izvršava određeni deo koda u isto vreme.

3.2. Rezultati poređenja serijske i paralelizovane implementacije Negamaks algoritma

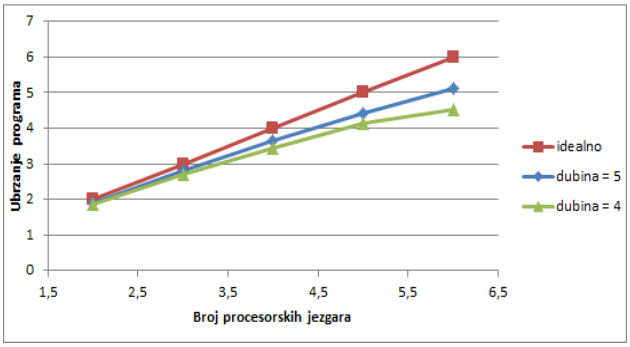
Za merenje vremena izvršavanja programa paralelizovane verzije programa korišćena je funkcija *omp_get_wtime* iz OpenMP bilioteke. To vreme se poredi sa vremenom izvršavanja serijske verzije, čime se dobija odnos koji predstavlja ubrzanje programa dobijeno paralelizacijom.

Po prirodi algoritma i načina implementacije algoritma bolji rezultati paralelizacije se očekuju pri većim dubinama pretrage. Ipak, bez drugih optimizacija, pretraga stabla već pri dubini 6 postaje neprihvatljiva za jedan šahovski program, a za veoma male dubine se dobijaju lošiji potezi. Zbog toga se kao optimalno pokazalo testiranje programa za dubine 4 i 5.

Brzina pretrage se značajno razlikuje od pozicije do pozicije. U situacijama kada je algoritam u šah–poziciji on uglavnom nema mnogo poteza na raspolaganju, pa je pri tome i serijska verzija optimalna i izvršava se u trajanju do 1 sekunde. Isto važi i za završnicu igre, kada je sa malim brojem figurica trajanje pretrage reda stotina, pa čak i desetina milisekundi. Zato je fokus na početku i sredini igre gde je trajanje dosta duže. Testiranje je izvršeno tako što je analiziran veliki broj različitih pozicija na određenom broju procesora. Za svaku poziciju se nalazi njeno ubrzanje, a potom je izračunato i srednje ubrzanje za odgovarajući broj procesora. U tabeli 1 i na slici 3 prikazani su dobijeni rezultati.

Tabela 1. *Rezultati paralelizacije algoritma*

Broj procesora	2	3	4	5	6
Ubrzanje (dubina = 4)	1,88	2,70	3,45	4,13	4,53
Ubrzanje (dubina = 5)	1,92	2,83	3,66	4,42	5,11



Slika 3. Prikaz prosečnog ubrzanja programa

5. SERVER I ANDROID APLIKACIJA

Šahovska mašina je prvobitno implementirana kao konzolna aplikacija, pri čemu je fokus bio na osnovnoj funkcionalnosti i performansama algoriritma pretrage Ipak, za potpuno korisničko iskustvo, važno je integrisati grafički korisnički interfejs (GUI). Zato je izabrana realizacija klijent-server arhitekture, trenutno u lokalnoj varijanti. Arhitektura zasnovana na serveru predstavlja dobar izbor za šahovsku mašinu, jer se klijentima omogućava daljinski pristup. S druge strane, Android aplikacija je dobra klijentska platforma zbog široke upotrebe mobilnih uređaja sa Android operativnim sistemom i svim njihovim mogućnostima.

Slika 4. *Prijava na server*

Slika 5. *Početni meni*

U server je integrisana razvijena šahovska mašina, koja samim tim čini uslugu kojima će pristupati klijenti. Server je realizovan pomoću Pthreads biblioteke kojom je omogućeno da se za svakog klijenta kreira posebna nit unutar koje se putem poruka vrši interakcija sa klijentom. Klijentska aplikacija realizovana je unutar Android

studija, pisana Java programskim jezikom i XML-ом за GUI.

Po otvaranju aplikacije od korisnika se očekuje da se poveže i unosom imena prijavi na server, pri čemu se sa serverske strane za njega kreira posebna nit. Njemu se tada otvara glavni meni pri čemu može da bira između nekoliko opcija:

- igra protiv servera, odnosno realizovane šahovske mašine nazvane Ždralin,
- igra protiv nekog drugog dostupnog igrača i
- analiza pozicije na osnovu unesenog FEN-a.

Slika 6. *Biranje figurica*

Slika 7. *Tok partije*

Pri izboru igre protiv servera korisnik može da bira boju svojih figurica. To nije moguće kada želi igrati protiv drugog korisnika, već je aplikacija implementirana tako da bele figurice ima onaj koji pošalje zahtev za igru. Zahtev se može poslati aktivnom korisniku koji u tom momentu nije u nekoj drugoj partiji ili analizi pozicije.

Tokom igre korisnik ima mogućnost da predajom prekine partiju. Dodatno, prilikom igre između dva igrača, može se ponuditi i remi, koji mora biti prihvaćen od strane drugog korisnika.

Kada dođe do šah pozicije polje napadnutog kralja bude obojeno crvenom, a ukoliko prilikom tome nastupi i mat, polje pobedničkog kralja bude obojeno žutom bojom.

Slika 8. *Kraj partije*

Slika 9. *Analiza pozicije*

Usled remija polja oba kralja budu obojena sivom bojom. Bez obzira na ishod igre, korisnik na kraju može ponuditi revanš. U revanš partiji korisnici igraju sa figuricama druge boje.

Za analizu pozicije dovoljan je samo unog pozicije u FEN formatu. Ukoliko je FEN ispravan, server analizira datu poziciju i odgovara najboljim potezom koje se označi i na samoj šahovskoj tabli.

5. ZAKLJUČAK

U ovom radu opisan je razvoj šahovske mašine i njoj prilagođene aplikacije. Pored implementacije logike igre, najvažniji segment je pretraga za najboljim potezom, što je istovremeno i vremenski najkritičniji postupak usled velikog stabla mogućih poteza.

Za pretagu stabla iskorišćena je Negamaks verzija Minimaks algoritma, koji je fundamentalni algoritam za ovu svrhu. Sa malo većim dubinama pretrage, rad algoritma počinje da traje nezahvalno dugo, što uslovljava poimenu neke optimizacije. Zato je pomoću OpenMP biblioteke primenjena paralelizacija korišćenog Negamaks algoritma u cilju dobijanja snažnije mašine, koja će za kraće vreme doći do isto kvalitetnih poteza.

Paralelizacija se pokazala uspešno sa zadovoljavajućim i korisnim ubrzanjem. Takođe, njeni rezultati pokazuju da je ubranje bliže idealnom sa većom dubinom pretrage. Ipak, za veće dubine mašina postaje nedovoljno efikasna i za serijsku i za paralelnu realizaciju. Kako bi se to poboljšalo, morale bi se primeniti i neke druge optimizacije kao što su Alfa-beta pretraga, upotreba transpozicionih tabela, primena iterativnog produbljivanje (engl. *iterative deepening*) i slično. Od njih je najznačanijia Alfa-beta pretraga koja podrazumeva smanjivanje broja terminalnih čvorova tako što se eleminišu grane, za koje se zna da ne mogu uticati na konačnu odluku.

Što se tiče Android aplikacije, pored dosadašnjeg ishoda i tu postoje velike mogućnosti poboljšanja: implementacija baze podataka, omogućavanje registracije i rangiranja igrača, postavljanje aplikacije na *Google* prodavnicu kako bi bila na raspolaganju svim korisnima itd. Uz to, ovaj klijent-server sistem bi se mogao proširiti na sledeći nivo konfigurisanjem servera da bude globalno vidljiv, kako bi mu pristupili korisnici sa raznih lokacija.

4. LITERATURA

[1] https://www.chess.com/terms/fen-chess (pristupljeno u septembru 2024)
[2] https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html (pristupljeno u septembru 2024)
[3] https://www.javatpoint.com/mini-max-algorithm-in-ai (pristupljeno u septembru 2024)
[4] https://www.chessprogramming.org/Root (pristupljeno u septembru 2024)

Kratka biografija:

Dejan Pejić rođen je u Bijeljini 2000. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Embeded sistemi i algoritmi odbranio je 2023.god. kontakt: sarajevskapejic@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.9
DOI: https://doi.org/10.24867/30BE15Milanovic

СОФТВЕРСКО РЕШЕЊЕ ЗА ОБРАДУ ПОДАТАКА О БЕРЗАМА ЗАСНОВАНО НА АРХИТЕКТУРИ *SERVERLESS*

A SOFTWARE SOLUTION FOR STOCK MARKET DATA PROCESSING BASED ON THE SERVERLESS ARCHITECTURE

Вук Милановић, *Факултет техничких наука, Нови Сад*

Област – Електротехничко и рачунарско инжењерство

Кратак садржај – *Данас, све више људи жели да улаже на берзи, али се често сусрећу са изазовима у избору и праћењу инвестиција. Постоји потреба за алатом који ће бити прилагодљив, бесплатан и који ће обухватити све основне функционалности за улагање и праћење инвестиција. Циљ овог мастер рада је да се корисницима омогући правовремено праћење промена цена акција, као и да им се пружи увид у пословање компанија од интереса. Стога, имплементирано је софтверско решење засновано на архитектури serverless, које омогућава преузимање, складиштење и визуелни приказ берзанских података. Основне функционалности система су приказ кретања цена акција компанија у реалном времену, комплетан преглед више врста финансијских извештаја, као и системско обавештавање крајњих корисника, услед наглих промена цена акција на тржишту.*

Кључне речи: *Рачунарство у облаку, архитектура serverless, обрада података у реалном времену, пакетна обрада података*

Abstract *–Today, an increasing number of people wish to invest in the stock market, but often encounter challenges in selecting and monitoring their investments. There is a need for a tool that is adaptable, free, and covers all the essential functionalities for investing and tracking investments. The goal of this master thesis is to enable users to timely monitor stock price changes and gain insights into the operations of companies of interest. Therefore, a software solution based on the serverless architecture has been implemented, allowing for the retrieval, storage, and visual display of the stock market data. The core functionalities of the system include real-time tracking of stock price movements and a comprehensive overview of various types of financial reports. Additionally, to enhance the user experience, a notification system has been implemented to alert end users of sudden stock price changes.*

Keywords: *Cloud computing, serverless architecture, real time processing, batch processing*

НАПОМЕНА:
Овај рад је проистекао из мастер рада чији ментор је био др Владимир Димитриески, ванредни професор.

1. УВОД

Историја берзе [1] датира још од почетка деветнаестог века, када је по лондонским кафетеријама почела да кружи трговина акција Британске источноиндијске компаније (енгл. *British East India Company*). Управо овај период обележио је настанак прве светске берзе у Лондону 1801. године. Крајем деветнаестог века, појавом телеграфа и телефона, комуникација се побољшала, омогућавајући трговину на већој удаљености и повезујући различита тржишта. Другу половину двадесетог века обележила је појава прве електронске берзе *NASDAQ*, што је довело до убрзања и поједностављења трговине акцијама уз помоћ рачунара. Електронске платформе су омогућиле шире учешће инвеститора и бољу анализу тржишних података, што је утицало на инвестиционе стратегије. У савременом периоду, све више људи жели да инвестира на берзи, али се често суочавају са потешкоћама у избору и праћењу сопствених инвестиција. Постоји потреба за алатом који ће бити прилагодљив, бесплатан и који ће обухвата све основне функционалности за управљање и праћење инвестиција. Циљ овог рада је да се крајњи корисници на правовремен начин информишу о променама цена акција компанија од интереса и да им се омогући адекватан увид у њихово пословање. Како би се то постигло имплементирано је савремено софтверско решење засновано на архитектури *serverless* [2], коришћењем технологија рачунарства у облаку, које омогућава преузимање, складиштење и визуелни приказ берзанских података.

2. АНАЛИЗА ПОСТОЈЕЋИХ СИСТЕМА

Кроз историју развијан је велики број система за обраду података, с различитим архитектурама и технологијама, у циљу добијања правовремених и тачних података. У наставку поглавља биће наведени описи неколико система сличних софтверском решењу које прати овај рад, као и њихови недостаци.

2.1. Блумберг терминал

Блумберг терминал је сложен систем из области финансија који нуди приступ подацима у реалном времену, као и анализу истих. Укључује историјске податке о акцијама, валутама, дериватима, а прва верзија система објављена је осамдесетих година прошлог века. Пружа напредне аналитичке алате и могућност комуникације између клијената. Иако се доказао као поуздан и ефикасан алат, Блумберг

терминал представља једно релативно старије (енгл. *legacy*) решење. С тим у вези, до пре неколико година, систем се ослањао искључиво на програмске језике Ц и Фортран, док није отпочела транзиција ка новијим језицима, као што су Јаваскрипт и Ц++. Такође, одржавање овог комплексног система било је тешко због одсуства технологија рачунарства у облаку, што је отежавало скалирање и управљање хардверским ресурсима и захтевало велике инвестиције у инфраструктуру и људске ресурсе.

2.2. *Google finance*

Компанија *Google* [3] је 2006. године представила бесплатну платформу која корисницима пружа разноврсне финансијске информације, укључујући вести и податке о тржишним трендовима. Корисници могу у реалном времену пратити цене акција светских компанија и анализирати различите временске периоде, уз приступ историјским, годишњим и кварталним извештајима. Ова платформа је послужила као мотивација за развој софтверског решења разматраног у овом раду, с намером да се постигне одређена предност у односу на постојеће решење. С тим у вези, *Google* користи свој апликативни програмски интерфејс (енгл. *API*), код ког се могу испољити одређена ограничења при изненадном повећању броја корисничких захтева, што може довести до кашњења и проблема са ажурирањем података. Поред тога, визуелизација података на *Google*-овој платформи је мање флексибилна, јер сви корисници добијају идентичну командну таблу. Насупрот томе, софтвер описан у овом раду омогућава корисницима да сами креирају или добију командну таблу по жељи, прилагођену њиховим потребама.

2.3. *Yahoo finance*

Yahoo finance [4] је платформа коју је компанија *Yahoo* лансирала 1997. године, као алат намењен за праћење и анализу финансијских података. Истиче се по својој интеграцији са широким спектром алата из области анализе финансијских тржишта (*TradingView*, *MetaTrader* и сл.). Исто тако, платформа интерно нуди вести, анализе и напредне алате, укључујући графичке приказе техничких индикатора и историјских података. У поређењу са *Google finance*-ом, *Yahoo* има више интерактивних графикона и опција за прилагођавање приказа. Такође, корисницима је омогућена пријава за различите врсте обавештења, али овај опција није бесплатна, као и већина претходно споменутих. Међутим, слична функционалност је предвиђена да буде део апликације коју прати овај рад, где би корисници били у стању да специфицирају обавештења за њима релевантне компаније, без било какве претплате.

3. АРХИТЕКТУРА СИСТЕМА

Систем је заснован на архитектури *serverless*, коришћењем Амазонових веб сервиса [5]. Одабиром оваквог приступа омогућено је динамичко скалирање, оптимизација ресурса и аутоматизација задатака система. Поред тога, систем има подразумеване карактеристике високе доступности и отпорности, које гарантује одабрани пружалац услуга у облаку.

Архитектура система је подељена у три целине тј. модула, што се може видети на слици 3.1, и ти модули ће у наставку поглавља бити описани у појединачним одељцима. Пре него што се приступи детаљном опису сваког модула појединачно, важно је споменути да сва озбиљнија софтверска решења, посебно она која складиште и анализирају велике количине података, садрже критичне компоненте које захтевају појачану безбедност и контролисану заштиту. Због тога је овај систем постављен у виртуелну приватну мрежу (енгл. *Virtual Private Cloud - VPC*), са конфигурисаним јавним и приватним подмрежама (енгл. *subnets*). Ресурси осетљиве природе, попут објектних складишта и база података, смештени су у приватну мрежу и доступни су само ресурсима који су део овог система. С друге стране, ресурси из јавне мреже, попут функција *Lambda* које преузимају податке са изворног *API*-ја, могу комуницирати са спољним светом.

3.1. Модул увлачења података у систем

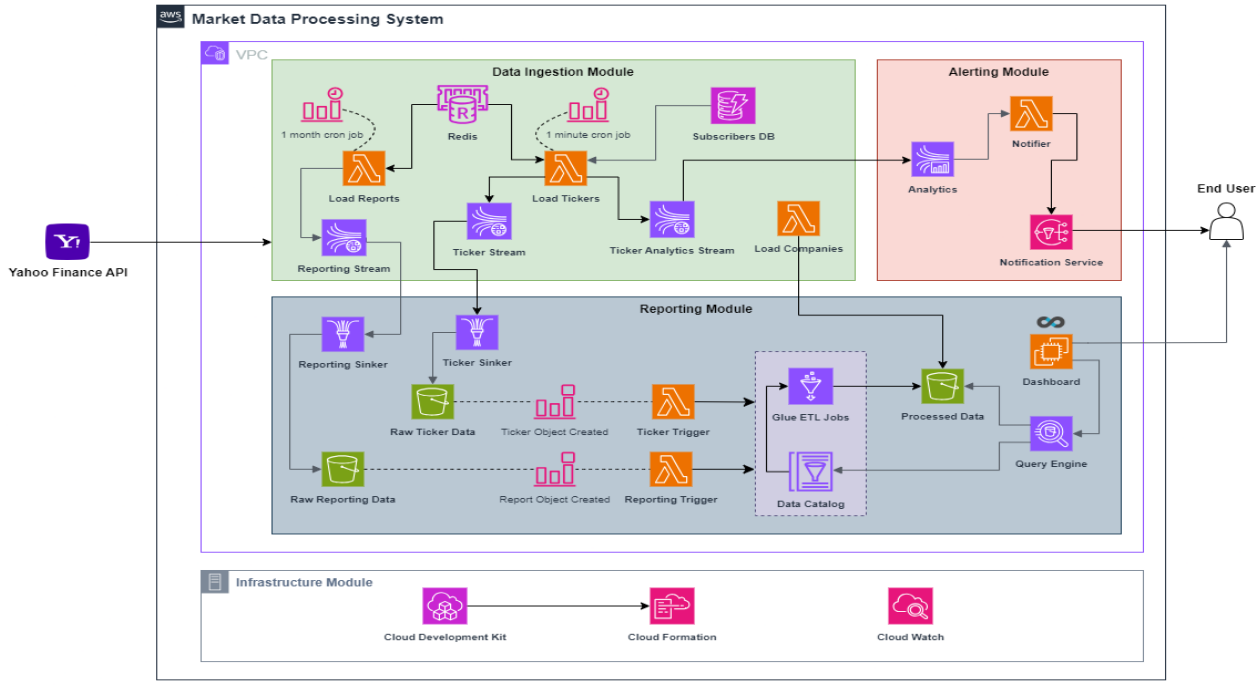
Модул увлачења, односно *Data Ingestion Module* на слици 3.1, представља срж овог софтверског решења, без којег систем не би могао да функционише. У систем се увлаче различите врсте података уз помоћ функција *Lambda*, где свака укључује скрипту написану у Пајтон програмском језику, која реализује имплементирану логику. У наставку ове секције биће описана намена скрипти за увлачење података:

1. *Load Reports* - скрипта се покреће једном месечно од стране сервиса *AWS Event Bridge* и проверава нове податке о годишњим финансијским извештајима. Разлог одабраног периода покретања лежи у томе што се финансијска година разликује од компаније до компаније.
2. *Load Tickers* - идемпотентна функција *Lambda* која се покреће сваки минут и инкрементално добавља податке о ценама акција свих праћених компанија.
3. *Load Companies* - скрипта која се користи за иницијално попуњавање или ажурирање табеле са информацијама о компанијама које систем прати.

Архитектура решења укључује две базе података с начином чувања података кључ-вредност: *Amazon ElastiCache* и *Amazon DynamoDB*. *Amazon ElastiCache* чува време последњег захтева ка *API*-ју како би систем избегао поновну обраду истих података, док *Amazon DynamoDB* чува трајне податке о корисницима. Ипак, разлог коришћења две сличне базе података лежи у пар кључних аргумената, а то су:

1. Трајност података - *Amazon DynamoDB* гарантује трајно чување података о корисницима, док *Amazon ElastiCache* не нуди трајну перзистенцију.
2. Безбедност података - податке о корисницима треба шифровати, док временски записи о увлачењу података, који су јавни, не захтевају шифровање.

Иако се на први поглед чини да је *DynamoDB* адекватно решење, због ниске латенције и напредних могућности кеширања, *ElastiCache* је боља опција за чување временских записа. Након добављања, подаци се прослеђују на даљу обраду и анализу помоћу токова



Слика 3.1 – Архитектура система

података. За потребе нашег система изабран је сервис *Amazon Kinesis* на три различита места у овом модулу:

1. *Reporting Stream* - прослеђује податке о финансијским извештајима у део система за обраду података у циљу извештавања.
2. *Ticker Stream* - прослеђује податке о тикерима у део система за обраду и приказ података крајњим корисницима.
3. *Ticker Analytics Stream* - прослеђује податке у нешто измењеном облику, у део система за потенцијално обавештавање корисника о наглим променама цена акција.

Услед могућег повећања оптерећења и времена извршавања функција *Lambda*, посебно мислећи на ону са дневним увлачењем цена акција компанија, очекује се да ће број корисника и обим података временом расти. Стога, постоје пар опција у виду будуће замене претходно споменутог сервиса, а то су *Amazon Elastic Container Service (ECS)* и *Amazon Elastic Cloud Compute (EC2)*. Генерално гледано, функције *Lambda* нису намењене за већа оптерећења, што оправдава потенцијалну надоградњу. Исто тако, када је реч о чувању података о корисницима, алтернатива за *DynamoDB* може бити било која релациона база података сервиса *Amazon RDS*.

3.2. Модул извештавања

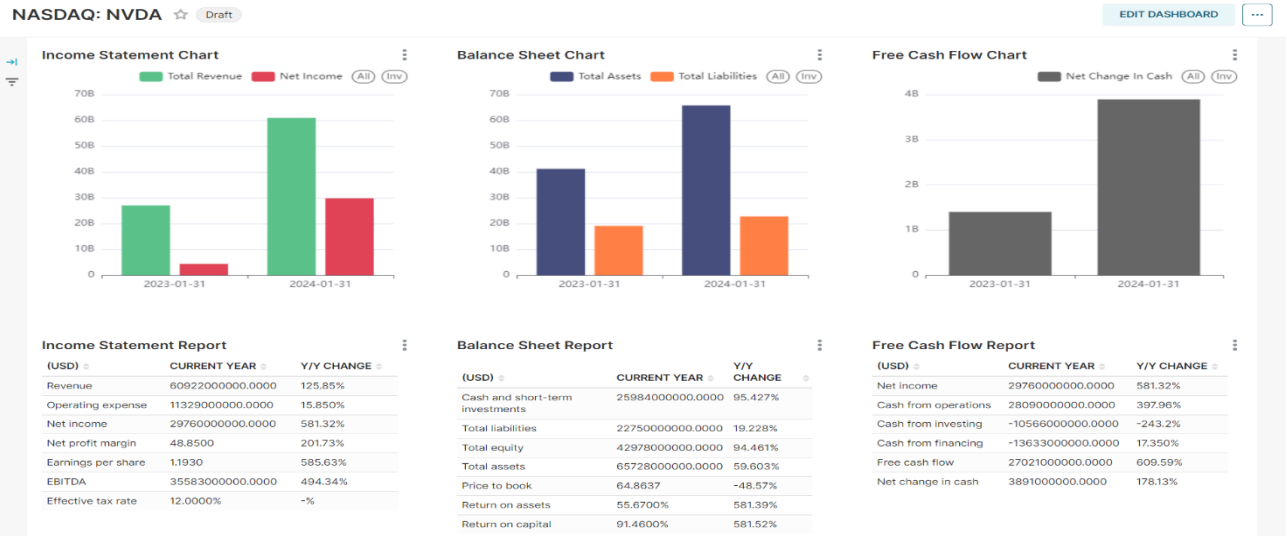
Модул извештавања, илити *Reporting Module* на слици 3.1, је најсложенији део система. Сирови подаци из токова података уливају се у сервисе *Amazon Data Firehose* који их прикупљају, трансформишу и прослеђују ка објектним складиштима. У овом модулу постоје два сервиса *Data Firehose*:

1. *Reporting Sinker* - за податке о финансијским извештајима.
2. *Ticker Sinker* - за податке о тикерима.

Крајње дестинације података из сервиса *Data Firehose* су два објектна складишта: *Amazon Simple Storage Service (S3)*. Објектна складишта су популаран начин за чување великих количина података разних формата због своје скалабилности и ниских трошкова. У нашем решењу, сирови подаци типа *JSON* се континуално складиште у језеру података (енгл. *data lake*), користећи *Yahoo Finance API* као једини извор. Сервиси који покрећу обраду сирових података су функције *Lambda*, које реагују при сваком доласку новог објекта у објектно складиште. Обрадом података управља сервис *AWS Glue* у комбинацији са *Data Catalog*-ом. Процес обраде је имплементиран помоћу Пајтон скрипти и радног оквира *Apache Spark*, који је познат по брзој и паралелној обради података. Након обраде, подаци се смештају у *S3* сервис *Processed data*, који представља складиште података (енгл. *data warehouse*) система. *AWS Athena* је изабрана као *query engine* због своје ефикасности и подршке за разне формате података. У будућности, за складиштење већих количина података, могло би се размислити о интеграцији *Amazon Redshift*-а, али за тренутне потребе би био превелик трошак. Након обраде, потребно је испроцесиране податке графички приказати. Сервис *Amazon QuickSight* је квалитетан, али и скуп као *Redshift*. Стога, алтернативу представља *Apache Superset*, који захтева посебан начин интегрисања у систем, с обзиром да није експлицитан Амазонов сервис. Између *Elastic Container Service* и *Elastic Compute Cloud*, изабрана је друга опција због једноставнијег постављања и подешавања *Superset*-а.

3.3. Модул упозоравања крајњих корисника

Модул упозоравања, односно *Alerting Module* на слици 3.1, део је нашег система који омогућава брзо обавештавање крајњих корисника о изненадним променама цена акција компанија које их интересују. Први корак је преузимање података из модула за



Слика 4.1 – Део контролне табле с годишњим финансијским извештајима

увлачење, који се затим анализирају у сервису *Amazon Kinesis Analytic*. Овај сервис, користећи *SQL* скрипту, пропушта само податке који испуњавају задати критеријум. Уколико дође до изненадног пораста или пада неке од цена акција компанија, која излази из кориснички дефинисаног опсега, подаци се прослеђују *Lambda* функцији *Notifier*. Уз посредство сервиса за слање нотификација (енгл. *Amazon Simple Notification Service*), циљном кориснику се аутоматски шаље обавештење у виду *SMS* поруке.

3.4. Инфраструктурни модул

Инфраструктурни модул на дну слике 3.1, састоји се из неколико компоненти. Системска инфраструктура је дефинисана у Пајтон програмском језику уз помоћ *AWS Cloud Development Kit*. Ова библиотека омогућава корисницима да одаберу и конфигуришу већину сервиса, након чега се инфраструктура испоручује у окружење у облаку путем *AWS Cloud Formation*, припремајући систем за рад. Осим тога, важно је нагласити улогу *Amazon Cloud Watch*-а, који омогућава надгледање (енгл. *monitoring*) кључних метрика сервиса и записивање активности (енгл. logging) функција *Lambda*. Ово омогућава правовремену реакцију на уска грла и унапређује процес дијагностике, доприносећи бржем решавању проблема у продукцији.

4. ЗАКЉУЧАК

Примарни циљ овог рада био је да се корисници система, на правовремен начини, информишу о ценама акција компанија, као и стању пословања истих. Развијен је софтверски пакет за инвеститоре и аналитичаре који омогућава праћење цена акција у реалном времену, комплетан преглед више врста финансијских извештаја, као и системско слање обавештења о наглим променама на тржишту. Систем се истиче као потпуно бесплатна услуга за праћење инвестиција, ослањајући се на Амазонове сервисе и интеграцију са *Apache Superset*-ом. Корисници могу да прилагоде командну таблу са слике 4.1, према својим потребама, што представља додатну предност у односу на друге системе. Инфраструктура је постављена коришћењем *AWS Cloud Development Kit*-а у Пајтону, уз разматрање алтернатива за будуће скалирање. Иако је ово функционално софтверско решење, постоје аспекти система који се могу додатно побољшати. Пре свега, кориснички интерфејс се може проширити додавањем функционалности за регистрацију корисника и измену личних података у складу са законом о заштити података. Такође, формати складиштених података могу бити унапређени коришћењем Parquet формата уместо JSON-а, што би побољшало перформансе при аналитичким упитима. Поред тога, било би корисно омогућити слање различитих врста обавештења, укључујући електронску пошту, као и унапређење увлачења података у различитим форматима ради флексибилнијег рада са различитим изворима података.

5. ЛИТЕРАТУРА

[1] StreetStocks. Stock Market Technology: – StockMarketLore ---streetstocks.medium.com. https://streetstocks.medium.com/stock-market-technology-tracing-its-historical-evolution stockmarketlore-a2d7bf60708e.
[2] Dr Rajan. Serverless architecture - a revolution in cloud computing. pages 88--93, 12 2018.
[3] Google Finance - Wikipedia --- en.wikipedia.org. https://en.wikipedia.org/wiki/Google_Finance.
[4] Yahoo Finance - Wikipedia --- en.wikipedia.org. https://en.wikipedia.org/wiki/Yahoo_Finance.
[5] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. Amazon Web Services. Alpha Press, 2010.

Кратка биографија:

Вук Милановић рођен је у 15. септембра 2000. године у Новом Саду. Завршио је гимназију „Исидора Секулић“ 2019. године након чега уписује Факултет техничких наука, смер Рачунарство и аутоматика. Дипломски рад је одбранио 2023. године када уписује мастер студије на студијском програму Рачунарство и аутоматика.

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.9
DOI: https://doi.org/10.24867/30BE16Topalov

МОДИФИКАЦИЈА PSO АЛГОРИТМА: ПРОМЕЊЉИВА ВЕЛИЧИНА ПОПУЛАЦИЈЕ, ХИБРИДИЗАЦИЈА СА НЕЛДЕР-МИД МЕТОДОМ И РЕШЕЊЕ ЈЕДНОГ ПЕРМУТАЦИОНОГ ПРОБЛЕМА

MODIFICATION OF THE PSO ALGORITHM: VARIABLE POPULATION SIZE, HYBRIDIZATION WITH THE NELDER-MEAD METHOD, AND SOLVING A PERMUTATION PROBLEM

Стефан Топалов, *Факултет техничких наука, Нови Сад*

Област – **ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО**

Кратак садржај – *Тема овог рада је решавање оптимизационих проблема применом алгоритама насталих модификовањем PSO алгоритма, са циљем уштеде процесорских ресурса. Први алгоритам који ће бити представљен је комбинација PSO алгоритма са промењљивом величином популације, предложеним у [1], са Нелдер-Мид алгоритмом. У другом делу рада, биће приказан један начин решавања проблема трговачког путника (енгл. Travelling Salesman Problem). Ефикасност алгоритама је нумерички потврђена.*

Кључне речи: *PSO алгоритам, Нелдер-Мид алгоритам, проблем трговачког путника*

Abstract - *This paper discusses solving optimization problems using modified v ersions o f t he PSO algorithm, aiming to reduce computational resource consumption. The first a pproach p resented combines the PSO algorithm, featuring a variable population size as proposed in [1], with the Nelder-Mead method. The second part of the paper demonstrates a solution to the Travelling Salesman Problem. The effectiveness of these algorithms is validated through numerical experiments.*

Keywords: *PSO algorithm, Nelder-Mead algorithm, Travelling Salesman Problem*

1. УВОД

У овом раду биће представљена два алгоритма за решавање типа проблема са којима се свакодневно сусрећемо, а то су оптимизациони проблеми. Први алгоритам решавања оптимизационих проблема који ће бити представљен, може се назвати „хибридни" јер комбинује модификовани *PSO* (енгл. *Particle Swarm Optimization*) са Нелдер-Мид (енгл. *Nelder-Mead*) алгоритмом. Преласком са модификованог *PSO* на Нелдер-Мид, постиже се ефикаснија претрага простора у блиској околини оптималног решења, у смислу тачности, што је поткрепљено нумеричким резултатима.

Други алгоритам намењен је решавању проблема трговачког путника (енгл. *Travelling Salesman Problem*). У циљу проналаска најкраће путање за обилазак свих потребних градова, уместо генетског алгоритма који се најчешће користи за решавање овог проблема, употребљен је модификовани *PSO* алгоритам и упоређене перформансе са генетским. Циљ имплементираних алгоритама је постизање задовољавајућих резултата претраге, уз уштеду меморијских и временских ресурса.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији је ментор била др Мирна Радовић, ванр. проф.

2. *PSO* АЛГОРИТАМ

Алгоритам оптимизације ројем честица осликава интеракцију и сарадњу међу јединкама унутар одређене популације, најчешће у потрази за храном, и тако решава оптимизационе проблеме. Утемељен је 1995. године од стране истраживача Џејмса Кенедија и Расела Еберхарта [2]. У контексту алгоритма, изједначени су појмови честице и јединке, као и рој и популације. Свака јединка представља потенцијално решење проблема, а цео систем се базира на концепту колективног учења, где се позиције и брзине сваке честице ажурирају у зависности од њихових најбољих постигнућа и постигнућа других честица у рою. Ова метода је посебно корисна за решавање комплексних оптимизационих проблема са великом просторном променљивошћу [3].

3. *PSO* АЛГОРИТАМ СА ПРОМЕНЉИВОМ ВЕЛИЧИНОМ ПОПУЛАЦИЈЕ

PSO алгоритам захтева рачунање нових позиција и вредности критеријумске функције у свакој итерацији, што може бити процесорски захтевно у случају комплексних функција. Модификације из [1] смањују број позива те функције, оптимизујући алгоритам смањењем броја честица током извршавања. Ове модификације омогућавају значајну уштеду ресурса без битног утицаја на квалитет решења, чиме се побољшава ефикасност *PSO* алгоритма.

3.1. Прва модификација

У првој модификацији алгоритма, критеријум за уклањање честица из популације представља промена вредности најбоље сопствене вредности кри-

теријума оптималности. Уколико се најбоља сопствена вредноста критеријума оптималности неке честице није променила унапред задати број итерација, а да при томе то није вредност која је уједно и најбоља вредност критеријума оптималности на нивоу целог роја, може се сматрати да честица не претражује простор довољно ефикасно и да нема велики утицај на проналазак оптималног решења, или да је запала у локални оптимум, па се она одстрањује из популације.

3.2. Друга модификација

У другој модификацији алгоритма, критеријум за елиминацију честица је међусобно еуклидско растојање. Како алгоритам напредује, честице се групишу око глобалног оптимума, често се налазећи на истој позицији, што доводи до дуплирања рачунања критеријумске функције. Уколико је растојање између честица мање од одређеног прага, честица са слабијом вредношћу критеријума се уклања, смањујући непотребно трошење ресурса без губитка на ефикасности у проналажењу оптимума.

4. ХИБРИДИЗАЦИЈА *PSO* АЛГОРИТМА

4.1. Нелдер-Мид алгоритам

Нелдер-Мид алгоритам (енгл. *Nelder-Mead*) је једноставна метода оптимизације, првобитно намењена за минимизацију функција више променљивих. Предложен је 1965. године од стране научника Џона Нелдера и Роџера Мида [4]. Алгоритам подразумева иницијализацију почетне геометријске структуре, познате као симплекс, са n+1 тачака у n-димензионалном простору. Након тога, симплекс итеративно пролази кроз низ трансформација ко-је би га померила ка оптималном решењу. Трансформације подразумевају замену темена са најлошијом вредношћу критеријума неком другом, израчунатом тачком и оне могу бити рефлексија (енгл. *reflection*), експанзија (енгл. *expansion*) и контракција (енгл. *contraction*) или, ако ове трансформације не успеју, скупљање (енгл. *shrinkage*) приликом ког се добија n нових темена.

4.2. Опис алгоритма

PSO алгоритам је веома значајан за глобалну претрагу, али успорава када се честице нађу близу оптимума. С друге стране, Нелдер-Мид алгоритам је прецизан за локалну оптимизацију. Комбинацијом ова два метода, *PSO* се прво користи за широку претрагу, а затим, након одређеног броја итераци-ја или када број честица опадне на број потребан за формирање симплекса, Нелдер-Мид преузима за фино подешавање решења. Ова стратегија балансира глобалну и локалну оптимизацију, побољшавајући брзину конвергенције и проналазак оптимума.

4.3. Нумерички резултати

У циљу истраживања перформанси претходно описаног алгоритма, извршено је тестирање на две стандардне тест функције за оптимизационе проблеме, са две и пет димензија, и резултати су упоређени са стандардним *PSO* алгоритмом и *PSO* алгоритмом са променљивом величином популације (модификовани *PSO*). За обе тест функције, алгоритми су извршени по сто пута и приказани су упросечени резултати. Приликом анализе, број честица иницијалне популације је износио 100, а број итерација је зависио од димензионалности проблема и износио је $100n$.

4.3.1. Аклијева функција

Ова тест функција дефинисана је формулом

$$f(x) = 20 + e - 20\exp(-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(-\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)), \tag{1}$$

где је n димензионалност простора. Њен глобални минимум је $f(x^*) = 0$. Иницијална популација дефинисана је на простору претраге
$-15 \leq x_i \leq 15,\ i = 1, 2, ..., n.$

У табели 1 приказани су резултати тестирања алгоритма на Аклијевој функцији са 2 димензије. Ако је са g_{best} означено пронађено оптимално решење и са m број позива функције за рачунање вредности критеријума оптималности, можемо закључити да уведени алгоритам комбинује предности класичног *PSO* алгоритма и *PSO* алгоритма са променљивом величином популације, те као резултат имамо подједнако добро решење проблема као и код класичног *PSO*, уз много мањи број позива функције за рачунање вредности критеријума, што је и био циљ увођења алгоритма описаног у раду.

Табела 1: *Анализа Аклијеве функције за две димензије*

Аклијева функција	g_{best}	m
PSO	$1.6520 \cdot 10^{-15}$	20000
Модификовани PSO	$3.5302 \cdot 10^{-6}$	4147.25
PSO + Нелдер-Мид	$1.9717 \cdot 10^{-15}$	4184.4

Из табеле 2 видимо да комбинација *PSO* и Нелдер-Мид алгоритма у потпуности оправдава разлог имплементације. Скоро у потпуности је очуван квалитет пронађеног решења уз знатно мањи број позива критеријумске функције.

Табела 2: *Анализа Аклијеве функције за пет димензија*

Аклијева функција	g_{best}	m
PSO	$3.6415 \cdot 10^{-15}$	50000
Модификовани PSO	$8.2464 \cdot 10^{-6}$	8311.09
PSO + Нелдер-Мид	$8.2796 \cdot 10^{-14}$	6904.11

4.3.2. Гриванкова функција

Гриванкова функција дата је формулом

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1, \tag{2}$$

где је n димензионалност простора. Њен глобални минимум је $f(x^*) = 0$, а поред глобалног, поседује

и велики број равномерно распоређених локалних минимума. Иницијална популација је дефинисана на простору претраге:
$-600 \leq x_i \leq 600,\ i = 1, 2, ..., n.$
Приликом тестирања на Гриванковој функцији, показало се да је за случај две и пет димензија, алгоритам потпуно надмашио и класични *PSO* и *PSO* са променљивом величином популације. У табели 3 може се видети да је алгоритам у сваком покретању пронашао тачно оптимално решење, уз значајно мањи број позива критеријумске функције у односу на класични PSO, док нам табела 4 показује да је решење скоро неприметно нарушено, уз такође значајно мањи број позива критеријумске функције.

Табела 3: *Анализа Гриванкове функције за две димензије*

Гриванкова функција	g_{best}	m
PSO	0	20000
Модификовани PSO	$2.9796 \cdot 10^{-5}$	2981.57
PSO + Нелдер-Мид	0	3116.28

Табела 4: *Анализа Гриванкове функције за пет димензија*

Гриванкова функција	g_{best}	m
PSO	$1.1102 \cdot 10^{-18}$	50000
Модификовани PSO	$3.0651 \cdot 10^{-7}$	7946.73
PSO + Нелдер-Мид	$1.7763 \cdot 10^{-17}$	7379.84

5. РЕШАВАЊЕ ПРОБЛЕМА ТРГОВАЧКОГ ПУТНИКА

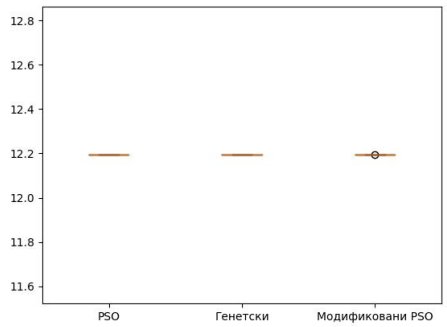
Проблем трговачког путника (енгл. *Travelling Salesman Problem - TSP*) представља један од најзначајнијих комбинаторних оптимизационих проблема. Формулисан средином 20. века [5], одговара на питање којим редоследом трговац треба да обиђе одређене градове, тако да сваки град посети тачно једном и да се врати у град из ког је и кренуо. Циљ је пронаћи најкраћу могућу руту која повезује све градове, минимизујући дужину пређеног пута.
Да би се смањио утрошак процесорских ресурса, меморије и времена, проблем пермутација ће се решавати *PSO* алгоритмом са променљивом величином популације. Сваку честицу у рою представљаће једна могућа пермутација градова. То се постиже трансформацијом координата честице у индексе сортираних координата, где сваку координату замењује њен индекс у сортираном низу од најмање до највеће вредности. Тако трансформисана позиција се затим користи за израчунавање критеријума оптималности, односно укупног пређеног пута на основу матрице растојања између градова.
Прва модификација примењена је на истоветан начин као што је раније објашњено.
У другој модификацији, адаптиран је начин мерења растојања између честица. Уместо еуклидског растојања, сада се користи разлика у редоследу обиласка градова. За сваке две честице, сабирају се апсолутне вредности разлике бројева на одговарајућим индексима пермутације градова. Ако је ово растојање веће од дефинисаног прага, честица са лошијом вредношћу критеријума оптималности се уклања из популације. Такође, ако су вредности критеријума идентичне, једна од честица се уклања из популације.

5.1. Нумерички резултати

Ради оцене ефикасности предложеног алгоритма, извршено је тестирање на проблемима са седам и десет градова. Резултати су упоређени са класичним *PSO* и генетским алгоритмом. С обзиром на присутност стохастике, сваки алгоритам је извршен сто пута, а затим је спроведена анализа резултата. Оцену перформанси представљају дијаграми који приказују распоред пронађених оптимума.
Приликом тестирања алгоритма на проблему трговачког путника са седам градова, величина иницијалне популације је износила 300 честица, а број итерација 100. У табели 5 налазе се дистанце између градова. У овом случају, модификовани *PSO* даје једно одступање, видљиво на слици 1, које се на основу анализе пронађених оптимума у сваком извршавању, може приписати грешци заокруживања. Узимајући у обзир време извршавања од 74.86 секунди за класичан, 18.51 секунд за модификовани *PSO* и 153.97 секунди за генетски алгоритам, може се закључити да је експеримент за овај случај успешан.

Табела 5: *Матрица растојања за проблем седам градова.*

	1	2	3	4	5	6	7
1	0.00	0.71	7.76	5.10	1.71	4.50	4.70
2	0.71	0.00	7.29	3.38	3.75	5.54	6.23
3	7.76	7.29	0.00	7.80	2.24	3.78	1.34
4	5.10	3.38	7.80	0.00	2.01	0.63	6.11
5	1.71	3.75	2.24	2.01	0.00	5.09	0.64
6	4.50	5.54	3.78	0.63	5.09	0.00	4.76
7	4.70	6.23	1.34	6.11	0.64	4.76	0.00

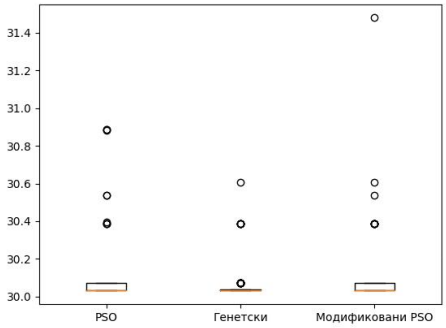


Слика 1: *Приказ пронађених оптимума за проблем седам градова.*

Са повећањем броја градова број могућих пермутација значајно расте. Из тог разлога, потребан је и већи број честица и итерација за решавање проблема десет градова, те је у овом тестирању коришћено 2000 честица и 500 итерација.
Дистанце између градова приказане су у табели 6. Очекивано, долази до појаве више одступања, што је видљиво на слици 2. На основу ових података, може се утврдити да је оптимизација генетским алгоритмом у овом случају оптимална. Али, како је један од разлога имплементације предложеног алгоритма и уштеда времена приликом извршавања, а класичан *PSO* се, у овом случају, извршавао приближно 36 минута, модификовани 5 минута, а генетски близу 8 сати, може се констатовати да постоји пуно простора за његово значајно унапређење повећањем броја честица и итерација, и постизање бољих резултата у односу на генетски алгоритам.

Табела 6: *Матрица растојања за проблем десет градова.*

	1	2	3	4	5	6	7	8	9	10
1	0.00	1.44	2.49	5.97	2.24	4.61	9.09	4.50	3.43	3.37
2	1.44	0.00	1.44	7.96	2.46	6.45	7.18	4.99	4.91	4.34
3	2.49	1.44	0.00	3.63	4.97	3.44	3.15	6.87	2.68	7.54
4	5.97	7.96	3.63	0.00	6.43	5.44	3.19	4.45	2.82	4.61
5	2.24	2.46	4.97	6.43	0.00	5.13	5.79	4.47	2.56	7.05
6	4.61	6.45	3.44	5.44	5.13	0.00	4.86	6.52	5.24	5.92
7	9.09	7.18	3.15	3.19	5.79	4.86	0.00	2.30	1.87	4.58
8	4.50	4.99	6.87	4.45	4.47	6.52	2.30	0.00	3.54	4.70
9	3.43	4.91	2.68	2.82	2.56	5.24	1.87	3.54	0.00	5.52
10	3.37	4.34	7.54	4.61	7.05	5.92	4.58	4.70	5.52	0.00



Слика 2: *Приказ пронађених оптимума за проблем десет градова.*

6. ЗАКЉУЧАК

У овом раду предложена су могућа унапређења *PSO* алгоритма за решавање оптимизационих про-блема. У уводним поглављима описане су теоријске основе самог алгоритма, а затим објашњени дета-љи имплементације модификација у [1], чији је резултат *PSO* алгоритам са променљивом величином популације, коришћен као основа за решавање оптимизационих проблема у овом раду. Извршена су два експеримента и закључци поткрепљени нумеричким резултатима.
Комбиновање *PSO* алгоритма са променљивом величином популације и Нелдер-Мид алгоритма, са циљем мањег утрошка ресурса у односу на класични *PSO* и постизања бољих резултата у односу на модификовани *PSO*, показало се као веома ефикасно и оправдало је разлог имплементације.
Други експеримент, решавање проблема трговачког путника применом *PSO* алгоритма са променљивом величином популације, такође се показао као успешан, уз могућности напредовања за проблем већег броја градова.
У случају да је неко од добијених решења незадовољавајуће, и потребно је прецизније коначно решење, то се може постићи повећањем броја честица иницијалне популације, повећањем броја итерација или корекцијом осталих параметара коришћених приликом тестирања алгоритама.
Могућности иновација на пољу оптимизационих алгоритама остају неограничене.

7. Литература

[1] Стефан Топалов, *„Модификовање PSO алгоритма променљивом величином популације”*, Факултет техничких наука, Универзитет у Новом Саду, 2023.

[2] J. Kennedy and R. Eberhart, *„Particle swarm optimization”*, Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[3] Жељко Кановић, Зоран Јеличић, Милан Рапаић *„Еволутивни оптимизациони алгоритми у инжењерској пракси”*, Факултет техничких наука, Нови Сад, 2017.

[4] Nelder, John A. and Roger Mead. *“A Simplex Method for Function Minimization.”* Comput. J. 7 (1965): 308-313.

[5] Dantzig, G. B, Fulkerson, R, and Johnson, S. M. (1954). *„Solution of a large-scale traveling-salesman problem”*, Operations Research, 2(4), 393–410.

Кратка биографија:

Стефан Топалов рођен је у Новом Саду 2000. године. Мастер рад на Факултету техничких наука из области Електротехнике и рачунарства одбранио је 2024. године.
Контакт: topalov.stefan.ns@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/30BE18Vukic

NAPADI VEZANI ZA REDOSLED I TEMPIRANJE NA L2 BLOCKCHAIN MREŽAMA

TIMING AND SEQUENCE ATTACKS ON L2 BLOCKCHAIN NETWORKS

Nikola Vukić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu će biti predstavljene L2 blockhcain mreže, napadi vezani za redosled transakcija, redosled instrukcija u transakciji, te napadi vezani za tempiranje na pomenutim mrežama. Rad uključuje i implementaciju jednostavnog pametnog ugovora za bolji prikaz efekata nekih od navedenih napada. Na kraju, dati su načini odbrambenih mehanizama za potpunu zaštitu, ili smanjivanje efektivnosti napada.*

Ključne reči: *blokčejn, problem skalabilnost, bezbednost*

Abstract – *This paper will present L2 blockchain networks, attacks based on transaction sequence, instruction sequence inside a transaction, and timing based attacks on said networks. The paper includes implementation of a simple smart contract to better show the effects of some of these attacks. Finally, the defense mechanisms, to completely mitigate the attacks, or reduce their effectiveness, are given.*

Keywords: *blockchain, scalability problem, security*

1. UVOD

Glavna primena blokčejna [1] (engl. *blockchain*) jeste u takozvanim decentralizovanim finansijama. Cilj osnivača jeste da se blokčejn koristi kao preferirani metod plaćanja opšte populacije u svakodnevnom životu.

Ova ideja ima dva problema, problem sigurnosti i problem skalabilnosti. Problem sigurnosti jeste taj što je logika na blokčejnu kontrolisana od strane pametnih ugovora, i ukoliko u njima postoji neka greška, sredstva mogu biti bespovratno ukradena. Problem skalabilnosti se ogleda u tome da je blokčejn mreža spora, te podržava svega oko 15 transakcija po sekundi. Ovo nije ni približno dovoljno da bi bila opšteprihvaćena, jer nije sposobna da podrži saobraćaj koji bi došao sa tolikim povećanjem broja korisnika mreže.

Još jedan dodatan problem jeste taj što skalabilnost na blokčjenu nije moguće povećavati u nedogled, jer se time smanjuje bezbednost sistema.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Veljko Petrović, docent.

[1]Postoji više vrsta i implementacija blokčejna, u daljem tekstu, termin *blokčejn* označavaće *Etirijum* (engl. *Ethereum) blokčejn mrežu*.

Zato što je blokčejn mrežu teško skalirati, odnosno povećati protok saobraćaja koji može da podrži, nastala su rešenja drugog sloja (engl. *Layer 2*, u daljem tekstu *L2*). Od svih vrste rešenja drugog sloja, najprimenjenija su rešenja zasnovana na namotavanjima (engl. *rollup*).

Osnovna ideja ovih rešenja jeste da bezbednost naslede od osnovne verzije blokčejna, takozvanog prvog sloja, a da povećaju skalabilnost tako što se transakcije vrše van samog prvog sloja, i s vremena na vreme, se više transakcija postavi na prvi sloj. Prvi sloj onda potvrdi njihovu ispravnost.

Kako uspešni napadi donose velike profite napadačima, veliki napori se ulažu da bi se ovi napadi analizirali, i da bi se stvorili odgovarajući bezbednosni mehanizmi. Neki od napada su omogućeni načinom funkcionisanja mreže, neki od njih su rezultat poslovne logike, a neki su jednostavno propusti pri programiranju pametnih ugovora. Napadi obrađeni u ovom radu pokrivaju sve tri od navedenih kategorija.

2. L2 BLOKCHAIN MREŽE

Kada se količina saobraćaja na mreži poveća, da bi se smanjilo zagušenje, transakcije postaju skuplje. Algoritam računanja cene transakcije je podešen da protok na mreži drži na oko pola njenog kapaciteta. Ako je mreža, zbog male skalabilnosti, zagušena, transakcije postaju izuzetno skupe, a samim time i nepristupačne za korisnike. Iz ovoga razloga bilo je neophodno da se napravi rešenje za problem manjka skalabilnosti. Iz ovoga su nastale L2 mreže [1].

Kako je već ranije pomenuto, najzastupljeniji tip L2 mreže jesu namotavanja. Postoje dve vrste namotavanja, optimistična namotavanja, i namotavanja nultog znanja. Obe vrste su zastupljene i imaju svoje prednosti i mane. Ono što im je zajedničko, da obe izvršavaju transakcije van prvog sloja, a na prvi sloj postavljaju rezultat tih transakcija. Transakcije su grupisane u pakete (engl. *batches*), i na ovaj način se na stotine i hiljade transakcija postavi kao jedna transakcija, čineći paketirane transakcije jeftinijim.

2.1. Optimistična namotavanja

Naziv optimistična namotavanja (engl. *optimistic rollups*) potiče iz činjenice da kada se paket transakcija sa drugog sloja pošalje na prvi sloj, podrazumeva se da su sve transakcije validne. Da bi se dokazalo suprotno, neophodno je priložiti dokaz nevalidnosti. Dokaz se prilaže u takozvanom periodu izazivanja (engl. *challenge period*), koji ima varijabilne dužine trajanja, ali obično traje oko 7 dana.

Prednost optimističnih namotavanja jeste taj što su zahtevi za računarskom moći manji nego u slučaju namotavanja nultog znanja, ali je mana to što se na prvi sloj moraju postavljati sve transakcije, da bi se mogla utvrditi njihova validnost. Još jedna mana jeste ta što da bi transakcije bile konačne mora se sačekati da prođe period izazivanja.

2.2. Namotavanja nultog znanja

Namotavanja nultog znanja (engl. *zero-knowledge rollups*) su dobila naziv zbog toga što sekvenceri koriste tehnike dokaza nultog znanja da bi dokazali ispravnost transakcija u pakteima koje postavljaju na prvi sloj. Dakle, ni za jedan paket se ne smatra da je validan, nego se validnost mora dokazati pri postavljanju svakog paketa.

Prednost optimističih namotavanja je ta što su transakcije brže, zbog nepostojanja perioda izazivanja. Mana im je to što dokazi nultog znanja zahtevaju više računarske moći, što može da umanji broj učesnika.

3. PREGLED LITERATURE

Kako je glavna primena blokčejna u decentralizovanim finansijama, bezbednosni propusti neretko dovode do trajnog gubitka sredstava. Iz ovog razloga, postoje radovi [3-5] koji skreću pažnju na česte tipove napada na blokčejnu. Uprkos tome, svake godine, velike količine digitalnog novca biva ukradeno.

Sa druge strane, problemom unapređenja L2 mreža se bavi čitava zajednica, svako može da priloži predlog za poboljšanje Etirijuma (engl. *Ethereum Improvement Proposal*, u daljem tekstu *EIP*), koji dobije jedinstveni identifikator u formi broja.

Iako postoji više EIP-a koji su se bavili problemom unapređenja mreže, dva najbitnija su EIP-1559 [6] i EIP-4844 [7].

EIP-1559 je uveo model plaćanja za transakcije kakav danas postoji na mreži, čime je stabilizovao cene transakcija, što značajno pomaže L2 mrežama kada treba da se sinhronizuju sa prvim slojem, odnosno postave pakete.

EIP-4844 je predlog koji je direktno vezan za poboljšanje skalabilnosti L2 mreža i čijim usvajanjem je zajednica prihvatila mreže bazirane na namotavanjima kao budućnost Etirijuma. Osnovna ideja jeste da se podaci koji su neophodni za dokazivanje validnosti čuvaju samo određeni vremenski period, koji je dovoljan da bi se dokazala validnost, a onda se brišu.

4. NAPADI VEZANI ZA REDOSLED I TEMPIRANJE

Napadi koji su obrađeni u ovom radu, posledica su redosleda transakcija u bloku, redosleda instrukcija u transakciji, ili su vezani za tempiranje, odnosno mogućnost manipulacije vremenskim otiscima i premalim intervalima proteklog vremena između zahteva.

4.1 Napadi zasnovani na MEV-u

Maksimalna ekstraktibilna vrednost, odnosno MEV, jeste sva vrednost koju validator može da dobije uvrštavanjem bloka u lanac, pored standardne nagrade koja mu sledi za ovaj čin.

Pre nego što budu uvršćene u blok, transakcije dospevaju u bazen transakcija (engl. *mempool*) u kome borave dok ih neko od validatora ne uvrsti u svoj blok. Za vreme boravka u bazenu, svi detalji transakcije su potpuno javni. Ovo stvara priliku za napadača da pronađe transakciju koju može da napadne.

Napadi zasnovani na MEV-u pripadaju tipu napada koji su vezani za redosled transakcija u bloku.

Tri osnovna tipa napada zasnovana na MEV-u jesu:

- **Prednjačenje** (engl. *frontrunning*) – napadač umeće svoju transakciju ispred one koju napada.
- **Zadnjačenje** (engl. *backrunning*) – napadač umeće svoju transakciju iza one koju napada.
- **Sendvič napad** (engl. *sandwich-attack*) – napadač umeće dve transakcije, jednu ispred, i jednu iza one koju napada. Predstavlja kombinaciju prednjačenja i zadnjačenja.

4.2. Napad na osnovu vremenske zavisnosti

Napadi na osnovu vremenske zavisnosti (engl. *timestamp dependence*) zasnivaju se na činjenici da sekvenceri mogu, u većoj ili manjoj meri, zavisno od konkretne L2 mreže, manipulisati vremenskim otiskom.

Iz ovog razloga ukoliko se vremenska zavisnost koristi kao izvor nasumičnih vrednosti, onda te vrednosti uopšte nisu nasumične nego su u potpunosti konotrlisane od strane sekvencera.

4.3. Napad ponovnim ulaskom

Napad ponovnim ulaskom (engl. *reentrancy attack*) je posledica neispravne sekvence instrukcija u transakciji. Ukoliko se eskterni poziv uputi ka određenoj adresi, pre nego što se ažurira stanje ugovora, onda, ukoliko je ova adresa pametni ugovor, može da iskoristi ovu činjenicu da izvrši napad ponovnim ulaskom.

Postoji više vrsta napada ponovnim ulaskom:

- **Ponovni ulazak u jednu funkciju** (engl. *single function reentrancy*) – najprostiji tip napada, napadaču se uputi poziv iz funkcije, on uputi poziv ka istoj funkciji.
- **Međufunkcijski ponovni ulazak** (engl. *cross-function reentrancy*) – napadaču se uputi poziv iz funkcije, on uputi poziv ka drugoj funkciji.
- **Međuugovorni ponovni ulazak** (engl. *cross-contract reentrancy*) – više ugovora deli stanje, ranjivi ugovor uputi poziv, pre nego što ažurira stanje, napadač pozove funkciju drugog ugovora. ugovora koji zahtevaju previše procesorkse moći, jer bi cena izvršavanja bila prevelika.
- **Međulančani ponovni ulazak** (engl. *cross-chain reentrancy*) – iako manje uobičajan, i dalje moguć. Slučaj u kome su ranjivi ugovori postavljeni na različite mreže.
- **Ponovni ulazak bez modifikacije stanja** (engl. *read-only reentrancy*) – poseban slučaj međuugovornog ulaska gde napadač uputi poziv

ugovoru, koji ne ažurira stanje pre upućivanja poziva napadaču, napadač napadne drugi ugovor, koji čita stanje iz prvog ugovora, koje nije ažurirano, i bude žrtva napada.

4.4 Napad onemogućavanja pružanja usluga

Efekat napada onemogućavanja pružanja usluga može biti privremen ili trajan. Trajan znači da ugovor više ne može da pruža neku uslugu, ili više njih, korisnicima. Privremen efekat onemogućavanja pružanja usluga nastaje kao rezultat premalo vremena između korisničkih zahteva.

Naime, ukoliko postoji uslov, nametnut stanjem ugovora, da bi se neka funkcionalnost započela, a ugovor još nije promenjen u to stanje, funkcionalnost se privremeno ne može izvršiti. Samim time se ne može pružiti usluga korisniku.

Nekada se ne može mnogo učiniti po pitanju privremenog onemogućavanja pružanja usluga, jer postoje situacije u kojima je spora promena stanja nametnuta logikom funkcionisanja ugovora.

5. PAMETNI UGOVOR

Napisan je pametni ugovor koji opisuje rad lutrije. Ugovor ima vlasnika, koji dobija 5% dobitka u svakom kolu, dok pobedniku ide ostatak. Svi učesnici uplaćuju jednaku cenu učešća. Sabiranjem svih cena učešća dobija se ukupni dobitak za kolo. Kolo traje minimalno sedam dana. Nakon sedam dana, pobednik se može izvući.

Dok ovaj period ne prođe, učesnik može da se povuče iz kola, čime dobija nazad uložena sredstva.

Pobednik se bira na osnovu rednog broja učesnika. Redni broj učesnika se bira kao ostatak pri deljenju vremenskog otiska bloka transakcije i broja učesnika u kolu. Nakon izvlačenja pobednika, pobednik se isplaćuje. Svi učesnici kola se brišu iz skupa učesnika, da bi se lutrija pripremila za naredno kolo. Onog momenta kada su svi dosadašnji učesnici obrisani, kreće se u novo kolo.

6. REZULTATI EKSPERIMENTA

Da bi se uporedile cene, radi pokazivanja prednosti cena na L2 mrežama, ugovor je bio postavljen na mrežu prvog, i drugog sloja.

Cena postavljanja na prvi sloj je bila, u protivvrednosti evra, 90 evra, dok je za drugi sloj bila 13 centi.

Pored ovoga, na ugovor su bili upućeni i napadi da bi se pokazali njihovi efekti. Na ugovor su bili upućeni:

- **Napad na osnovu vremenske zavisnosti** – simulira se da situacija u kojoj validator odabere vremenski otisak transakcije i bude pobednik.
- **Napad ponovnim ulaskom u jednu funkciju** – bilo koji učesnik povuče sredstva koja je dao kao cenu učešća i napadne ugovor.
- **Napad onemogućavanja pružanja usluga –** veliki broj učesnika onemogući njihovo čišćenje između dva kola.

TABELA 1: UPUĆENI NAPADI

Napad	Rezultat	Uzrok ranjivosti
Na osnovu vremenske zavisnosti	Deterministično izvlačenje pobednika, namešten ishod lutrije	Način funkcionisanja vremenskih otisaka na mrežnom nivou i način biranja nasumičnog broja
Ponovnim ulaskom	Ukraden sveukupan dobitak za kolo	Pogrešan raspored instrukcija u okviru transakcije
Onemogućavanja pružanja usluga	Onemogućavanje izvlačenja pobednika i nastavka funkcionanisanja ugovora	Način odbrane mreže od onemogučavanja pružanja usluga i logika sistema koji ugovor opisuje

7. ODBRANE

Prikazano je na koji način je neophodno izmeniti implementirani pametni ugovor da bi se zaštitilo, koliko je to moguće, od napada koji su bili upućeni.

7.1 Odbrane od napada zasnovanih na MEV-u

Kako implementirani pametni ugovor za lutriju nije bio podložan njima, prikazane su mrežne modifikacije koje implementiraju neke L2 mreže da bi se zaštitile od ovog tipa napada. Pored ovoga, dati su mehanizmi koji se mogu primeniti za zaštitu u određenim situacijama na mrežama koje ne implementiraju nikakve posebne mehanizme. Mrežne modifikacije postoje na L2 mrežama:

- **Arbitrum** – sekvencer nema mogućnost da menja redosled transakcija u bloku. Transakcije se isključivo redaju prema vremenskim otiscima pristizanja [8].
- **Optimism –** bazen transakcija je privatan, što onemogućava nadgledanje transakcija pre nego što budu uvrštene u blok [9].

Što se tiče ostalih mehanizama zaštite, opisana je potvrdi-otkrij šema, koja nudi odbranu u slučaju anonimnih glasanja ili aukcija, te mehanizam lomljenja transakcija u slučaju razmena na menjačnicama.

7.2 Odbrana od napada na osnovu vremenske zavisnosti

Ukoliko je neophodno generisati nasumičan broj, onda se za to ne može koristiti ni vremenski otisak, ni bilo kakva informacija vezana za blok, kojom validator može da manipuliše.

Sa druge strane, ukoliko je neophodno kontrolisati vreme trajanja nekog stanja, recimo trajanja glasanja, ili trajanja aukcije, onda je moguće da se, umesto samog vremenskog otiska, koristi broj blokova.

Blokovi se na različitim mrežama drugog sloja proizvode različitim brzinama, i shodno mreži na kojoj se postavi

ugovor, moguće je proračunati željeni broj blokova koji želimo da traje neko stanje.

7.3 Odbrana od napada ponovnim ulaskom

Za odbranu od napada ponovnim ulaskom su predložena dva mehanizma:

- **PEI šablon** – nastao kao skraćenica od principa redanja instrukcija: provere, efekti, interakcije (engl. *checks*, *effects*, *interactions*, to jest *CEI*). Podrazumeva prvobitnu proveru validnosti zahteva za transakciju. Potom, prvo se promeni stanje ugovora, odnosno izvrše se efekti transakcije. Tek na kraju se upućuju eksterni pozivi ka drugim adresama.

- **Modifikator za zaključavanje funkcije** – u ugovor se dodaje logika kojom se onemogućuje da isti korisnik uputi još jedan zahtev ka ugovoru, pre nego što se prethodni zahtev ne izvrši.

7.4 Odbrana od napada onemogućavanja pružanja usluga

Kao što je ranije pomenuto, trajno onemogućavanje pružanja usluga svedeno je na privremeno mehanizmom paketiranja. Osnovna ideja je da se ograniči broj iteracija u petljama i na taj način se onemogući prebacivanje maksimalne cene gasa koju transakcija može da ima.

Na ovaj način se izbegava da se neka transakcija ne može izvršiti nikada. Međutim, iako je razbijena na više transakcija, od kojih se svaka može izvršiti, dok se sve ne izvrše, korisnici neće moći biti usluženi.

8. ZAKLJUČAK

U radu su predstavljene osnove blokčejna, i objašnjeno je zašto su neophodne L2 blokčejn mreže. Objašnjene su osnove rada L2 mreža zasnovanih na namotavanjima.

Zatim, analizirani su napadi koji su vezani za redosled, kako transakcija u bloku, tako i redosleda instrukcija u transakcijama. Pored ovoga, predstavljeni su i napadi vezani za tempiranje: korišćenje vremenskih otisaka blokova, i premalo vremena između zahteva ka ugovorima.

Za napade koji su navedeni predstavljeni su mehanizmi odbrane koji, zavisno od prirode logike pametnog ugovora, mogu da umanje ili negiraju efekat napada.

9. LITERATURA

[1] Ethereum Foundation. Layer 2. https://ethereum.org/en/layer-2/ (pristupljeno u septembru 2024.)

[2] Merkleovo stablo https://en.wikipedia.org/wiki/Merkle_tree/ (pristupljeno u septembru 2024.)

[3] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. 2020. A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses. ACM Comput. Surv. 53, 3, Article 67 (May 2021), 43 pages. https://doi.org/10.1145/3391195.

[4] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur and H. -N. Lee, "Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract," in ***IEEE Access***, vol. 10, pp. 6605-6621, 2022, doi: 10.1109/ACCESS.2021.3140091.

[5] P. Daian ***et al***., "Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability," ***2020 IEEE Symposium on Security and Privacy (SP)***, San Francisco, CA, USA, 2020, pp. 910-927, doi: 10.1109/SP40000.2020.00040.

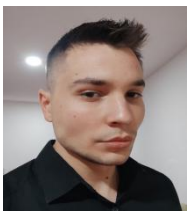
[6] Rick Dudley (@AFDudley) Matthew Slipper (@mslipper)-Ian Norden (@i-nor-den) Abdelhamid Bakhta (@abdelhamidbakhta) Vitalik Buterin (@vbuterin), Eric Conner (@econoar). "eip-1559: Fee market change for eth 1.0 chain," ethereum improvement proposals, no. 1559. https://eips.ethereum.org/EIPS/eip-1559, April 2019. (pristupljeno u septembru 2024.)

[7] Diederik Loerakker (@protolambda) George Kadianakis (@asn-d6) Matt Garnett (@lightclient) Mofi Taiwo (@Inphi) Ansgar Dietrichs (@adietrichs) Vitalik Buterin (@vbuterin), Dankrad Feist (@dankrad). "eip-4844: Shard blob transactions," ethereum improvement proposals, no. 4844. https://eips.ethereum.org/EIPS/eip-4844, February 2022.
(pristupljeno u septembru 2024.)

[8] Arbitrum. Gas and fees: Tips in l2. https://docs.arbitrum.io/how-arbitrum-works/gas-fees#tips-in-l2
(pristupljeno u septembru 2024.)

[9] Optimism. Rollup protocol overview - block production. https://docs.optimism.io/stack/protocol/rollup/overview#block-production
(pristupljeno u septembru 2024.)

Kratka biografija:



Nikola Vukić rođen je u Tesliću 2000. godine. Osnovnu školu i gimnaziju završio u Doboju. Diplomski rad na Fakultetu tehničkih nauka u Novom Sadu odbranio je 2023. god.

kontakt: nikola.vukic@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/30BE20Stojic

ODREĐIVANJE VRSTE TUMORA MOZGA NA OSNOVU MRI SNIMAKA ENDOKRANIJUMA PRIMENOM METODA VEŠTAČKE INTELIGENCIJE

DETERMINING THE TYPE OF BRAIN TUMOR BASED ON MRI IMAGES OF THE ENDOCRANIUM USING ARTIFICIAL INTELLIGENCE METHODS

Aleksa Stojić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – Elektrotehnika i računarstvo

Kratak sadržaj – *Cilj rada je brža i pouzdanija klasifikacija vrste tumora korišćenjem metoda veštačke inteligencije. Rešavanje problema klasifikacije slika MRI snimaka na vrste tumora. Rešenje je implementirano finim podešavanjem parametara na tri pretrenirana modela, Resnet50, DenseNet121 i EfficientNet-B3. Najbolji rezultat dalo je fino podešavanje parametara na pretreniranom modelu EfficientNet-B3, sa tačnošću od 98%..*

Ključne reči: *Klasifikacija slika, Računarska vizija, tumor mozga, fino podešavanje parametara pretreniranog modela*

Abstract – *The goal of the work is a faster and more reliable classification of the type of tumor using artificial intelligence methods. Solving the problem of classifying MRI images into tumor types. The solution was implemented by fine-tuning parameters on three pre-trained models, Resnet50, DenseNet121 and EfficientNet-B3. The best result was obtained by fine-tuning the parameters on the pre-trained EfficientNet-B3 model, with an accuracy of 98%..*

Keywords: *Image classification, computer vision, brain tumor, fine tuning of pre-trained model*

1. UVOD

Tumor mozga predstavlja značajan izazov za zdravstvene sisteme širom sveta. Prema podacima Američke asocijacije za tumor mozga, približno 700.000 ljudi u Sjedinjenim Državama trenutno živi sa primarnim tumorom mozga, dok se godišnje očekuje se oko 84.000 novih slučajeva. Svetska zdravstvena organizacija procenjuje da tumori mozga čine oko 1,6% svih kancera širom sveta. Stopa smrtnosti je podjednako zabrinjavajuća, s obzirom da su tumori mozga vodeći uzrok smrtnosti od kancera kod osoba starih od 15 do 39 godina. Stopa preživljavanja varira u zavisnosti od vrste tumora, mesta na kom se nalazi i terapije, ali prosečna petogodišnja stopa preživljavanja za maligne tumore mozga je približno 36% [1] .

S medicinskog stanovišta, tumori mozga su kategorisani u različite tipove na osnovu načina nastanka i ponašanja. Primarni tipovi između ostalih uključuju gliome, meningiome i tumore hipofize. Za ovaj rad naznačajniji su gliome, meningiome i tumori hipofize koji se nalaze na slikama u korišćenom setu podataka. Gliomi su najčešći primarni tumori mozga i potiču od glijalnih ćelija koje pružaju podršku i ishranu neuronima. Mogu biti od niskog do visokog stepena malignosti. Meningiomi su drugi po učestalosti tumori mozga. Nastaju iz meninge, zaštitnih membrana koje pokrivaju mozak i kičmenu moždinu. Uglavnom su benigni, ali u nekim slučajevima mogu postati maligni i zahtevati hirurško uklanjanje. Tumori hipofize nastaju u hipofizi, maloj žlezdi smeštenoj u osnovi mozga koja reguliše mnoge važne hormonske funkcije. Većina ovih tumora je benigna, ali čak i benigni tumori mogu izazvati značajne zdravstvene probleme zbog svog uticaja na hormonalnu ravnotežu i pritiska koji mogu vršiti na okolna tkiva mozga. Razumevanje ovih različitih tipova tumora, njihovog ponašanja, i njihovog uticaja na pacijente je ključno za efikasno dijagnostikovanje i lečenje. Rana detekcija i precizna klasifikacija mogu značajno poboljšati ishode za pacijente, omogućavajući bolju personalizaciju terapije i dugoročnu kontrolu bolesti[2][3][4].

Dijagnostika tumora mozga uključuje i invazivne i neinvazivne metode. Neinvazivne tehnike kao što su magnetna rezonanca (MRI) i kompjuterska tomografija (CT) često se koriste za početnu detekciju i praćenje. Međutim, konačna dijagnoza često zahteva invazivne postupke kao što je biopsija, tokom koje se uzimaju uzorci tkiva za ispitivanje kancerogenih ćelija. Proces može biti vremenski zahtevan i složen, što često dovodi do odlaganja u početku tretmana. Pored tog problema, postoji značajan nedostatak medicinskih stručnjaka u nekim regijama. Na primer, u Srbiji, odnos broja neurologa na broj stanovnika je relativno nizak u poređenju sa evropskim standardima, što doprinosi odlaganjima u dijagnozi. U Sjedinjenim Američkim Državama, iako je pristup specijalistima generalno lakši, i dalje postoje razlike između različitih država i ruralnih područja [5][6].

Iz prethodno navedenih razloga, cilj ovog rada je istraživanje i procena različitih pristupa u klasifikaciji tumora mozga korišćenjem metoda mašinskog učenja. Koristeći napredne tehnike u mašinskom učenju i analizi slika, ovo istraživanje ima za cilj da pruži uvid u poboljšanje tačnosti i efikasnosti dijagnoze. Predložene metodologije će biti upoređene radi utvrđivanja njihove efikasnosti u identifikovanju različitih tipova tumora,

---

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

potencijalno nudeći nove puteve za ranu detekciju i bolje ishode za pacijente [7].

2. PODACI

Skup podataka koji je upotrebljen za potrebe ovog rada sastoji se od slika MRI snimaka endokranijuma čoveka koji je podeljen u dva direktorijuma, trening i test. U okviru tih direktorijuma nalaze se još 4 dodatna direktorijuma koji predstavljaju 4 klase koje se nalaze u setu. Ukupno sadrže 3264 fotografije. Nazivi klasa su glioma (926 fotografija) , meningioma (937 fotografija), bez tumora(500 fotografija) i tumor hipofize (901 fotografija.

Informacija o nazivu direktorijuma korišćena je kao klasa (labela) u procesu klasifikacije. Iako je skup bio podeljen na trening i test skup, tokom učitavanja sve slike su uz informaciju kojoj klasi pripadaju učitane u jednu kolekciju. Slike su skalirane na jednake dimenzije 224 x 224. Takav skup koji sadrži sve fotografije zatim je podeljen na trening skup i test skup u odnosu 80:20. Dalje transformacije vršene su nad trening skupom. Od tog skupa stvoren je još jedan skup koji je uz originalne fotografije sadržao i augmentovane fotografije. Od svake fotografije nasumično je stvorena jedna augmentovana primenom jednog od odabranih metoda koje su činili nasumično rotiranje fotografije za 90 stepeni, obrtanje po horizontalnoj i vertikalnoj osi. U slučaju da je u tom procesu došlo do nastanka novih piksela za metod popunjavanja izabran je najbliži. Takva dva trening seta su zatim podeljena na trening i validacioni skup u odnosima 50:50, 70:30 i 90:10. Ovim načinom pretprocesiranja i podele trening skupa svaki model je treniran na 6 različitih pristupa i testiran istim skupom podataka. Prilikom ovih podela vođeno je računa da raspodela klasa u svim skupovima ostane ista kao i u originalnom skupu. Opisani skup podataka preuzet je sa izvora [8].

3. EKSPERIMENT

U ovom poglavlju je predstavljen opis implementacije sistema za određivanje vrste tumora mozga na osnovu slika MRI snimaka endokranijuma. U osnovu sistema je klasifikacija slika na jednu od četiri moguće klase: slika bez tumora, tumor hipofize, glioma i meningioma. Ulaz u sistem je slika MRI snimka endokranijuma pacijenta, a izlaz je dijaznoza predviđena od strane modela. Implementacija sistema biće opisana na visokom nivou apstrakcije.

Nad pretreniranim modelima primenjene su tehnike učenja sa prenosom znanja i finog podešavanja parametara. Svi modeli su trenirani na dva skupa podataka. Prvi skup podataka čine slike iz izvornog skupa podataka na kojima je primenjeno pretprocesiranje u vidu skaliranja slike na 224 x 224 piksela, dok je na drugom skupu podataka na delu slika namenjenom za trening primenjena augmentacija, opisana u prethodnom poglavlju.

Pretrenirani modeli su prilagođeni za rešavanje problema klasifikacije MRI snimaka endokranijuma tako što je poslednji sloj modifikovan tako da kao izlaz ima 4 klase koje su prisutne u ulaznom setu podataka. Takođe su korišćene tehnike treniranja sa prenosom znanja i finog podešavanja parametara tako što je isprobano da se određen broj poslednjih slojeva prilagodi ulaznom setu podataka. Broj odmrznutih poslednjih slojeva je iznosio od 0 do 5.

Tokom treniranja kao kriterijum za gubitak je korišćena CrossEntropyLoss funkcija. Ona se oristi za klasifikacione probleme u kojima se predviđaju klase i čini tipičan izbor za zadatke višeklasne klasifikacije. Za optimizator je korišćen Adam koji je jedan od najčešće korišćenih algoritama za optimizaciju u dubokom učenju. On se koristi za računanje težina modela na osnovu grešaka izračunatih tokom treniranja. U svakoj iteraciji model se proverava validacionim skupom podataka. Izlaz iz te operacije se koristi za računanje gubitaka. Tokom treniranja implementirano je rano zaustavljanje. Svaki model koji na validacionom skupu podataka ima manji gubitak od najboljeg do tog trenutnka čuva se kao najbolji model. Ukoliko se gubitak ne pobošlja u 5 uzastopnih epoha, trening se zaustavlja kako bi se izbeglo preterano treniranje.

3.1. Model ResNet50

U ovom pristupu za rešavanje problema klasifikacije medicinskih slika oproban je model razvijen od strane Microsoft-a. Model je objavaljen u 2015. godini i broj 50 u modelu predstavlja broj slojeva u mreži. Arhitektura modela podeljena je u 4 dela koje čine konvolucioni slojevi, identitet blok, konvolucioni blok i konvolucioni blok koji je zadružen za procesiranje i transformaciju odlika. Na kraju se nalazi potpuno povezani sloj koji vrši krajnju klasifikaciju. Model je treniran na ImageNet setu podataka koji sadrži više od 14 miliona fotografija i 1000 klasa. Iz tog razloga predstavlja moćan model koji može da se koristi za više klasifikacionih problema poput detekcije objekata, prepoznavanja lica i analize medicinskih slika. Takođe ima primenu i u ekstrakciji odlika za druge zadatke, kao što je detekcija objekata i semantička segmentacija.Kao ulaz koristi fotografije dimenzija 224 x 224 koje sadrže 3 kanala.

3.2. Model DenseNet121

U ovom pristupu za rešavanje problema klasifikacije medicinskih slika oproban je model razvijen od strane grupe naučnika. Model je objavaljen u 2016. godini i broj 121 u modelu predstavlja broj slojeva u mreži.

Sastoji se od serije gustih blokova od kojih svaki sadrži više konvolucionih slojeva. Svaki blok kao ulaz prima izlaz prethodnog bloka ali i sve izlaze prethodnih blokova. To zajedno čini gusto povezanu konvolucionu mrežu između svih slojeva u modelu što omogućava informacijama da efikasnije putuju kroz mrežu, po čemu je model i dobio ime. Model se široko koristi za rešavanje problema klasifikacije što mu je omogućilo treniranje na više setova podataka od kojih su neki ImageNet, CIFAR-10, and CIFAR-100. Ograničenje ovog modela jeste da ulazne fotogracije moraju biti dimenzija 224 x 224 i imati tačno 3 kanala, što znači da sadrže boju. Postoje varijacije ovog modela koje podržavaju drugačije dimenzije. Nalazi primenu u klasifikaciji fotogracija, detekciji objekata, analizi medicinskih slika i procesiranju jezika.

3.3. Model EfficientNet-B3

U ovom pristupu za rešavanje problema klasifikacije medicinskih slika korišćen je model EfficientNet-B3, koji je razvijen kao deo serije EfficientNet modela predstavljenih 2019. godine. EfficientNet predstavlja arhitekturu neuronskih mreža i metod skaliranja koji uniformno skalira sve dimenzije dubine, širine/rezolucije koristeći složeni koeficijent. Arhitektura EfficientNet-a se sastoji od naslaganih konvolucionih slojeva obično organizovanih hijerarhijski. Svaki konvolucioni sloj je praćen aktivacionom funkcijom i slojevima za normalizaciju. Konvolucije razdvojive po dubini su široko primenjene u ovoj mreži, što omogućava visoku tačnost uz značajno manje parametara u poređenju sa tradicionalnim arhitekturama. Model je dizajniran tako da izvlači maksimalne performanse dok smanjuje broj računanja i memorije potrebne za treniranje, čineći ga posebno pogodnim za primenu na medicinskim slikama gde je preciznost ključna. EfficientNet-B3 se trenira na različitim skupovima podataka, uključujući ImageNet, i pokazao je izvanredne rezultate u zadacima klasifikacije slika, kao i detekciji objekata. Model nalazi primenu u analizi medicinskih slika, kao i u drugim oblastima poput klasifikacije fotografija i procesiranja jezika.

5. REZULTATI

Na osnovu sprovedenih eksperimenata zaključak je da su najlošije rezultate beležili modeli čija je osnova bio pretrenirani model ResNet50. Eksperiment sa ovim modelom proizveo je najmanje modela koji su imali tačnost veću od 90% i najbolji model ove klase (tačnost 94.79%) je slabiji od najboljih modela iz preostala dva eksperimenta. Modeli proistekli iz eksperimenta čija je osnova bio pretrenirani model DenseNet121 su pokazali bolje rezultate u proseku i iz ovog eksperimenta proističe drugi najbolji model (tačnost 96.32%). Eksperiment u kom je korišćen pretrenirani model EfficientNet-B3 je proizveo u proseku najbolje modele i pojedinačno model sa najboljim preformansama nakon evaluacije (tačnost 98.01%). U nastavku poglavlja date su tabele u kojima se porede najrelevantnije mere performansi najjačeg modela iz svakog eksperimenta. Za klasu snimaka bez tumora to je mera preciznosti (tabela 1).

Tabela 1. *Vrednosti mere preciznosti za klasu no_tumor za najbolje modele iz sva tri eksperimenta*

Model	Bez tumora
ResNet50	0.97
DenseNet121	0.97
EfficientNet-B3	0.97

Za tumor klase mera odziva (tabela 2).

Tabela 2. *Vrednosti mere odziva za tumor klase za najbolje modele iz sva tri eksperimenta*

	Glioma	Meningioma	Tumor hipofize
ResNet50	0.96	0.89	1
DenseNet121	0.95	0.93	1
EfficientNet-B3	0.98	0.96	1

Takođe je data tabela sa marko f-merama svih eksperimenata (tabela 3).

Tabela 3. *Vrednosti makro f-mere za najbolje modele iz sva tri eksperimenta*

Model	Makro f-mera
ResNet50	0.9485
DenseNet121	0.9647
EfficientNet-B3	0.9802

Po svakom od ovih kriterijuma, model dobijen finim podešavanjem parametara od pretreniranog modela EfficientNet-B3 beleži najbolje rezultate.

6. ZAKLJUČAK

Iz rezultata prikazanih u poglavlju 5., može se primetiti da sistem koristeći pretrenirani model EfficientNet-B3 daje rezultat od 98% tačnosti, što je veoma dobar rezultat. Glavna prednost korišćenja ovog sistema u odnosu na čoveka je ta što je sposoban da uoči šablone i odnose između piksela koje golom oku možda i nisu značajni. Iako sistem poseduje dobre performanse zbog osetljivosti problema kojim se bavi ne bi trebalo da se koristi nezavisno od lekara, već samo kao alat koji bi pomogao sugerišući na potencijalan tumor. Sistem beleži mali broj lažnih negativnih slučajeva za klasu bez tumora što je značajno jer može skrenuti pažnju lekara da je potrebno obaviti dodatnu analizu za nekog pacijenta. Ukoliko bi sistem klasifikovao snimak kao da poseduje tumor sprovela bi se dodatna analiza od strane lekara i na osnovu dijagnoze započela konkretna terapija.

Jedno od mogućih proširenja ovog sistema je integracija sa aparatom za snimanje MRI snimaka. Na taj način bi se dijagnoza obavljala odmah nakon snimaka i lekar bi imao uvid u predikciju čim pristupi snimcima. Drugi način jeste korišćenje složenijih i zahtevnijih pretreniranih modela od modela korišćenih u ovom radu. Još jedan od načina jeste proširenje skupa podataka čime bi se postigla bolja generalizacija sistema i rezultati na nepoznatim setovima. Uz to bi bilo moguće uključiti dodatne vrste tumora u

klasifikaciju. Uz same snimke u proces klasifikacije mogli bi se uključiti i drugi podaci o pacijentu koji se mogu koristiti pri dijagnositici tumora na mozgu. Ovim postupcima bi dijagnoza bila još tačnija i sistem bi potencijalno mogao da se koristi i u realnim uslovima kao dodatni alat pri dijagnostici za medicinko osoblje.

7. LITERATURA

[1] https://www.abta.org/about-brain-tumors/brain-tumor-education/ (pristupljeno u avgustu 2024.)

[2] D. N. Louis, A. Perry, G. Reifenberger, A. von Deimling, D. Figarella-Branger, W. K. Cavenee, and D. W. Ellison, "The 2016 World Health Organization classification of tumors of the central nervous system: a summary," Acta Neuropathologica, vol. 131, no. 6, pp. 803-820, Jun. 2016.

[3] Q. T. Ostrom, G. Cioffi, H. Gittleman, N. Patil, K. Waite, C. Kruchko, and J. S. Barnholtz-Sloan, "CBTRUS statistical report: Primary brain and other central nervous system tumors diagnosed in the United States in 2012–2016," Neuro-oncology, vol. 21, no. Suppl 5, pp. v1-v100, 2019.

[4] J. Hardy and J. Rees, "Pituitary tumours: diagnosis and management," Postgraduate Medical Journal, vol. 84, no. 991, pp. 172-178, 2008.

[5] World Health Organization, "Global health workforce statistics," 2022.

[6] J. Smith and A. Lee, Advances in brain tumor classification and detection, New York: Springer, 2023.

[7] https://www.atlasofms.org/chart/serbia/epidemiology/number-of-people-with-ms (pristupljeno u avgustu 2024.)

[8] https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri?resource=download (pristupljeno u avgustu 2024.)

Kratka biografija:

Aleksa Stojić rođen je u Novom Sadu 1999. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva odbranio je 2022.god. kontakt: aleksa.stojic19@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/30BE22Ristic

RAZVOJ SISTEMA ZASNOVANOG NA DUBOKOM UČENJU ZA PREPOZNAVANJE LICA I GESTOVA ŠAKE

DEVELOPMENT OF A SYSTEM BASED ON DEEP LEARNING FOR FACE AND HAND GESTURE RECOGNITION

Branislav Ristić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu će biti predstavljena implementacija sistema za prepoznavanje lica i gestova šake implementiranog u Python programskom jeziku. Rad uključuje teorijske osnove, kao i implementacione detalje.*

Ključne reči: *veštačka inteligencija, mašinsko učenje, neuronske mreže, duboko učenje, prepoznavcanje lica, prepoznavanje gesta šake, računarski vid*

Abstract – *This paper will present the implementation of a face and hand gesture recognition system implemented in the Python programming language. The paper includes theoretical foundations as well as implementation details.*

Keywords: *artificial intelligence, machine learning, neural networks, deep learning, face recognition, gesture recognition, computer vision*

1. UVOD

Razvoj sistema za prepoznavanje lica i gestova šake predstavlja jednu od ključnih oblasti u istraživanju računarskog vida i interaktivnih tehnologija. Ovakvi sistemi omogućavaju primenu u širokom spektru oblasti, uključujući bezbednost, automatizaciju i korisničke interfejse. Prepoznavanje lica je postalo standard u različitim aplikacijama, od automatskog otključavanja uređaja do nadzornih sistema, dok se prepoznavanje gestova šake sve više koristi u razvoju prirodnih i intuitivnih metoda interakcije.

U ovom radu predstavljen je sistem za prepoznavanje lica i gestova šake, koji koristi savremene metode računarskog vida i mašinskog učenja. Osnovna ideja rada je razvoj rešenja koje omogućava interakciju korisnika sa okruženjem pomoću prirodnih metoda komunikacije, kao što su izrazi lica i pokreti šake.

Prepoznavanje lica uključuje detekciju i identifikaciju lica, dok se prepoznavanje gestova fokusira na analizu pokreta šake. Razvoj jednog ovakvog sistema podrazumeva primenu algoritama dubokog učenja za ekstrakciju i analizu relevantnih informacija iz video-snimaka. Pored toga, ispituje se efikasnost i robusnost predloženog pristupa u različitim scenarijima. Rad takođe uključuje pregled postojećih rešenja, kao i analizu predloženog modela sa aktuelnim metodama koje se koriste u ovoj oblasti.

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

2. OSNOVE MAŠINSKOG UČENjA I NEURONSKIH MREŽA

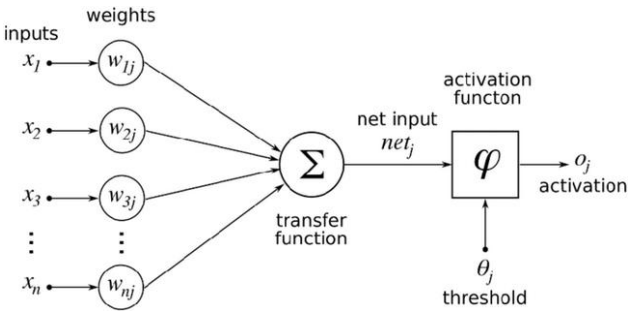
Veštačka inteligencija, u širem smislu, predstavlja mogućnost posedovanja inteligencije od strane mašina, konkretno, računarskih sistema [8].

Algoritam (ili model) mašinskog učenja jeste algoritam koji poseduje mogućnost da uči iz podataka koji mu se pruže. Mašinsko učenje predstavlja podksup veštačke inteligencije. U okviru mašinskog učenja veliku oblast predstavlja duboko učenje [2, p. 99].

Algoritmi mašinskog učenja se ugrubo mogu podeliti u nadgledane i nenadgledane. Nadgledani algoritmi mašinskog učenja poseduju obeležja koja predstavljaju ulaz u algoritam (trening skup), ali uz njih i ciljno obeležje. Nenadgledani algoritmi imaju za cilj da među podacima sami pronađu šablone [2, pp. 104-105].

2.1. Neuronske mreže

Veštački neuron jeste matematička funkcija koja je nastala po ugledu na neuron u ljudskom mozgu. Predstavlja elementarnu jedinicu veštačkih neuronskih mreža, koje su sastavljene od slojeva neurona. Veštački neuron na ulazu ima listu ulaza (x_i), odnosno dendrite, svakom od ulaza ima pridruženu težinu (w_{ij}). Sumiranjem proizvoda ulaza i težina, potom nadodavanjem praga (θ_j) i prosleđivanjem tog rezultata kao ulaz aktivacionu funkciju (φ) dobija se aktivacija, odnosno akson, O_j, j-tog neurona u sloju neuronske mreže. Ilustracija se nalazi na Slici 1 [9].



Slika 1. *Prikaz veštačkog neurona (preuzeto sa [14])*

Unapred propagirajuće (engl. *feed forward*) neuronske mreže su klasa unutar neuronskih mreža koju karakteriše nepostojanje povratnih sprega u mreži. Duboke neuronske

mreže obuhvataju mreže koje imaju velik broj slojeva neurona [1, p. 75].

2.2. Konvolucione neuronske mreže

Konvolucione neuronske mreže predstavljaju posebnu vrstu neuronskih mreža koje obrađuju podatke sa rešetkastom topologijom. Međutim, konvolucija se zapravo ne koristi u konvolucionim mrežama, već se najčešće koristi kros-korelacija.

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(i+m, j+n) K(m,n) \quad (1)$$

Kros-korelacija prikazana u (1) suštinski predstavlja konvoluciju fotografije *I*, bez rotacije kernel matrice *K*. Rezultat operacije kros-korelacije naziva se mapa svojstava (engl. feature map) [2, pp. 330-333].

2.3. Rezidualne neuronske mreže

Problem obuke veoma dubokih neuronskih mreža je rešiv upotrebom prečica između slojev. Odnosno, ulaz u blok i izlaz iz bloka se kombinuju kako bi se dobio jedan rezultat. Na ovaj način, pošto se inicijalno težine i pragovi inicijalizuju na nasumične vrednosti, se omogućava u početku manji faktor šuma pri propagaciji unapred i unazad. Ovo dalje dozvoljava kreiranje i obuku rezidualnih neuronskih mreža sa do nekoliko stotina ili hiljada slojeva. Koriste se između ostalog i za generisanje vektorske reprezentacije objekata [3].

3. OPIS TEHNOLOGIJA

U ovom poglavlju će biti predstavljene korišćene tehnologije, sa fokusom na alate i metode koji su primenjeni u istraživanju i razvoju.

3.1. *Linux*

Linux je familija operativnih sistema otvorenog koda koja je bazirana na Linux kernelu (kernel u ovom kontekstu označava jezgro operativnog sistema, ne kernel matricu). Otvorenost koda omogućava visoku podesivost sistema prema različitim potrebama, stoga postoji veliki broj distribucija Linuxa [11].

3.2. *Python*

Python je programski jezik visokog nivoa apstrakcije koji pripada grupi interpretiranih jezika. Poznat je po čitljivosti i svestranosti. Zbog svojih osobina popularan je odabir od strane početnika, a i iskusnih programera. Podržava više programskih paradigmi uključujući i proceduralnu, objektnu i funkcionalnu [13].

3.3. *MediaPipe*

MediaPipe predstavlja skup biblioteka i alata koje omogućuju programerima brzu upotrebu veštačke inteligencije. Produkt je kompanije Google. Programerima je omogućeno stvaranje modela po meri, ali i korišćenje već istreniranih modela, koje Gugl zvanično čini dostupnim.

Detekcija ključnih tačaka šake (engl. Hand landmarks detection) dozvoljava detekciju ključnih tačaka svih šaka koje se nalaze u kadru.

Model prepoznavanja gesta šake (engl. Gesture recognition) je paket koji koristi rezultate prethodno opisanog modela za ključne tačke šake kao ulaze u model za klasifikaciju. Kategorije gestova šake: nepoznat gest, oznaka: *Unknown*; zatvorena šaka, oznaka: *Closed_Fist*; otvoreni dlan, oznaka: *Open_Palm*; pokazivanje nagore, oznaka: *Pointing_Up*; palac dole, oznaka: *Thumb_Down*; palac gore, oznaka: *Thumb_Up*; pobednička gestikulacija, oznaka: *Victory*; ljubav, oznaka: *ILoveYou* [4, 5, 6].

3.4. *DeepFace*

Sefik Serengil je softverski inženjer i kreator biblioteke *DeepFace* koja omogućava podršku mašinskog učenja vezano za lica. *DeepFace* predstavlja omotač oko raznih modela poput *VGG-Face, Google Facenet, Facebook DeepFace, DeepID, OpenCV, Dlib* [7].

3.5. *OpenCV*

OpenCV je softverska biblioteka otvorenog koda koja pruža podršku kompjuterskog vida u realnom vremenu. Nastala je u okviru kompanije Intel 2000. godine, a 2011. godine počinje da pruža podršku GPU akceleracije. Napisana je u jeziku C++, dok su dostupna vezivanja za Python, MATLAB, Java programske jezike [12]. Biblioteka poseduje model koji omogućava analizu sentimenata lica. Rezultat analize sentimenata predstavlja klasu koja može imati sledeće vrednosti: ljuta, oznaka: angry; gađenje, oznaka: disgust; strah, oznaka: fear; sreća, oznaka: happy; tuga, oznaka: sad; iznenađenje, oznaka: surprise; neutralna, oznaka: neutral. [7]

3.6. *Dlib*

Dlib predstavlja biblioteku opšte namene napisane u jeziku C++. Od početka razvoja godine 2002. implementirani su razni alati, te u 2016. skup alata je posedovao podršku za: niti, strukture podataka, linearnu algebru, mašinsko učenje, obradu slike, s tim da se poslednjih godina fokus usmerava na mašinsko učenje [10].

4. OPIS IMPLEMENTACIJE

U ovom poglavlju razmatraće se implementacija sistema koja objedinjuje prepoznavanje lica i gesta šake. Osim toga, analiziraće se tehničke prepreke i rešenja za optimizaciju performansi i tačnosti prepoznavanja.

Cilj sistema je da podrži prepoznavanje gestova šake, povezivanje šake sa odgovarajućim licem, prepoznavanje emocije na licu i da li je lice već poznato od ranije (da li je korisnik već interagovao sa sistemom), i najzad, beleženje rezultata gesta. Interakcija sa sistemom se svodi na pokazivanje gesta šake, gledajući u kameru.

Sistem koristi postojeće algoritme mašinskog učenja, među kojima su:

- prepoznavanje gesta šake - MediaPipe v0.10.14,
- prepoznavanje emocije - DeepFace v0.0.92 (OpenCV v4.10.0.84),
- prepoznavanje izgleda lica - DeepFace v0.0.92 (Dlib v19.24.4).

Za prepoznavanje gesta šake korišćen je HandGestureClassifier, odnosno gesture_recognizer.task iz MediaPipe softverskog paketa [4].

Za prepoznavanje emocija korišćen je facial_expression_model_weights.h5 iz *OpenCV*, dok je za prepoznavanje izgleda lica korišćen dlib_face_recognition_resnet_model_v1.dat iz Dlib paketa.

Sistem podržava vizuelizaciju u pogledu obojenih granica (kutija) oko detektovanih šaka i lica, kao i tekstualno beleženje rada sistema na standardnom izlazu.

4.1. Podaci

Podaci koji predstavljaju rezultate rada sistema su:

- FaceHash,
- emocija,
- tip gesta,
- vreme (Unix timestamp)

FaceHash predstavlja base64 kodovanu vrednost SHA256 funkcije nad vektorskom reprezentacijom lica. Emocija predstavlja klasu čije su vrednosti nabrojane u poglavlju 3.5. Tip gesta podrazumeva vrednosti koje su nabrojane u poglavlju 3.3.2. Vreme prestavlja standardni Unix timestamp izražen u sekundama.

4.2. Memorija

U ovom poglavlju će biti obrađena tri nivoa memorije u kojima se skladište podaci u sistemu.

4.2.1 Pret-keš

Pret-keš se koristi kako bi se poboljšala tačnost algoritma za prepoznavanje lica. Uprkos velikoj tačnosti algoritma, može se dogoditi da pri računanju vektorske reprezentacije lica dobije netačno podudaranje. Iz tog razloga se rezultati obrade kadrova koji su vremenski blizu, uzimaju kao jedna celina. Algoritam čeka, i ukoliko se ne pojavi ni jedno lice u okviru od 30 kadrova (podesiv parametar), pokreće se obrada. Nad blokom koji se obrađuje se većinski odlučuje koje je vrste lice (novo lice ili neko staro). U pret-kešu se pored gorenavedenih podataka čuva i vektorska reprezentacija lica. Ovo služi kao kratkotrajna memorija, rezultat obrade pret-keša se smešta u keš kao jedan podatak.

4.2.2 Keš

Vrednosti sa kojima se upoređuju potencijalna nova lica u kadru se nalaze u kešu. Keš je mesto u kojem se skladište rezultati dobijeni obradom bloka rezultata u pret-kešu. Ukoliko se desi da je rezultat zabeležen za neko staro lice, vrednosti gesta vremena i emocije se ažuriraju. U kešu se, takođe, pored gorenavedenih podataka čuva i vektorska reprezentacija lica. Ovo služi kao srednjetrajna memorija.

4.2.2 Masovna memorija

Ukoliko za dato lice u kešu prođe vremenski period od 30 minuta, lice se memoriše na disk (baza podataka) i potom uklanja iz keša.

Takođe, svakih 15 minuta se pokreće kopiranje podataka iz keša na disk (snapshotting, bez brisanja iz keša).

Oba načina čuvanja podataka na disku se ogledaju korišćenjem .csv datoteke. Kao baza podataka se koristi jedna datoteka, dok se za stanja (snapshot) sistema koristi iznova nova datoteka, koja u svom nazivu sadrži i vreme kreiranja iste.

4.3. Algoritam

Algoritam je u stanju da detektuje maksimalno 5 šaka u kadru od jednom. Kao mera udaljenosti lica korišćeno je euklidsko rastojanje.

Aplikacija prima signal sa veb-kamere u vidu nizova slika. Ukoliko algoritam u kadru uspešno prepozna gest šake koji nije "*None*", pokreće se detekcija lica i analiza sentimenata. Ukoliko je ta operacija uspešna, vrši se računanje vektorske reprezentacije lica. Ukoliko je ta operacija uspešna, povezuje se šaka sa licem i skladišti kao međurezultat. Zatim se proverava da li je lice već interagovalo sa sistemom (već je u kešu). Ukoliko jeste, dodeljuje mu se odgovarajuća SHA256 vrednost, ukoliko nije, neophodno je da se izračuna. Taj rezultat se skladišti u pret-kešu.

Svakih 30 kadrova se obrađuje pret-keš. Ovo je način da se poboljša tačnost algoritama mašinskog učenja u slučaju da dođe do netačnog poklapanja ili nepoklapanja.

Svaki treći kadar se zapravo obrađuje, dok se za ostale kadrove korisniku prikazuju prethodno izračunati rezultati (granice lica i šaka). Ovo je način da se unapredi odziv sistema budući da metoda koja je odgovorna za prepoznavanje gestova lica vrši obradu sinhrono.

Paralelno sa niti glavnog algoritma se izvršava i nit pražnjenja keša na disk i beleženja stanja keša (engl. *snapshotting*). Ona započinje svoj rad kada i glavni algoritam.

4.4. Detaljnija razmatranja

Obrada svakog trećeg kadra je posledica nestabilne asinhrone obrade u okviru MediaPipe biblioteke. Ovaj parametar je svakako podesiv prema računarskoj moći mašine na kojoj se aplikacija izvršava. Algoritam detektuje maksimalno 5 šaka u kadru. Ovo je omogućeno kako bi se eventualne nepravilnosti rada algoritma prepoznavanja šaka ublažile.

Računanje aritmetičke sredine nad pret-kešom se ispostavlja da značajno poboljšava tačnost modela mašinskog učenja za prepoznavanje lica, iako je reklamirana tačnost 97,53%. Za poređenje sličnosti dva lica se koriste isključivo vektorske reprezentacije lica. U masovnoj memoriji se ne skladišti vektorska reprezentacija lica, već samo SHA256 vrednost vektora. Mogućnost poravnavanja lica pri detekciji se svodi na rotaciju za korake od po $\pi/2$, stoga je neophodno što ispravnije (potpuno vertikalno) postaviti kameru. Performanse aplikacije variraju u odnosu na osvetljenje.

Za najbolje performanse obezbediti dobro, ali ne previše, osvetljenu prostoriju. Preterano pomeranje i pričanje za vreme interacije sa sistemom takođe utiče na performanse rada algoritma. Pozadina bi, idealno, trebalo da bude jednobojno platno.

5. ZAKLJUČAK

Predloženo rešenje u ovom radu omogućava interakciju sa sistemom isključivo putem prirodne, neverbalne komunikacije, koristeći gestove i vizuelne signale. Ovaj pristup je omogućen primenom relativno jednostavnih i lako dostupnih tehnologija koje ne zahtevaju velike hardverske resurse, čineći sistem pogodnim za širok spektar aplikacija. Osnovna ideja je da se korisnička interakcija obavlja bez verbalne komunikacije, čime se omogućava intuitivniji i prirodniji način rada sa sistemom.

Sistem nudi visoku fleksibilnost kroz podešavanje parametara kao što su broj kadrova koji se preskaču, maksimalan broj detektovanih šaka, kao i izbor različitih modela mašinskog učenja. Ova fleksibilnost omogućava da se sistem lako prilagodi različitim okruženjima i hardverskim konfiguracijama, što je korisno u raznovrsnim scenarijima primene. Pored toga, upotreba pret-keša i keša povećava efikasnost obrade podataka, omogućavajući postepenu obradu i unapređenu tačnost sistema.

Sistem takođe delimično garantuje privatnost korisnika, jer trajno ne skladišti osetljive podatke kao što su vektorske reprezentacije lica, osim u RAM-u tokom procesa uparivanja lica.

Dodatna poboljšanja mogu uključivati razvoj naprednijih algoritama za prepoznavanje lica koji bi mogli bolje da se nose sa različitim uglovima i uslovima osvetljenja. Takođe, uvođenje podrške za složenije gestove i korišćenje više ruku istovremeno moglo bi da proširi primenu sistema u složenijim scenarijima. Razmatranje ovih poboljšanja moglo bi značajno povećati primenu i efikasnost ovog sistema u različitim oblastima.

6. LITERATURA

[1] Andriy Burkov. (2019). THE HUNDRED-PAGE MACHINE LEARNING BOOK. Andriy Burkov.

[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. The MIT Press

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2015, December 10). Deep Residual Learning for Image Recognition. ArXiv.org; arXiv. https://arxiv.org/abs/1512.03385

[4] Google. (n.d.-a). Gesture recognition task guide | Google AI Edge. Google for Developers. Retrieved September 2, 2024, from https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer

[5] Google. (n.d.-b). Hand landmarks detection guide | Edge. Google for Developers. Retrieved September 2, 2024, from https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker

[6] Google. (n.d.-c). SOLUTION DETAILS. https://storage.googleapis.com/mediapipe-assets/Model%20Card%20Hand%20Tracking%20(Lite_Full)%20with%20Fairness%20Oct%202021.pdf

[7] Serengil, S. I. (2020, August 30). serengil/deepface. GitHub. https://github.com/serengil/deepface

[8] Wikipedia. “Artificial Intelligence.” Wikipedia, Wikimedia Foundation, 18 Feb. 2019, en.wikipedia.org/wiki/Artificial_intelligence. Accessed 2 Sept. 2024.

[9] Wikipedia Contributors. “Artificial Neuron.” Wikipedia, Wikimedia Foundation, 4 Oct. 2018, en.wikipedia.org/wiki/Artificial_neuron. Accessed 2 Sept. 2024.

[10] Wikipedia Contributors. “Dlib.” Wikipedia, Wikimedia Foundation, 22 Sept. 2019, en.wikipedia.org/wiki/Dlib. Accessed 2 Sept. 2024.

[11] Wikipedia Contributors. “Linux.” Wikipedia, Wikimedia Foundation, 14 Sept. 2019, en.wikipedia.org/wiki/Linux. Accessed 2 Sept. 2024.

[12] Wikipedia Contributors. “OpenCV.” Wikipedia, Wikimedia Foundation, 29 Aug. 2019, en.wikipedia.org/wiki/OpenCV. Accessed 2 Sept. 2024.

[13] Wikipedia Contributors. “Python (Programming Language).” Wikipedia, Wikimedia Foundation, 4 May 2019, en.wikipedia.org/wiki/Python_(programming_language). Accessed 2 Sept. 2024.

[14] Polat, Ali. (2017). Hidden bank charges amidst ethics, transparency, and regulation: Case of rebate programs of international banks.

Kratka biografija:

Branislav Ristić je rođen 30. maja 2000. godine u Novom Sadu. Osnovne akademske studije završio je školske 2022/2023. godine, smer Informacioni inženjering. Planira da nastavi sa svojim usavršavanjem kroz doktorske akademske studije i rad na fakultetu.

kontakt: branislav.ristic@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.313
DOI: https://doi.org/10.24867/30BE23Golic

PREGLED MODIFIKACIJA KALMANOVOG FILTERA NA PRIMERU PRAĆENJA POKRETNOG OBJEKTA
REVIEW OF KALMAN FILTER MODIFICATIONS USING THE EXAMPLE OF TRACKING A MOVING OBJECT

Anastasija Golić, *Fakultet Tehničkih nauka, Novi Sad*

Oblast – **Računarstvo i automatika**

Kratak sadržaj – *Cilj ovog rada je uporedjivanje performansi modifikacija Kalmanovog filtra u sv rhu estimacije trajektorije pokretnog objekta*

Ključne reči: *Kalmanov filtar, unscented Kalmanov filtar, extended Kalmanov filtar, esnamble Kalmanov filtr*

Abstract - *This guideline for submitting a manuscript for the Proceedings of the Faculty of Technical Sciences is given for publication of scientific and technical papers.*

Keywords: *Kalman filtar, unscented Kalamn filtar, extended Kalman filtar, ensemble Kalman filtar*

1. UVOD

U savremenim aplikacijama kao što su autonomna vožnja, navigacija robota i sistemi za kontrolu letelica, precizno praćenje objekata duž različitih trajektorija predstavlja ključni izazov. Za postizanje tačnosti u proceni pozicije, brzine i orijentacije, potrebne su napredne tehnike filtriranja koje integrišu nesigurne ili zašumljene podatke u realnom vremenu. U autonomnim vozilima, na primer, vozilo mora neprekidno procenjivati svoju poziciju i poziciju drugih učesnika u saobraćaju kako bi moglo bezbedno i efikasno navigirati [1]. Sličan princip važi za robote i letelice, gde precizna procena stanja i trajektorije omogućava uspešno obavljanje zadataka u dinamičnim okruženjima.

Osnovni problem je integracija nesigurnih podataka iz različitih senzora u konzistentan model kretanja. Senzori kao što su *GPS*, kamere i *LIDAR* često pružaju podatke koji su podložni šumu i nepreciznostima, što komplikuje procenu stanja [2]. Napredne tehnike filtriranja, poput Kalmanovog filtra, koriste se za postizanje tačnosti u proceni pozicije i orijentacije.

2. TEORIJSKI UVOD

Fizičkim sistemom upravlja se skupom upravljačkih signala, dok se njegovi izlazi ocenjuju pomoću mernih uređaja, tako da se znanje o ponašanju sistema dobija isključivo kroz ulaze i posmatrane izlaze. Realni sistemi šuma senzora i procesa, što zahteva stohastičke modele. Kalmanov filter se koristi za procenu stanja linearnih stohastičkih sistema. Stohastički procesi su opisani sa:

$$x_{k+1} = Ax_k + Bu_k + \omega_k \tag{1}$$

$$z_k = Hx_k + \nu_k \tag{2}$$

gde je ω_k Gausov beli šum sa kovarijansom $\mathbf{E}[\omega_k \omega_k^T] = Q_k$ ako $l = k$, a ν_k šum merenja sa kovarijansom $\mathbf{E}[\nu_k \nu_k^T] = R_k$ ako $l = k$. Iako su matrice kovarijanse Q i R parametri za podešavanje, u praksi statistika šuma često nije poznata ili nije Gausova [3].

Algoritam Kalmanovog filtra sastoji se iz dva koraka, predikcije i korekcije. Predikcija podrazumeva procenu stanja na osnovu matematičkog modela u prostoru stanja, dok se u korekciji procenjeno stanje koriguje na osnovu merenja. Korak predikcije opisa je sa

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{3}$$

$$\Sigma_k^- = A\Sigma_{k-1}A^T + Q \tag{4}$$

dok je korak korekcije definisan

$$K_k = \Sigma_k^- H^T (H\Sigma_k^- H^T + R)^{-1} \tag{5}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{6}$$

$$\Sigma_k = (I - K_k H)\Sigma_k^- \tag{7}$$

gde je $\hat{x}_k^-$ je predviđeno stanje u trenutku k, zasnovano na prethodnom stanju $\hat{x}_{k-1}$ i ulazu u_{k-1}, dok je Σ_k^- predikcija kovarijanse greške stanja, koja uzima u obzir procesni šum Q.

Kalmanov pojačivač K_k određuje koliko će novo merenje z_k uticati na korigovano stanje $\hat{x}_k$, a Σ_k predstavlja korigovanu matricu kovarijanse greške.

2.1. Algoritam proširenog kalmanovog filtra

Ukoliko je posmatrani sistem nelinearan, koristi se takozvani prošireni Kalmanov filtar. Nelinearni sistem opisan je jednačinama stanja i merenja:

$$x_{k+1} = f(x_k, u_k) + \omega_k \tag{8}$$

$$z_k = h(x_k) + \nu_k \tag{9}$$

gde su f i h nelinearne vektorske funkcije stanja i merenja. Vektori ω_k i ν_k predstavljaju šum procesa i merenja sa normalnom raspodelom i kovarijansama Q i R, redom.

Da bi se primenio *EKF*, nelinearne funkcije f i h linearizuju se u okolini trenutnog stanja primenom razvoja u Tejlorov red

$$f(x_{k-1}) \approx f(\hat{x}_{k-1}) + A_k(x_{k-1} - \hat{x}_{k-1}) \tag{10}$$

gde je A_k Jakobijeva matrica funkcije f:

$$A_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1}, u_{k-1}, 0}$$

Slično tome, linearizacija funkcije merenja h je:

$$h(x_k) \approx h(\hat{x}_k) + H_k(x_k - \hat{x}_k) \tag{11}$$

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Mirna Radović, vanred. prof.

gde je H_k Jakobijeva matrica funkcije h:

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_k}$$

Linearizovani model u prostoru stanja, glasi:

$$x_{k+1} = A_k x_k + B_k u_k + \omega_k \quad (12)$$

$$z_k = H_k x_k + \nu_k \quad (13)$$

gde je A_k matrica prelaza stanja, H_k matrica merenja, B_k matrica koja povezuje upravljačke signale sa stanjem, ω_k šum procesa, a ν_k šum merenja, [4]. Koraci estimacije prikazani su u tabeli 1

Tabela 1: *Koraci prosirenog Kalmanovog filtra*

Korak	Formule
Predikcija stanja	$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1})$
Predikcija kovarijanse	$\Sigma_k^- = A_k \Sigma_{k-1} A_k^T + Q$
Kalmanovo pojačanje	$K_k = \Sigma_k^- H_k^T (H_k \Sigma_k^- H_k^T + R)^{-1}$
Korekcija stanja	$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H_k \hat{x}_k^-)$
Korekcija kovarijanse	$\Sigma_k = (I - K_k H_k)\Sigma_k^-$

2.2. Algoritam unscented Kalmanovog filtra

Prošireni Kalmanov filtar (*EKF*) ima određena ograničenja zbog aproksimacije nelinearnih funkcija razvojem u Tejlorov red, što može dovesti do grešaka kada sistem nije skoro linearan. Osim toga, pretpostavlja se da su slučajne promenljive sa Gausovom raspodelom, što ograničava primenljivost u nekim realnim sistemima. Unscented Kalmanov filtar (*UKF*) je alternativni algoritam koji omogućava bolju procenu stanja za nelinearne sisteme. Za razliku od *EKF*-a, *UKF* ne koristi linearizaciju već primenjuje *unscented* transformaciju, koja preciznije računa statistike nelinearnih funkcija, [5].

Unscented Kalmanov filter koristi $2n+1$ sigma tačku za procenu srednje vrednosti i kovarijanse. Sigma tačke se generišu tako da zadržavaju istu srednju vrednost i kovarijansu kao originalna distribucija. One se zatim propagiraju kroz nelinearnu funkciju, što omogućava preciznu procenu izlaznih statistika.

Formulacija sigma tačaka je sledeća:

$$X_0 = \hat{x} \quad (14)$$

$$X_i = \hat{x} + \sqrt{(n+\lambda)}\Sigma_{xx}^i, \quad (15)$$

$$X_{i+n} = \hat{x} - \sqrt{(n+\lambda)}\Sigma_{xx}^i \quad (16)$$

gde je λ parametar koji kontroliše raspored sigma tačaka, n je dimenzija stanja, a Σ_{xx} je kovarijansna matrica stanja.

Nakon što su sigma tačke formirane, one se transformišu kroz nelinearnu funkciju i usrednjavaju da bi se dobile nove procene srednje vrednosti i kovarijanse. Težine koje se dodeljuju sigma tačkama određuju se na sledeći način:

$$W_0^{(m)} = \frac{\lambda}{n+\lambda} \quad (17)$$

$$W_0^{(c)} = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta) \quad (18)$$

$$W_i^{(m)} = W_i^{(c)} = \frac{1}{2(n+\lambda)} \quad \text{za} \quad i = 1,2,\ldots,2n \quad (19)$$

Ovaj metod omogućava tačnu aproksimaciju do drugog reda, što ga čini znatno preciznijim u poređenju sa *EKF*-om kada je u pitanju procena stanja nelinearnih sistema. Koraci ovog algoritma prikazani su tabelom 2

Tabela 2: Koraci Unscented Kalmanovog filtra (UKF)

Korak	Opis
1. Generacija sigma tačaka	$X_0 = \hat{x}$ $X_i = \hat{x} + \sqrt{(n+\lambda)\Sigma_{xx}}\mathbf{e}_i$ $X_{i+n} = \hat{x} - \sqrt{(n+\lambda)\Sigma_{xx}}\mathbf{e}_i$
	$Y_i = f(X_i)$
	$\hat{y} = \sum_{i=0}^{2n} W_i^{(m)} Y_i$ $\Sigma_{yy} = \sum_{i=0}^{2n} W_i^{(c)} (Y_i - \hat{y})(Y_i - \hat{y})^T$
2. Korekcija	$Z_i = h(X_i)$ $\hat{z} = \sum_{i=0}^{2n} W_i^{(m)} Z_i$ $\Sigma_{zz} = \sum_{i=0}^{2n} W_i^{(c)} (Z_i - \hat{z})(Z_i - \hat{z})^T$ $\Sigma_{xz} = \sum_{i=0}^{2n} W_i^{(c)} (X_i - \hat{x})(Z_i - \hat{z})^T$ $K = \Sigma_{xz}\Sigma_{zz}^{-1}$ $\hat{x} = \hat{x} + K(z - \hat{z})$ $\Sigma_x = \Sigma_x - K\Sigma_{zz}K^T$

2.3. Algoritam ensemble Kalmanovog filtra

Ensemble Kalman Filter (*EnKF*) je algoritam koji koristi metode *Monte Carlo* simulacija za procenu stanja dinamičkih sistema. Umesto jednog vektora stanja i kovarijanse, *EnKF* koristi ansambl stanja koji predstavlja verovatnu raspodelu sistema. Ovaj pristup je opisan modelom koji je definisan nelinearnim diferencijalnim jednačinama:

$$x_{k+1}^{(i)} = f(x_k^{(i)}, u_k) + \omega_k^{(i)} \quad (20)$$

i mernim sistemom:

$$z_k^{(i)} = h(x_k^{(i)}) + \nu_k^{(i)} \quad (21)$$

gde $x_{k+1}^{(i)}$ predstavlja stanje i-tog člana ansambla u trenutku $k+1$, dok f i h opisuju nelinearne funkcije dinamike i merenja. Šumovi procesa $\omega_k^{(i)}$ i merenja $\nu_k^{(i)}$ su opisani kovarijansama Q i R.

Pretpostavimo da je ansambl $x_{k-1}^{(1)}, \ldots, x_{k-1}^{(N)}$ uzorak iz raspodele filtriranja u trenutku $k-1$:

$$x_{k-1}^{(i)} \sim \mathcal{N}_n(\hat{x}_{k-1}, \Sigma_{k-1}) \quad (22)$$

Slično kao i kod standardnog Kalmanovog filtra, *ensemble* Kalmanov filtar se sastoji od dva koraka na svakom vremenskom koraku k: korak predikcije i korak korekcije.

Predikcija stanja se dobija usrednjavanjem:

$$\hat{x}_k^- = \frac{1}{N}\sum_{i=1}^{N} x_k^{(i)} \quad (23)$$

Analogno prethodno objašnjenim algoritmima, vrši se korekcija stanja:

$$z_k^{(i)} = h(x_k^{(i)}) + \nu_k^{(i)} \quad (24)$$

$$\hat{z}_k^- = \frac{1}{N}\sum_{i=1}^{N} z_k^{(i)} \tag{25}$$

$$\Sigma_{zz} = \frac{1}{N-1}\sum_{i=1}^{N}[z_k^{(i)} - \hat{z}_k^-][z_k^{(i)} - \hat{z}_k^-]^\top + R \tag{26}$$

$$\Sigma_{xz} = \frac{1}{N-1}\sum_{i=1}^{N}[x_k^{(i)} - \hat{x}_k^-][z_k^{(i)} - \hat{z}_k^-]^\top \tag{27}$$

$$K = \Sigma_{xz}\Sigma_{zz}^{-1} \tag{28}$$

$$\hat{x}_k = \hat{x}_k^- + K[z_k - \hat{z}_k^-] \tag{29}$$

$$\Sigma_k = \Sigma_k^- - K\Sigma_{zz}K^\top \tag{30}$$

3. MATEMATIČKI MODEL KRETANJA I SKUP PODATAKA

U ove svrhe stimacije putanje kretanja pokretnog objekta na osnovu zašumljenih merenja korišćen je model kretanja preuzet iz [2] i skup podatkata sa merenjima preuzet sa [7].

3.1. Matematicki model kretanja

Položaj objekta je određen vektorom $\mathbf{x} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$, gde x i y predstavljaju koordinate položaja, a θ ugao u odnosu na pozitivni pravac x-ose. Kontrola u trenutku t je $\mathbf{u}_t = \begin{pmatrix} v \\ \omega \end{pmatrix}$, gde je v translatorna brzina, a ω rotaciona brzina. Ako su brzine konstantne tokom vremena Δt, objekat se kreće krivom radijusa $r = \frac{v}{\omega}$, dok se za $\omega = 0$ kreće pravolinijski. Centar kružnice je dat sa:

$$x_c = x - v \cdot \Delta t \cdot \sin\theta \tag{31}$$

$$y_c = y + v \cdot \Delta t \cdot \cos\theta \tag{32}$$

Nakon vremena Δt, objekat će biti na poziciji $\mathbf{x}_t = (x, y, \theta)^T$ sa:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} x_c + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ y_c - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \theta + \omega\Delta t \end{pmatrix}$$

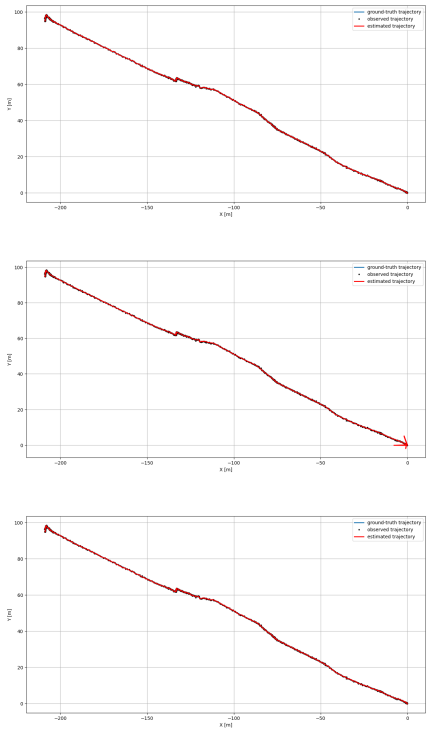
Polozaj, odnosno koordinate objekta su veličine koje se direktno mere, dok se ugap θ estimira.

3.2. Opis skupa podataka

KITTI je značajan skup podataka za računarski vid i autonomna vozila, koji pruža video snimke, podatke o dubini, poziciji i orijentaciji vozila prikupljene visokorezolucionim kamerama i *Velodyne* laserskim skanerom, kao i *GPS* i *IMU* podatke. Obuhvata različita okruženja i može sadržati mnoge objekte kao što su automobili i pešaci, što omogućava testiranje algoritama u realističnim uslovima. *XML* fajlovi sadrže informacije o poziciji, dimenzijama i tipu vozila. Podaci su korisni za testiranje i unapređenje algoritama za detekciju, praćenje i segmentaciju [7].

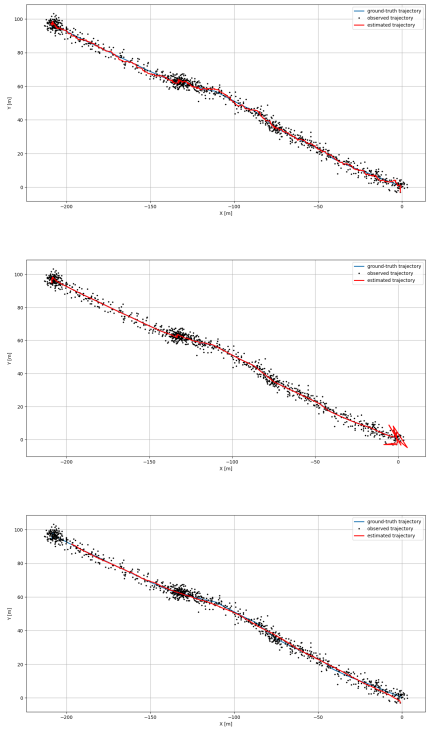
4. REZULTATI

U okviru ovog rada, simulacija je sprovedena za tri različita Kalmanova filtra: prošireni Kalmanov filtar, *unscented* Kalmanov filtar i *ensemble* Kalmanov filtar. Svaki od ovih filtara primenjen je na isti model kretanja objekta kako bi se uporedile njihove performanse u proceni putanje objekta na osnovu zašumljenih merenja. Simulacija je pokrenuta sa istim parametrima za sva tri algoritma, kako bi se poredile performanse estimatora pod istim uslovima. Parametri koji mogu ucitati na rezultat estimacije u sva tri slucaja su matrice kovarijansi $\mathbf{Q}, \mathbf{R}$ i Σ. Matrica $\mathbf{R}$ predstavlja kovarijansu mernih grešaka, odnosno određuje koliko poverenja imamo u merenja – manja vrednost znači veće poverenje u merenja, dok veća vrednost signalizira prisustvo većeg šuma. Matrica $\mathbf{Q}$ odnosi se na kovarijansu procesne greške, što reflektuje nesigurnost u modelu sistema – manja vrednost označava visok stepen poverenja u model, dok veća vrednost označava veću fleksibilnost filtera. Matrica Σ predstavlja kovarijansu početne procene stanja – manja vrednost ukazuje na sigurnost u početne uslove, dok veća vrednost signalizuje veću neizvesnost. Sve ove matrice zajedno utiču na to kako filter kombinuje informacije iz modela i merenja radi optimalne estimacije stanja sistema. Simulacija je sprovedena koristeći isti skup podataka i inicijalne uslove za sve metode. Cilj simulacije je bio da se uporede performanse filtara u smislu tačnosti procene stanja, stabilnosti i sposobnosti da se nosi sa različitim nivoima šuma u merenjima. Rezultati simulacije su analizirani kako bi se odredila najbolja metoda za konkretne uslove i zahteve praćenja objekta. Na slici 2 prikazani su rezultate estimacije trajektorije u slučaju da je standardna devijacija šuma merenja $\sigma = 0.15m$. Za male vrednosti šuma može se zaključiti da će sva tri algoritma dati zadovoljavajuće rezultate.



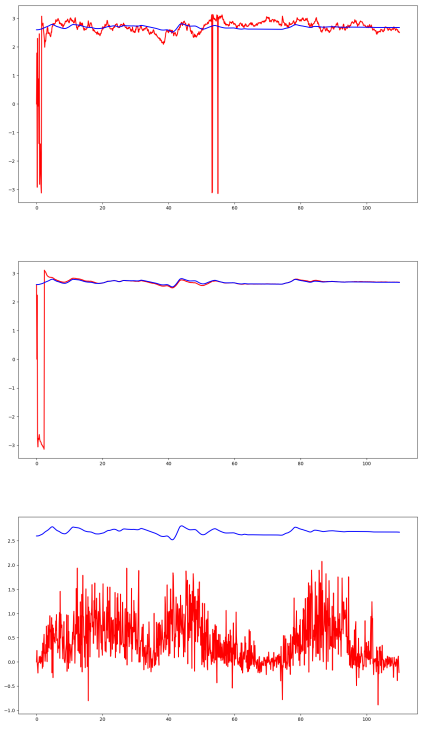
Slika 1: *Poređenje rezultata estimacije upotrebom 1.EKF 2.UKF 3.EnKF,* σ*=2m*

Slika 1 prikazuju rezultate estimacije trajektorije u slučaju da je standardna devijacija šuma merenja $\sigma = 2m$. Za razliku od slučaja kada šum merenja nije značajan, sada algoritam proširenog Kalmanovog filtra i *ensemble* Kalmanovog filtra daju znatno lošije rezultate, dok algoritam *unscented* Kalmanovog filtra daje tačne estimacije.



Slika 2: *Poređenje rezultata estimacije upotrebom 1.EKF 2.UKF 3.EnKF,* σ*=2m*

Osim estimacije koordinata objekta na osnovu zašumljenih merenja, vršena je i estimacija ugla usmerenosti na osnovu zadatog modela. Slika 3 prikazuje rezultate estimacije ugla θ. U sličaju da inicijalne vrednosti ugla nisu poznate, *UKF* daje jako dobre rezultate, dok *EnKF* ne uspeva da estimira nepoznati ugao.



Slika 3: *Poredjenje rezultata estimacije*

5. ZAKLJUČAK

Na osnovu izvršenih eksperimenata, *UKF* je pokazao najbolje performanse u smislu preciznosti procene stanja u prisustvu nelinearnosti sistema sa istim yazadtim parametrima.. Iako su *EKF* i *EnKF* postigli zadovoljavajuće rezultate, *UKF* je uspešno kombinovao efikasnost sa stabilnijom procenom varijacija u sistemu, što ga čini optimalnim izborom za ovaj problem estimacije trajektorije. UKF je postigao bolje rezultate u poređenju sa *EnKF* i *EKF* zbog efikasnijeg pristupa proceni stanja. *EnKF* koristi ensemble uzoraka za aproksimaciju kovarijanse, ali njegova tačnost zavisi od broja uzoraka, što može dovesti do većih varijacija u rezultatima kada je taj broj ograničen. U mom eksperimentu, *EnKF* se pokazao slabijim zbog ovih varijacija, dok je *UKF*, koristeći sigma tačke, dao stabilnije i preciznije rezultate, što ga čini pouzdanijim u situacijama sa nelinearnim sistemima i ograničenim brojem uzoraka. Dok *EKF* koristi linearnu aproksimaciju sistema pomoću Jacobian matrica, *UKF* koristi sigma tačke za propagaciju tačnih nelinearnih transformacija stanja i kovarijance. Ovaj pristup omogućava bolju procenu distribucije stanja, smanjujući greške koje nastaju usled linearizacije.

6. LITERATURA

[1] T. Omeragic, J. Velagic, “Tracking of moving objects based on extended Kalman filter,” Faculty of Electrical Engineering, Sarajevo, 2020.

[2] S. Thrun, W. Burgard, D. Fox, “Probabilistic Robotics,” Cambridge, MA, MIT Press, 2005.

[3] P. S. Maybeck, “The Kalman Filter: An Introduction to Concepts,” Autonomous Robot Vehicles, 1990.

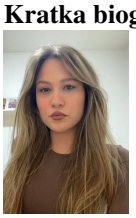
[4] G. A. Terejanu, “Extended Kalman Filter Tutorial,” Department of Computer Science and Engineering, University at Buffalo, 2024.

[5] S. J. Julier, J. K. Uhlmann, “A New Extension of the Kalman Filter to Nonlinear Systems,” *Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, pp. 182–193, 1997.

[6] M. Katzfuss, J. R. Stroud, C. K. Wikle, “Understanding the Ensemble Kalman Filter,” Department of Statistics, Texas A&M University, College Station, TX, USA; Department of Statistics, George Washington University, Washington DC, USA; Department of Statistics, University of Missouri, Columbia, MO, USA, 2016.

[7] `https://www.cvlibs.net/datasets/kitti/raw_data.php?type=city` (pristupljeno u avgustu 2024.)

Kratka biografija:



Anastasija Golić Rođena je u Jagodini 2000.god. Master rad na Fakultetu tehničkih nauka iz oblasti Računarstva i autmatike odbranila je 2024. god.
Kontakt: golicanastasija@gmail.com

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.8
DOI: https://doi.org/10.24867/30BE24Zivanovic

ОПТИМАЛНО УПРАВЉАЊЕ СИСТЕМИМА НЕЦЕЛОГ РЕДА УЗ АЛГЕБАРСКА ОГРАНИЧЕЊА

OPTIMAL CONTROL OF FRACTIONAL-ORDER SYSTEMS WITH ALGEBRAIC CONSTRAINTS

Николина Живановић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Овај рад је у потпуности посвећен проблемима оптималног управљања, код којих је процес описан фракционим диференцијалним једначинама, а компоненте вектора управљања трпе алгебарска ограничења. Анализом модела континуалног реактора и ХИВ инфекције, пружен је увид у примену теорије оптималног управљања. Примењен је поступак претварања нецелих диференцијалних једначина у обичне, ослањајући се на теорију Атанацковића и Станковића, и израчунато је управљање са алгебарским ограничењем користећи теорију Л. С. Понтрјагина. На крају, Хамилтонова функција је оптимизована нумерички. Теоријске основе потврђене су симулацијама.*

Кључне речи: *Системи нецелог реда, Експанзиона формула за фракциони извод, Оптимално управљање*

Abstract - *This work is entirely dedicated to the problems of optimal control where the process is described by fractional differential equations, and the control vector components are subject to algebraic constraints. Through the analysis of two models—a continuous reactor with mixing and a mathematical model of HIV infection—insights into the application of optimal control theory are provided. A multi-step procedure is applied to convert fractional differential equations into ordinary differential equations, relying on the theory of Atanacković and Stanković, and to compute control with algebraic constraints using L.S. Pontryagin's theory. Finally, the Hamiltonian function is numerically optimized. The theoretical foundations of this study are validated through numerical simulations.*

Keywords: *Fractional order systems, Expansion formula for fractional derivative, Optimal control*

1. УВОД

Проблеми оптималног управљања са ограничењима над управљачким променљивим су сложени чак и када је систем описан обичним диференцијалним једначинама. Иако постоје моћни нумерички алгоритми, израчунавање оптималне стратегије за нецеле изводе зависи од реда извода и постаје једноставније ако је он близак целом. У раду су разматрана два примера оптималног управљања са алгебарским ограничењима: модел континуалног реактора са мешањем (*Continuous Stirred Tank Reactor* - CSTR) и модел ХИВ инфекције, који је због своје сложености централни пример и главни предмет истраживања овог рада. Оба су илустративна са становишта нумеричке студије, а према сазнањима аутора модели које смо користили нису досад разматрани у фракционом облику у литератури.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био проф. др Зоран Д. Јеличић.

2. ОСНОВЕ ФРАКЦИОНОГ РАЧУНА

Фракциони рачун обухвата изводе и интеграле у нецелим степенима као што су $\frac{1}{2}$ или $1+2j$, познате као фракциони оператори. У овом раду акценат је на Риман-Љувиловом приступу, док је Капуто предложио алтернативу због недостатака тог приступа. Гринвалд-Летњиковљев метод заснован на коначним разликама је еквивалентан Риман-Љувиловом у одређеним условима и важан је за апроксимацију фракционих оператора у нумеричким алгоритмима. Поред "левих" оператора, у варијационом рачуну и оптималном управљању користе се и "десни" фракциони оператори. Заинтересованог читаоца упућујемо на литературу [1], где су дефинисани сви горе наведени оператори и изведене њихове особине. Риман-Љувилов извод дефинисан је као

$$ {}_0D_t^{\alpha} f = \frac{d^{n(\alpha)}}{dt^{n(\alpha)}} \, {}_0I_t^{n(\alpha)-\alpha} f. \tag{1} $$

За $0 < \alpha < 1$ следи

$$ {}_0D_t^{\alpha} f = \frac{1}{\Gamma(1-\alpha)} \frac{d}{dt} \int_0^t \frac{f(\tau)}{(t-\tau)^{\alpha}} d\tau. \tag{2} $$

Фракциони изводи су важни јер су дефинисани преко интеграла и узимају у обзир не само непосредну околину око тачке, већ и све претходне тачке.

2.1. Декомпозиција Атанацковића и Станковића

Декомпозиција је кључна за наш начин решавања проблема оптималног управљања системима описаним фракционим диференцијалним једначинама,

омогућавајући претварање у систем обичних диференцијалних једначина првог реда. Одабир довољног броја елемената у реду је важан и у нашем раду постигнут је емпиријски кроз симулације.
Представићемо експанзиону форму за фракциони извод реда $\alpha \in (0,1)$, предложену од Атанацковића и Станковића. Формула омогућава развој фракционог извода функције као линеарну комбинацију функције и њених момента, где су коефицијенти функције времена. Крајњи израз је представљен у [3, 2, 1].
Решење диференцијалне једначине са фракционим изводом своди се на интеграцију система диференцијалних једначина првог реда. За добијање задовољавајућих резултата често је довољно решити мали број тих једначина. Апроксимација експанзионе формуле је дата као

$$ {}_0D_t^{\alpha} f \approx \frac{A_{N_2}(\alpha)}{t^{\alpha}} f(t) + \sum_{i=2}^{N_2} \frac{B(\alpha,i)}{t^{i-1+\alpha}} \widetilde{W}_i(t), \qquad (3) $$

где су:

$$ A_{N_2}(\alpha) = \frac{1}{\Gamma(1-\alpha)} - \frac{1}{\Gamma(2-\alpha)\Gamma(\alpha-1)} \sum_{i=2}^{N_2} \frac{\Gamma(i-1+\alpha)}{(i-1)!}, \qquad (4) $$

$$ B(\alpha,i) = -\frac{1}{\Gamma(2-\alpha)\Gamma(\alpha-1)} \frac{\Gamma(i-1+\alpha)}{(i-1)!}, \qquad (5) $$

а помоћне променљиве $W_i(t)$ се добијају решавањем Кошијевог проблема:

$$ \dot{W}_i = -(i-1)t^{i-2} f(t), \qquad (6) $$

$$ W_i(0) = 0. \qquad (7) $$

Задовољавајуће апроксимације се постижу са N_2 у опсегу од 7 до 20. Када се ред фракционог извода приближи целобројној вредности, као за $\alpha = 0.1$ или $\alpha = 0.9$, или када је коефицијент уз фракциони члан мали, обично је потребно додати мањи број додатних чланова, често је довољно 3 до 5 додатних чланова за проблем оптималног управљања, [1].

3. ОПТИМАЛНО УПРАВЉАЊЕ УЗ АЛГЕБАРСКА ОГРАНИЧЕЊА

У овом раду фокусирали смо се на релејно (енг. *bang-bang*) управљање, које подразумева укључивање или искључивање управљања уместо постепене промене. Код релејног управљања, вредности управљачког вектора могу имати само две крајње вредности – горњу или доњу границу, уз коначан број прелазних тачака током интервала. Односно $\forall t$ важи

$$ u_{min} \leq u(t) \leq u_{max}. \qquad (8) $$

Релејно управљање се постиже уколико се критеријум оптималности формира тако да из њега произилази Хамилтонијан који линеарно зависи од управљачке променљиве. Један пример таквог критеријума оптималности, по угледу на [1], би био

$$ \jmath = \int_0^T \left\{ \frac{1}{2} \mathbf{x}^T(t) \mathbf{Q} \mathbf{x}(t) + R u(t) \right\} dt, \qquad (9) $$

где смо због једноставности претпоставили да се управљање врши путем једне променљиве, односно да је управљачка променљива скалар. Уколико је процес описан математичким моделом

$$ \dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)u(t); \quad \mathbf{x}(0) = \mathbf{x}_0, \qquad (10) $$

тада је по Понтрјагиновом принципу максимума Хамилтонијан

$$ \mathcal{H}(\mathbf{x}, \mathbf{p}, \mathbf{u}) = \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + R u + \mathbf{p}^T (\mathbf{A}\mathbf{x} + \mathbf{B}u), \qquad (11) $$

где са $\mathbf{p}$ означавамо придружену функцију (генералисани импулс). Даље се поступак оптимизације своди на минимизацију Хамилтонијана. Одатле произилазе канонске једначине које представљају потребне услове екстремума

$$ \dot{x_i} = \frac{\partial \mathcal{H}}{\partial p_i} \quad \wedge \quad \dot{p_i} = -\frac{\partial \mathcal{H}}{\partial x_i}, \qquad (12) $$

при чему је $i = 1, .., n$, где је n димензионалност вектора стања. Уз то имамо и природне граничне услове

$$ x_i(0) = \alpha_i \quad \wedge \quad p_i(T) = 0. \qquad (13) $$

Понтрјагинов принцип максимума налаже да је оптимално управљање оно које Хамилтоновој функцији (11) саопштава минимум. Хамилтонијан се може написати на следећи начин

$$ \mathcal{H}(\mathbf{x}, \mathbf{p}, \mathbf{u}) = \left(\mathbf{p}^T \mathbf{B} + R\right) u + \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{p}^T \mathbf{A} \mathbf{x}, \qquad (14) $$

одакле се дефинише оптимално управљање

$$ u^*(t) = \begin{cases} u_{min} & , \mathbf{p}^T\mathbf{B} + R > 0, \\ произвољно & , \mathbf{p}^T\mathbf{B} + R = 0, \\ u_{max} & , \mathbf{p}^T\mathbf{B} + R < 0. \end{cases} \qquad (15) $$

Поступак проналажења оптималног управљања подразумева решавање двотачкастог граничног проблема.

4. АЛГОРИТАМ НУМЕРИЧКЕ ОПТИМИЗАЦИЈЕ

Задатак који решавамо је двотачкасти гранични проблем, где су за променљиве стања познати почетни услови, а за придружене крајњи. Нумеричка метода коришћена у овом раду је метод напредназад *FBSM* (енг. *Forward-Backward Sweep Method*), који обухвата следеће кораке:

1. Дефинисање почетних вредности за променљиве стања, уз крајње вредности за придружене променљиве постављене на нулу. Почетна претпоставка за управљачке променљиве је нула.

2. У свакој итерацији, текућа вредност променљивих стања се одређује на основу претходне, што представља пропагацију унапред.

3. Пропагацијом уназад, у свакој интерацији се одређују текуће вредности придружених променљивих на основу њихове наредне вредности.

4. На основу ажурираних вредности променљивих стања и придружених променљивих, добијају се нове вредности за управљачке променљиве $u_{new}(t)$.

5. За континуална управљања, могуће је користити $u(t) = u_{new}(t)$, али то често није довољно за постизање конвергенције.

6. Проверава се конвергенција. Ако су све израчунате променљиве унутар одређене толеранције у односу на претходну итерацију, прихватамо их као конвергентне, а у супротном се враћамо на 2. корак.

На конвергенцију нумеричких решења за оптимално управљање утичу почетна претпоставка, критеријуми за конвергенцију, ажурирање управљања и тежински коефицијенти. У овом раду коришћена је почетна претпоставка $u \equiv 0$. Конвергенција се проверава на основу релативне разлике:

$$\frac{|u_{\text{updated}} - u_{\text{old}}|}{|u_{\text{updated}}|} \leq \epsilon, \tag{16}$$

где је ϵ релативна толеранција. Прилагођени критеријум је:

$$\epsilon \sum_{i=1}^{n} |u_{\text{updated}}(i\Delta t)| - \sum_{i=1}^{n} |u_{\text{updated}}(i\Delta t) - u_{\text{old}}(i\Delta t)| \geq 0. \tag{17}$$

Ажурирање управљања се врши као тежинска комбинација старог и новог управљања:

$$u_{\text{updated}} = \omega u_{\text{old}} + (1 - \omega) u_{\text{new}}, \tag{18}$$

где ω зависи од типа управљања и модела. Веће ω повећавају шансе за конвергенцију, али захтевају више итерација, док мање ω брже мењају управљање, али могу отежати конвергенцију.

5. РЕЗУЛТАТИ НУМЕРИЧКИХ СИМУЛАЦИЈА

У овом поглављу представићемо основне резултате нумеричких симулација оптималног управљања модела. Прво разматрамо оптимизацију четири управљачке променљиве хемијског реактора ради максимизације економског добитка. Систем обухвата четири хемијске реакције у изотермалном континуално мешаном реактору (*CSTR*) [4], са контролним променљивим протока и уносом енергије. У модел је унет фракциони извод реда $\alpha \in (0, 1)$ у једну од једначина стања како би се анализирао утицај параметра α на оптимизацију.

$$\max_{u_1, u_2} \jmath = \int_0^{0.2} \Big[5.8(qx_1 - u_4) - 3.7u_1 - 4.1u_2 + q(23x_4 + 11x_5 + 28x_6 + 35x_7) - 5u_3^2 - 0.099 \Big] dt \tag{19}$$

$$\dot{x}_1 = u_4 - qx_1 - 17.6x_1x_2 - 23x_1x_6x_3 \tag{20}$$

$$\dot{x}_2 = u_1 - qx_2 - 17.6x_2 - 146x_2x_3 \tag{21}$$

$$\dot{x}_3 + k_1 {}_0D_t^{\alpha} x_3 = u_2 - qx_3 - 73x_3^2 \tag{22}$$

$$\dot{x}_4 = 35.2x_1x_2 - 51.3x_3x_4 \tag{23}$$

$$\dot{x}_5 = qx_5 + 219x_7x_3 - 51.3x_2x_5 \tag{24}$$

$$\dot{x}_6 = 102.6x_3 - 146x_1x_6x_3 \tag{25}$$

$$\dot{x}_7 = qx_7 + 46x_1x_6x_3 \tag{26}$$

$$q = u_1 + u_2 + u_4 \tag{27}$$

$$0 \leq u_1 \leq 20 \tag{28}$$

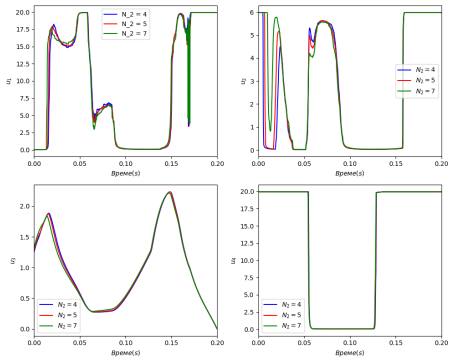
$$0 \leq u_2 \leq 6 \tag{29}$$

$$0 \leq u_3 \leq 4 \tag{30}$$

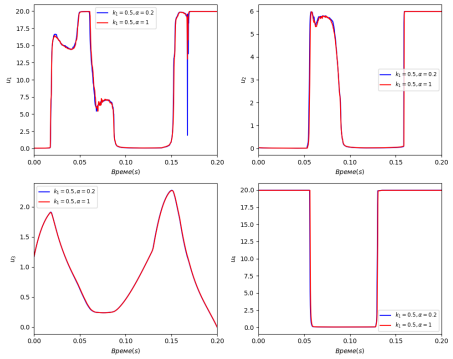
$$0 \leq u_4 \leq 20 \tag{31}$$

$$t_f = 0.2 \tag{32}$$

На слици 1 разматрамо случај $\alpha = 0.5$ и $k_1 = 1$, где не долази до значајнијих разлика у резултатима, осим код управљања $u_2(t)$.



Слика 1: *Оптималне управљачке променљиве за математички модел CSTR, када је параметар* $\alpha = 0.5$.



Слика 2: *Оптималне управљачке променљиве за математички модел CSTR, када је параметар* $N_2 = 4$.

Кроз тестирања, закључили смо да увођење оператора фракционог извода не доводи до великих промена у решењу, као што се види на слици 2. За симулације смо користили $\omega = 0.995$ и $\epsilon = 10^{-3}$. Управљање $u_4(t)$ је већ *bang-bang*, док $u_1(t)$ и $u_2(t)$ захтевају још итерација.

Други модел је фракциони модел ХИВ инфекције (по угледу на [5]), математички записан као

$$\dot{x} = \lambda - d_1 x - \frac{k_1(1-u_1)xv}{x+v} \tag{33}$$

$$\dot{s} = \frac{k_1(1-u_1)xv}{x+v} - d_2 s - k_2 s \tag{34}$$

$$\dot{y} = k_2 s - d_3 y - pyz \tag{35}$$

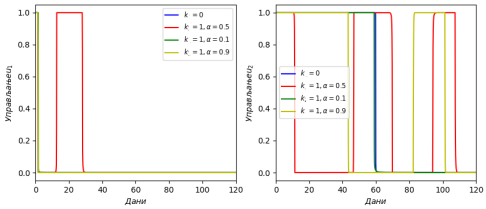
$$\dot{v} = a(1-u_2)y - d_4 v \tag{36}$$

$$\dot{z} + k_0 D_t^{\alpha} z = cyz - bz, \tag{37}$$

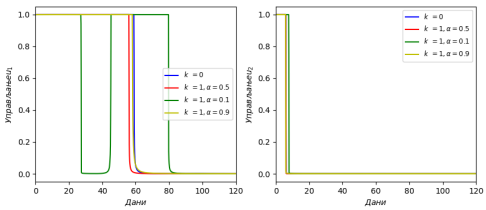
за познате почетне услове. Критеријумска функција, коју је потребно минимизовати, формулише се у складу са минимизацијом трошкова лечења и концентрација инфицираних ћелија

$$\jmath = \frac{1}{2}\int_0^{t_f} \{Q_1 s^2(t) + Q_2 y^2(t) + Q_3 v^2(t) + A_1 u_1(t) + A_2 u_2(t)\}\, dt, \tag{38}$$

где t_f представља трајање терапије. Параметри A_1, A_2 представљају губитке (финансијске и физиолошке) услед примене терапија. Оптималним управљањем ћемо постићи жељене резултате. Закључили смо да систем увек конвергира ка незараженом еквилибријуму, независно од комбинације параметара. Утицај параметара на управљање види се на сликама 3 и 4.



Слика 3: *Оптимално релејно управљање у функцији времена. Тежински параметри:* $A_1 = A_2 = 5, Q_1 = Q_2 = Q_3 = 1,\ N_2 = 5.$



Слика 4: *Оптимално релејно управљање у функцији времена. Тежински параметри:* $A_1 = 1, A_2 = 7, Q_1 = Q_2 = Q_3 = 1,\ N_2 = 5.$

6. ЗАКЉУЧАК

У овом раду истражен је проблем оптималног управљања за нелинеарне системе описане обичним и фракционим диференцијалним једначинама. Кроз примену на моделе континуалног хемијског реактора и ХИВ инфекције, показано је да фракциони модели омогућавају репрезентативније описе динамике сложених система. Истраживање примене оптималног управљања на фракционе моделе показало је да је алгоритам *Forward Backward Sweep Method* користан за решавање двотачкастих граничних проблема. Конвергенција у случају *bang-bang* управљања је била изазовнија у поређењу са континуалним управљањем, али подешавање параметара и већи број итерација могу довести до задовољавајућих решења. Поред тога у поступку оптимизације, примењен је принцип декомпозиције Атанацковића и Станковића који олакшава решавање диференцијалних једначина нецелог реда, трансформишући их у систем обичних диференцијалних једначина првог реда. Нумеричке симулације оптималног управљања показале су да комбинована терапија значајно побољшава резултате лечења, повећавајући ниво неинфицираних CD4+ Т-ћелија и смањујући концентрацију инфицираних и латентно инфицираних ћелија. Прекид терапије доводи до поновног активирања инфекције, што наглашава значај доследне терапије.

Литература

[1] M. R. Rapaić, "Optimalno i suboptimalno upravljanje klasom sistema sa raspodeljenim parametrima", PhD thesis, Fakultet tehničkih nauka u Novom Sadu, Novi Sad, 2011.

[2] Z. D. Jeličić and N. Petrovacki, "Optimality conditions and a solution scheme for fractional optimal control problems", *Structural and Multidisciplinary Optimization*, Vol. 38, No. 6, pp. 571-581, 2008, Springer, doi:10.1007/s00158-008-0307-7.

[3] T. Atanacković and B. Stanković, "An Expansion Formula for Fractional Derivatives and its Application", *Fractional Calculus and Applied Analysis*, Vol. 7, No. 3, pp. 365-378, 2004, Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, `http://eudml.org/doc/11253`.

[4] E. Balsa-Canto, et al., "Dynamic optimization of chemical and biochemical processes using restricted second-order information", *Computers & Chemical Engineering*, Vol. 25, No. 4-6, pp. 539-546, 2001, Elsevier, doi:10.1016/s0098-1354(01)00633-0.

[5] J. Danane and K. Allali, "Optimal control of an HIV model with CTL cells and latently infected cells", *Numerical Algebra, Control & Optimization*, Vol. 10, No. 2, pp. 207–225, 2020, doi:10.3934/naco.2019048.

Кратка биографија:

Николина Живановић рођена је у Зворнику 2000. год. Мастер рад на Факултету техничких наука из области Електротехнике и рачунарства - Оптимално управљање системима нецелог реда уз алгебарско ограничење одбранила је 2024. год.
Контакт: zivanovic.nikolina@uns.ac.rs

Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.438
DOI: https://doi.org/10.24867/30BE25SaboCibolja

IMPLEMENTACIJA GOSIP PROTOKOLA U PROGRAMSKOM JEZIKU RAST I ANALIZA UTICAJA TOPOLOGIJE MREŽE DISTRIBUIRANOG SISTEMA NA EFIKASNOST GOSIP PROTOKOLA

IMPLEMENTATION OF THE GOSSIP PROTOCOL IN THE RUST PROGRAMMING LANGUAGE AND ANALYSIS OF THE IMPACT OF NETWORK TOPOLOGY IN DISTRIBUTED SYSTEMS ON THE EFFICIENCY OF THE GOSSIP ALGORITHM

Andrea Sabo Cibolja, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad se bavi analizom Gosip protokola i njegove efikasnosti u različitim mrežnim topologijama . Gosip protokoli se koriste za efikasno širenje informacija u distribuiranim sistemima što dokazuje veliki broj dosadašnjih istraživanja, ali uticaj mrežne topologije na njihovu efikasnost ostaje nedovoljno ispitan. U ovom radu je implementirana varijanta Gosip algoritma sa periodičnim prosleđivanjem pristiglih informacija između čvorova distribuiranog sistema, uz analizu i unapređenje performansi samog Gosipa. Rešenje je implementirano u Rast programskom jeziku, jer uz svoju podršku za konkurentno programiranje i bezbedno upravljanje memorijom predstavlja odgovarajući alat za implementaciju distribuiranih algoritama.*

Ključne reči: *Gosip protokol, topologija mreže, Rast, širenje informacija, distribuirani sistem*

Abstract – *This paper analyzes the Gossip protocol and its efficiency in various network topologies. Gossip protocols are widely used for the efficient dissemination of information in distributed systems, yet the impact of network topology on their performance remains insufficiently studied. In this paper, a variant of the Gossip algorithm with periodic forwarding of received information between nodes in a distributed system is implemented, along with an analysis and optimization of the Gossip protocol's performance. The solution is implemented in the Rust programming language, which, with its support for concurrent programming and safe memory management, represents an appropriate tool for the implementation of distributed algorithms.*

Keywords: *Gossip Protocol, Network Topology, Rust, Information Dissemination, Distributed Systems*

1. UVOD

Motivacija iza ovog istraživanja leži u potrebi za analizom različitih varijanti komunikacije putem Gosip protokola u distribuiranim sistemima radi unapređenja efikasnosti razmene informacija. Primena ovih algoritama je usko povezana sa razvojem modernih mreža, gde nije više bilo jedino neophodno obezbediti efikasnu komunikaciju kao što je to bio slučaj u telefonskim mrežama i internetu [1].

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

Novije mreže zbog specifičnih namena zahtevaju jednostavnije algoritme, nemaju definisanu infrastrukturu, već pokazuju nepredvidivu dinamiku i suočavaju se sa ograničenjima resursa [1]. Upravo ova ograničenja dovode do Gosip algoritama, kao najboljeg mogućeg arhitektonskog rešenja. U radu je analiziran i implementiran Gosip algoritam sa periodičnim širenjem glasina uz analizu performansi druge varijante Gosip protokola. Rad pruža kompletnu analizu uticaja topologije mreže na efikasnost ove dve varijante protokola.

2. PREGLED I ISTORIJAT GOSIP ALGORITAMA

Iako distribuirani sistemi donose veliki broj prednosti svojom arhitekturom, Upravljanje ovim sistemima predstavlja kompleksan zadatak. Glavna mana ovih sistema je problem sa bezbednošću, dok prekidi mreže u važnijim delovima mogu značajno da poremete rad sistema. Kako bi se opravdalo uvođenje ove infrastrukture u računarski sistem, komunikacija među članovima mora da bude što efikasnija. U nastavku je opisana istorija, podele Gosip algoritama i primeri primene, dok je u četvrtom poglavlju rada detaljno opisan način funkcionisanja Gosip algoritma.

2.1. Istorijat Gosip algoritama

Gossip protokol, poznat i kao epidemiološki protokol, je decentralizovana pir-tu-pir (engl. *Peer-to-peer*) tehnika za prenos poruka u distribuiranim sistemima [2]. Ključni koncept je da čvorovi nasumično šalju poruke drugima, što omogućava širenje informacija sa velikom verovatnoćom. Prvi put je opisan 1972. godine [3] , a značajno je unapređen 1987. u radu Demersa i saradnika, koji su ga koristili za održavanje replikovanih baza podataka [2, 4].

2.2. Vrste Gosip algoritama

Gosip algoritmi se dele se na dve glavne kategorije: anti entropija algoritmi i širenje glasina [5]. Anti-entropija algoritmi dele celu bazu podataka u svakom ciklusu kako bi se smanjila entropija, dok algoritmi bazirani na širenju glasina šalju samo nova ažuriranja, što smanjuje mrežni saobraćaj [5].

2.3. Strategije za širenje Gosipa

PUŠ (engl. *PUSH*) strategija podrazumeva da infektivni čvorovi šalju ažuriranja susedima, dok kod PUL (engl. *PULL*) strategije čvorovi aktivno traže ažuriranja. PUŠ-PUL (engl. *PUSH-PULL*) kombinuje oba pristupa, omogućavajući istovremeno slanje i traženje novih informacija [5].

2.4. Upotreba Gosip algoritama

Gosip algoritmi se koriste za održavanje članstva u klasterima, postizanje konsenzusa i detekciju kvarova. Njihova primena je rasprostranjena u sistemima kao što su Apače Kasandra (engl. *Apche Cassandra*), Riak i Kubernetes, gde omogućavaju sinhronizaciju čvorova i upravljanje resursima [4]. U blokčejn tehnologija, Gosip se koristi za distribuciju transakcija i blokova.

3. PREGLED UPOTREBLJENIH TEHNOLOGIJA

U ovom istraživanju korišćen je programski jezik Rast (engl. *Rust*) za implementaciju Gosip algoritma. Rast kombinuje visoke performanse sa bezbednošću memorije, sprečavajući greške poput dereferenciranja pokazivača koji ne pokazuju ni na šta (engl. *null pointers*) [6]. Njegova podrška za konkurentnost i asinhrono programiranje čini ga pogodnim za implementaciju algoritama u distribuiranim sistemima [7]. Serde radni okvir Rasta omogućava efikasnu serijalizaciju i deserijalizaciju podataka, ključnih za razmenu između čvorova [7]. Podržava formate poput Džejsona (engl. *JSON*), Bisona (engl. *BSON*) i Jamla (engl. *YAML*), a izgrađen je na moćnom Rastovom sistemu osobina (engl. *Traits*) [6]. Asinhrono programiranje u Rastu se postiže pomoću asink (engl. *async*) biblioteke. Moduli `std::thread` i `std::sync` omogućavaju kreiranje niti, sinhronizaciju, i komunikaciju pomoću kanala [6]. Muteks se koristi za bezbedan pristup deljenim podacima [6].

Rezultati implementiranog algoritma su upoređeni sa algoritmom implementiranim u Rubi programskom jeziku (engl. *Ruby*), koji je poznat po jednostavnosti i fleksibilnosti. Rubi je dinamičan, interpretirani jezik sa podrškom za više programskih paradigmi [8]. Ima podršku za kreiranje niti, mutekse i monitore. Jednostavno asinhrono programiranje u upoređivanoj implementaciji je postignuto pomoću povratnih funkcija i RPC (engl. *RPC – Remote Procedure Call*) poziva. Iako nije najbrži, njegova jednostavna sintaksa i fleksibilnost olakšavaju razvoj [8].

Testiranje je obavljeno na Maelstrom platformi, koja simulira realistične scenarije u distribuiranim sistemima [9]. Omogućava testiranje prototipova algoritama u kontrolisanim uslovima, simulirajući mrežne uslove poput kašnjenja, grešaka i mrežnih prekida.

4. PRINCIP RADA GOSIP ALGORITMA

Osnovni koncept Gosip algoritma leži u distribuiranom načinu širenja informacija u mreži, gde svaki čvor komunicira sa odabranim podskupom drugih čvorova i razmenjuje podatke [4, 5]. Ovaj proces se ponavlja u više rundi (gosipa), što omogućava da informacije progresivno stignu do svih čvorova u mreži. Sistem ne zavisi od centralizacije, što ga čini robusnim u uslovima gde čvorovi mogu da otpadnu ili da se priključe novi.

4.1. Koraci Gosip algoritama

Prvi korak je da čvorovi budu svesni svih članova klastera, nakon toga je neophodno definisati inicijalni skup suseda, koji u bilo kom trenutku može da bude promenjen. Nakon toga čvor pristigle ili generisane podatke može da deli bilo periodično ili odmah pri pristizanju (PUŠ lagoritam). Ovaj korak obezbeđuju da podaci stignu do svih čvorova u sistemu. Kod periodičnog Gosip algoritma ključni parametar je Gosip interval, koji određuje koliko često čvorovi šalju poruke, kontrolišući time brzinu i efikasnost širenja informacija.

4.2. Efikasnost i latencija

Iako Gosip algoritmi efikasno rasprostranjuju informacije, neizbežna je određena latencija u sistemu. Latencija zavisi od mrežnih uslova, topologije mreže i broja skokova potrebnih da poruka stigne do svih čvorova. Sa povećanjem broja čvorova latencija može dosta da poraste, ali je bitno da se i dalje postiže konvergencija.

4.3. Značaj topologije

Mrežna topologija direktno utiče na performanse Gosip algoritma. U radu su analizirane neke od značajnijih topologija, to su linearna, totalna, grid i topologija stabla topologije. Linearna topologija, na primer, ima ograničen broj putanja za prenos podataka, što može povećati latenciju. Grid topologija nudi više putanja i otpornija je na kvarove, totalna je najbrža, dok topologije stabla mogu omogućiti brže širenje informacija uz bolje korišćenje resursa.

4.4. Detekcija i ispravljanje grešaka

Gosip algoritmi koriste signale „otkucaje srca“ za detekciju nefunkcionalnih čvorova [10]. Ako susedi čvora ne prime signal od čvora tokom određenog vremenskog intervala, čvor se proglašava mrtvim i sistem se prilagođava da bi obezbedio nastavak rada. Ovi algoritmi takođe mogu koristiti mehanizme za balansiranje opterećenja, gde čvorovi razmenjuju informacije o svom trenutnom stanju i ravnomerno raspoređuju zadatke [11].

5. IMPLEMENTACIJA GOSIP ALGORITMA U PROGRAMSKOM JEZIKU RAST

U ovom radu implementirana je varijanta Gosip protokola sa periodičnim slanjem poruka susedima u Rast programskom jeziku. U nastavku su navedene glavne funkcionalnosti sa fokusom na primenu jezika Rast u implementaciji.

```
let jh= std::thread::spawn(move ||{
    let stdin = std::io::stdin().lock();
    let stdin = stdin.lines();


    for line in stdin {
        let line = line.context("Maelstrom input from STDIN could not be read")?;
        let input: Message<P> = serde_json::from_str(&line)
            .context("Maelstorm input from STDIN could not be deserialized")?;

        if let Err(_) = tx.send(Event::Message(input)){
            return Ok::<_, anyhow::Error>(());
        }
    }
let _ = tx.send(Event::EOF);
Ok(())
});
```

Slika 5. *Korišćenje niti za obradu poruka sa standardnog ulaza*

Prihvatanje i obrada poruka koje stižu preko standardnog ulaza postignnuti su upotrebom Serde biblioteke, kao i upotrebom niti, kanala i muteksa iz odgovarajućih biblioteka u Rastu (slika 5). Prilikom prijema, vrši se konverzija u odgovarajuće strukture podataka iz pristiglog Džejson formata. U slučaju uspešne deserijalizacije, poruke se šalju preko kanala pošiljaoca kao događaj tipa `Event:Message`.

Nakon toga kanal primalac prihvata događaj i prosleđuje ga `handleMessage` funkciji čvora (slika 6).

```
for input: Event<P> in rx {
    node.handleMessage(input, output: &mut stdout) Result<(), Error>
```

Slika 6. *Kanal primalac prosleđuje poruku funkciji čvora*

Inicijalizacija čvora omogućeno je pristizanjem inicijalizacione poruke, čvor dobija svoj identifikator i svest o ostalim čvorovima mreže (slika 7). Takođe će se kreirati nit koja će periodično da signalizira čvoru da treba da odradi gosip rundu slanjem `InternalGossip` poruka.

```
fn node_initialization(
    _state: (),
    init: Init,
    tx: std::sync::mpsc::Sender<Event<Payload>>,
) -> anyhow::Result<Self> {
    std::thread::spawn(move || {
        loop {
            std::thread::sleep(dur: Duration::from_millis(30));
            if let Err(_) = tx.send(Event::InternalGossip) {
                break;
            }
        }
    });

    let mut acknowledged_messages: HashMap<String, HashSet<usize>> = HashMap::new();

    for node_id: String in init.node_ids {
        acknowledged_messages.insert(k: node_id, v: HashSet::new());
    }

    Ok(Self {
        node: init.node_id,
        id: 1,
        messages: HashSet::new(),
        known: acknowledged_messages,
        neighborhood: Vec::new(),
    })
} fn node_initialization
```

Slika 7. *Inicijalizacija čvora*

Prijem novih informacija, **ažuriranje liste suseda**, **gosip runde** se realizuju kroz `handleMessage` funkciju čvora, koja obrađuje različite tipove poruka kao što je prikazano na slici 8. Čvorovi šalju susedima samo one poruke za koje nije poznato da su ih susedi primili. Optimizacija je postignuta slanjem određenog procenta susedu već poznatih poruka, kako bi se sused obavestio da čvor poznaje te informacije. Ovim se smanjuje saobraćaj i izbegava beskonačno slanje istih poruka.

```
Event::Message(input) => {
    let mut reply = input.create_reply(Some(&mut self.id));
    match reply.body.content {
        Content::Gossip { seen } => {
            self.known
                .get_mut(&reply.dest)
                .expect("got gossip from unknown node")
                .extend(seen.iter().copied());
            self.messages.extend(seen);
        }
        Content::Broadcast { message } => {
            self.messages.insert(message);
            reply.body.content = Content::BroadcastOk;
            reply.send(&mut *output).context("reply to broadcast")?;
        }
        Content::Read => {
            reply.body.content = Content::ReadOk {
                messages: self.messages.clone(),
            };
            reply.send(&mut *output).context("reply to read")?;
        }
        Content::Topology { mut topology } => {
            self.neighborhood = topology
                .remove(&self.node)
                .unwrap_or_else(|| panic!("no topology given for node {}", self.node));
            reply.body.content = Content::TopologyOk;
            reply.send(&mut *output).context("reply to topology")?;
        }
        Content::BroadcastOk | Content::ReadOk { .. } | Content::TopologyOk => {}
    }
}
```

Slika 8. *Rukovanje različitim događajima i porukama u sistemu*

Periodični algoritam je sam po sebi otporan na otkaze. U slučaju da je neki čvor nedostupan, slanje istih informacija će se ponavljati, sve dok čvor ne signalizira pošiljaocu da je postao svestan tih informacija.

Ova implementacija služi kao demonstracija efikasnosti Gosip algoritama u distribuiranim sistemima i pruža mogućnost za dalje usavršavanje pre primene na stvarnim mrežama.

6. TESTIRANJE IMPLEMENTACIJE GOSIP ALGORITMA NA MALESTROM PLATFORMI

Implementiran je periodični Gosip algoritam, a njegove performanse su upoređene sa performansama PUŠ Gosip algoritma koji je implementiran u Rubiju u različitim topologijama mreže. Rezultati su prikazani u tabelama 1 i 2.

Pomenuti PUŠ algoritam funkcioniše tako što čvor automatski po prijemu prosleđuje poruku svojim susedima [9]. Ovaj pristup smanjuje kašnjenje poruka, ali generiše veći broj poruka u mreži. U uslovima prekida mreže, PUŠ algoritam u Rubiju ima ugrađenu otpronost kroz mehanizam povratnih poziva (engl. *callbacks*) i udaljenih procedura (engl. *RPC* – skr. *Remote Procedure Calls*) [9]. Ako primalac nije dostupan usled prekida u mreži, poruka će biti ponovo poslata.

Algoritmi su testirani na mreži od 25 čvorova u trajanju od 30 sekundi, uz frekvenciju slanja od 100 poruka u sekundi i kašnjenje od 100 milisekundi po poruci između čvorova.

Detaljnom analizom uticaja topologije na ove dve varijante algoritm PUŠ algoritam je pokazao manje kašnjenje u odnosu periodični algoritam u svim topologijama, međutim on dovodi do značajno većeg opterećenja mreže.

Linearna topologija stvara najmanje zagušenje mreže, ali uzrokuje najveće kašnjenje. Analizirajući rezultate, topologije stabla (engl. *Tree*) su donele značajna poboljšanja u brzini propagacije poruka uz neznatno zagušenje mreže u odnosu na linearnu topologiju u oba algoritma. Konkretno najbolje performanse pruža topolgija stabla sa četvoro dece po čvoru („tree4“).

Analiziranjem pomenutih algoritama u mreži sa periodičnim prekidima (tabela 1), uočava se da je latencija znatno više uticala na PUŠ algoritam. U topologiji stabla je latencija znatno više porasla kod PUŠ algoritma, vreme kašnjenja je približno izjednačeno sa vremenom kašnjenja periodičnog algoritma, dok je opterećenje mreže i dalje znatno veće. Dolazi se do zaključka da su periodični algoritmi efikasniji u mreži sa prekidima

Testiranje Gosip algoritma pokazalo je da je izbor algoritma i mrežne topologije ključan za ostvarivanje optimalnih performansi. Dok periodični algoritmi smanjuju mrežno zagušenje, PUŠ algoritmi omogućavaju brže širenje poruka. Najbolji algoritam za neki sistem je onaj koji odgovara specifičnim zahtevima mreže, što ukazuje na to da ne postoji univerzalno rešenje za sve distribuirane sisteme

7. ZAKLJUČAK

Tabela 1. *Vrednosti merenja sa različitim topologijama*

Gosip algoritam	Topologija	Broj poruka po operaciji	Srednje kašnjenje (milisekunde)	Maksimalno kašnjenje (milisekunde)	Broj poruka (klijent)	Brodkast poruke	Poruke za čitanje	Broj poruka (serveri)	Broj poruka na serverima po brodkastu	Zastarele poruke	Stabilne poruke
Periodični	Linearna	2,25	4704	7192	5944	1420	1502	6571	4,63	1415	1420
PUŠ	Linearna	24,03	1600	2409	5876	1446	1442	69408	50	1446	1446
Periodični	Grid	3,86	1526	2376	5896	1465	1433	11200	7,65	1465	1465
PUŠ	Grid	55,38	464	808	5698	1384	1415	155008	112	1383	1384
Periodični	Totalna	42,5	260	740	4483	1099	1104	89083	81,05	1059	1099
PUŠ	Totalna	/	/	/	/	/	/	/	/	/	/
Periodični	„tree2”	2,21	1817	2453	6006	1465	1488	6528	4,45	1465	1465
PUŠ	„tree2”	23,48	585	819	5982	1439	1502	69072	48	1433	1439
Periodični	„tree3”	2,27	1301	1752	5906	1427	1476	6576	4,6	1419	1427
PUŠ	„tree3”	23,67	434	662	5922	1436	1475	68928	48	1433	1436
Periodični	„tree4”	2,28	1102	1420	5878	1418	1471	6576	4,64	1418	1418
PUŠ	„tree4”	23,23	376	523	6114	1455	1552	69840	48	1448	1455

Tabela 2. *Vrednosti merenja sa periodičnim prekidima mreže*

Gosip algoritam	Topologija	Broj poruka po operaciji	Srednje kašnjenje (milisekunde)	Maksimalno kašnjenje (milisekunde)	Broj poruka (klijent)	Brodkast poruke	Poruke za čitanje	Broj poruka (serveri)	Broj poruka na serverima po brodkastu	Zastarele poruke	Stabilne poruke
Periodični	Linearna	2,23	6804	17053	6008	1485	1469	6576	4,47	1479	1485
PUŠ	Linearna	27,11	2627	12348	5844	1415	1457	77865	55	1415	1415
Periodični	„tree4”	2,25	4826	14436	5938	1460	1459	6576	4,5	1459	1460
PUŠ	„tree4”	32,95	4120	12533	5802	1438	1413	93942	65,33	1432	1438

8. LITERATURA

[1] Devavrat Shah, “Gossip algorithms“, *Now publishers*, Massachusetts Institute of Technology, Cambridge, 2009.

[2] A. Demers., D. Greene, C. Hauser, W. Irish, J. Larson, “Epidemic algorithms for replicated database maintenance”, Xerox Palo Research Center, 1987.

[3] Brenda Baker, Robert Shostak, “Gossips and Telephones”, Jun 1972.

[4]https://managementfromscratch.wordpress.com/2016/04/01/introduction-to-gossip/ (pristupljeno u septembru 2024.)

[5] https://highscalability.com/gossip-protocol-explained/ (pristupljeno u septembru 2024.)

[6] https://doc.rust-lang.org/book/title-page.html (pristupljeno u septembru 2024.)

[7] V. Kumar, “Rust in Distributed System Environment”, Jul 2023.

[8] https://ruby-doc.com/docs/ProgrammingRuby/
(pristupljeno u septembru 2024.)

[9] https://github.com/maelstrom-software/maelstrom

(pristupljeno u septembru 2024.)

[10] G. Pandey, “Gossip Protocol: Failure Detection in Distributed Systems”, Avgust 2023.

[11] A. Rowstron, P. Druschel, “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, *IFIP/ACM Middleware 2001*, Heidelberg, Novembar 2001.

Kratka biografija:

Andrea Sabo Cibolja rođena je u Subotici 1999. godine. Završila je Gimnaziju „Svetozar Marković“ u Subotici 2018. godine. Diplomirala je 2022. godine na Fakultetu tehničkih nauka u Novom Sadu, na smeru Računarstvo i automatika. Upisala je master iz iste oblasti.

kontakt: andrea.saboc@gmail.com

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.4
DOI: https://doi.org/10.24867/30BE26Milisic

ВЕБ АПЛИКАЦИЈА ЗА ВИЗУЕЛИЗАЦИЈУ ОБУЧАВАЊА НЕУРОНСКЕ МРЕЖЕ У РЕАЛНОМ ВРЕМЕНУ

A WEB APPLICATION FOR REAL-TIME VISUALIZATION OF NEURAL NETWORK TRAINING

Дане Милишић, *Факултет техничких наука, Нови Сад*

Област – Примењене рачунарске науке и информатика

Кратак садржај – *Овај рад се бави развојем веб апликације за визуелизацију обучавања неуронске мреже у реалном времену. Циљ овог рада је истражити да ли је могуће уз помоћ актуелних веб технологија, попут Риект ЈС библиотеке, реализовати алат за праћење обучавања неуронске мреже у реалном времену. Решење имплементирано у овом раду има могућност коришћења различитих библиотека за испуњивање истих захтева, како би се могло извршити тестирање њихових перформанси и одабрати најбоље решење за специфичне захтеве.*

Кључне речи: *ЈаваСкрипт, Риект ЈС, Сокет ИО, Пајтон, Неуронске мреже, Веб апликација*

Abstranct – *This paper deals with development of web application for real-time visualization of neural network training. Goal of this paper is researching if it is possible to develop tool for tracking of neural network training in real-time, using current web technology, such as React.JS. Application developed in this research has different solutions for same problems. Therefore, this application can be used for testing different solutions and selecting the best one for different segments of application.*

Keywords: *JavaScript, React.JS, Socket.IO, Python, Neural networks, Web application*

1. УВОД

Тренутно, колико је аутору познато, не постоји алат за визуелизацију и праћење рада неуронске мреже који је реализован као веб апликација. Проблем који би решио такав алат огледа се у томе да у случају да корисник жели да сагледа структуру или да прати обучавање неке велике неуронске мреже, морао би модел те неуронске мреже да поседује на својој машини. Такође, корисник не би био приморан да поседује разне технологије које захтевају алати развијени као десктоп апликација, али би се такође и заобишао процес у ком корисник треба да пронађе адекватан модел. Највећи потенцијални проблем на који се наилази приликом реализације решења веб апликације за визуелизацију обучавања неуронске мреже у реалном времену, јесу нешто слабије перформансе најпопуларнијих веб технологија, када је реч о обради великог броја података. Из претходно поменутог разлога, решење описано у овом раду садржи више решења за различите сегменте у апликацији, како би се омогућило тестирање различитих технологија.

Рад има следећу структуру. У другом поглављу се описује развој веб апликације за визуелизацију неуронске мреже у реалном времену, где су представљене коришћене технологије и структура апликације. У трећем поглављу представљена је имплементација апликације, пропраћена сликама корисничког интерфејса. У четвртом поглављу представљена је анализа перформанси и употребљивости система. Пето поглавље представља закључак, док је у шестом наведена литература. На самом крају рада налази се биографија аутора.

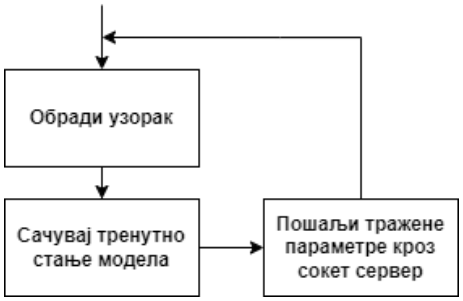
2. РАЗВОЈ ВЕБ АПЛИКАЦИЈЕ ЗА ВИЗУЕЛИЗАЦИЈУ ОБУЧАВАЊА НЕУРОНСКЕ МРЕЖЕ У РЕАЛНОМ ВРЕМЕНУ

Задатак овог рада, представљен у наслову, може се поделити у три целине, а то су: визуелизација у веб апликацији [1], комуникација у реалном времену и обучавање неуронске мреже. Идеја је да поменуте три целине буду развијене као три апликације које нису директно зависне једне од других. С обзиром на то да је неопходно реализовати комуникацију у реалном времену између веб апликације и серверске апликације са неуронском мрежом, потребно је креирати сокет (енгл. *socket*) сервер. Сокет сервер има улогу да отвара сокете између клијената или клијента и сервера, који омогућава комуникацију у реалном времену, и за разлику од традиционалне комуникације преко ХТТП (енгл. *Hypertext Transfer Protocol*) захтева, овај вид комуникације остаје отворен. Развој сокет сервера је реализован уз помоћ Сокет ИО (енгл. *Socket.IO*) технологије [2], која осим што омогућава развој сервера и клијената, такође и омогућава комуникацију између апликација које су развијене у различитим технологијама. Комуникација преко сокет сервера се огледа у томе да се клијенти првенствено повежу на

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

сервер, а затим док неки клијенти емитују догађај, који носи неку поруку, други клијенти се претплаћују на такав догађај. Када клијент емитује неки догађај, сервер тај догађај пролеђује клијентима који су претплаћени на њега.

Развој неуронске мреже представља велик посао, међутим уз помоћ библиотеке Тенсорфлоу (енгл. *Tensorflow*) [3] у склопу програмског језика Пајтон (енгл. *Python*) [4] процес је у великој мери поједностављен. Неуронска мрежа, такође, мора бити адаптирана тако да врши комуникацију уз помоћ сокет сервера, али и да складишти податке, након сваког узорка приликом обучавања. Начин на који се неуронска мрежа обучава јесте да након сваког обрађеног узорка, чув тренутно стање свих тежина и бајес (енгл. *bias*) вредности у моделу, а затим одређене вредности шаље корисницима који су се на исте претплатили (слика 1).

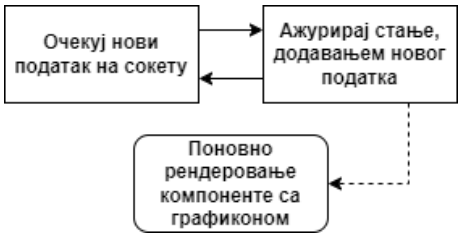


Слика 1. *Процес обучавања неуронске мреже уз комуникацију и чување стања*

Како би се реализовала визуелизација у веб апликацији која подржава промене у реалном времену, неопходно је употребити неку библиотеку која користи виртуелни ДОМ (енгл. *Document Object Model*). Пример такве библиотеке је Риект ЈС (енгл. *React.js*) [5], која такође подржава интеграцију са другим технологијама, попут библиотека за изградњу графова и графикона [6].

Кориснички део апликације пре свега има задатак да представи структуру неуронске мреже уз помоћ графа, који такође треба да подржава интеракцију са корисником. Неке од библиотека које омогућавају интеграцију оваквих врста графова у веб апликацију су Ситоскејп ЈС (енгл. *Cytoscape.js*) [7] и Сигма ЈС (енгл. *Sigma.js*) [8]. Интеракција са графом подразумева зум и пан, који су подразумевано имплементирани, филтер податак који се приказује, али и дефиницију различитих догађаја. Филтер се односи на податке који су приказани на графу и захтева да се после примене учита граф са новим подацима. Поновно учитавање се лако реализује у Риект ЈС библиотеци, тако што се подаци чувају у стањима, а променом стања све компоненте које су повезане са истим се поново учитавају. Такође, коришћењем Ситоскејп ЈС библиотеке могуће је дефинисати акције које прате одређене догађаје и на тај начин се омогућава директна интеракција са корисником. Специфично у овој апликацији, кориснику треба омогућити да притиском на одређену грану или чвор на графу добије приказ промена тежине, тј. бајес вредности кроз време, за одабрани елемент на графу.

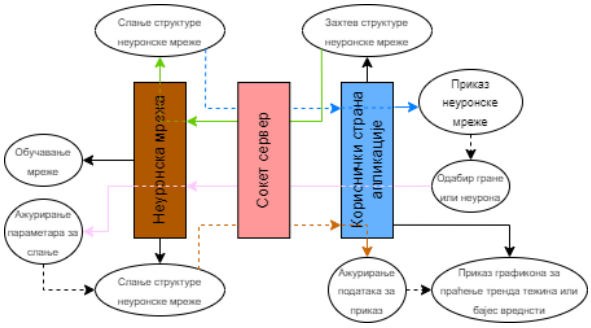
Како би се приказала промена поменутих вредности кроз време, неопходно је у апликацију интегрисати линијски графикон. Неке од библиотека које подржавају линијске графиконе су Чарт ЈС (енгл. *Chart.js*) [9], Ричартс (енгл. *Recharts*) [10] и Виктори (енгл. *Victory*) [11]. С обзиром на то да подаци на графикону подлежу константним променама, неопходно их је као и у случају са графом дефинисати уз помоћ стања у склопу Риект ЈС библиотеке. Такође, ствара се потреба за процесом који ће у позадини да очекује новопристигле податке и исте додаје у поменуто стање, како би били приказани на графикону (слика 2).



Слика 2. *Дијаграм процеса који преузима податке из сокета и смешта их на графикон*

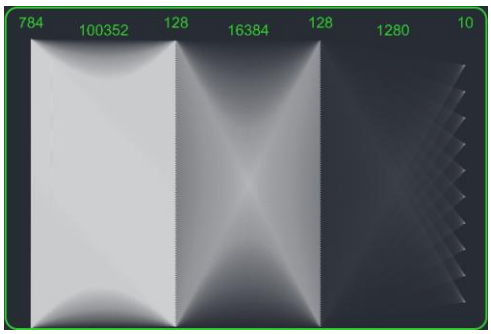
3. РЕШЕЊЕ

Решење веб апликације за визуелизацију обучавања неуронске мреже у реалном времену се састоји из три сегмента, као што је то описано у претходном поглављу. Кориснички део апликације и неуронска мрежа комуницирају у реалном времену уз помоћ сокет сервера, док у међувремену обе стране омогућавају извршавање својих функционалних захтева (слика 3).



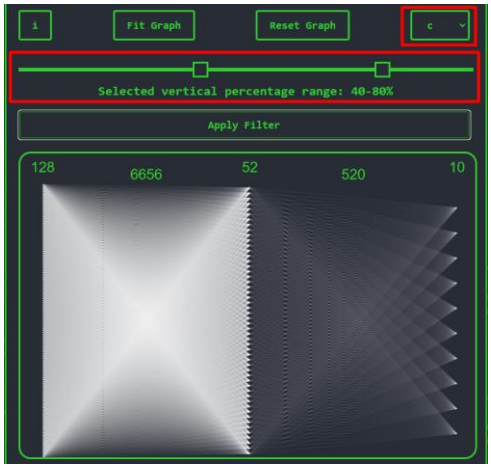
Слика 3. *Дијаграм функционалних захтева и комнуникације у склопу система*

Приказ неуронске мреже имплементиран је уз помоћ Ситоскејп библиотеке. Неуронска мрежа је приказана кроз граф, који такође садржи и лабеле које информишу корисника о броју неурона и грана у сваком слоју неуронске мреже (слика 4).



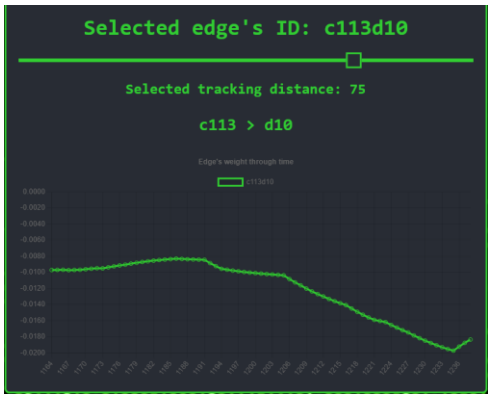
Слика 4. *Приказ неуронске мреже на корисничкој страни апликације*

Кориснику је омогућена интеракција са графом у виду зума, пана и филтрирања података. Корисник може да одабере слој који ће бити приказан као централни, а уз њега ће се, ако постоје, приказати претходни и наредни слој. Такође, у склопу одабраног слоја, може да бира који део ће бити приказан. Филтрирање ажурира и лабеле за приказ броја неурона и грана у сваком слоју (слика 5).



Слика 5. *Филтрирање података који ће се приказати на графу*

Корисник, такође, има могућност да одабере грану или неурон за коју ће пратити тренд промене тежине, тј. бајес вредности. Тренд је приказан преко линијског графикона, који је такође подложан променама у реалном времену које одговарају обучавању неуронске мреже (Слика 6).



Слика 6. *Приказ тренда промене тежине за одабрану грану*

4. АНАЛИЗА ПЕРФОРМАНСИ И УПОТРЕБЉИВОСТИ СИСТЕМА ЗА ВИЗУЕЛИЗАЦИЈУ

Анализа перформанси овог система подразумева испитивање како се систем понаша када су у оптицају велике количине података. Када је у питању обучавање неуронске мреже узорак по узорак са праћењем долази до значајног успоравања. Обучавање неуронске мреже на уобичајен начин са 60 хиљада података у три епохе траје свега 10 до 15 секунди, међутим када се то промени на обучавање узорак по узорак уз праћење, брзина обучавања износи 150 узорака по минути.

Када је реч о перформансама сокет сервера, очекивано, комуникација се извршава у реалном времену.

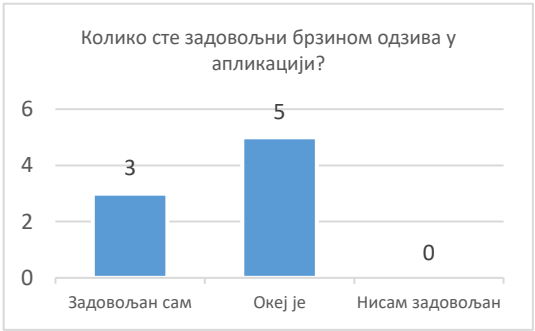
Део апликације, чије су перформансе највише погођене растом броја података јесте корисничка страна. Пре свега учитавање структуре неуронске мреже се знатно успорава са растом броја неурона и грана, које, такође, утиче и на време одзива (Табела 1).

Број елемената	Време учитавања	Време одзива
≈ 1000	100ms	*real-time*
≈ 10000	1000ms	100ms
≈ 100000	8000ms	3000ms

Табела 1. *Време учитавања и одзива у односу на број елемената*

Када је реч о анализи употребљивости, пре свега је спроведено тестирање са корисницима, који су по завршетку тестирања попуњавали анкету, где су имали прилику да оцене апликацију и њену употребљивост. Оцене за неке од најзначајнијих сегмената приказане су на сликама 7 и 8.

Слика 7. *Расподела оцена квалитета интеракције у апликацији*



Слика 8. *Однос оцена у вези са брзином одзива у апликацији*

Такође, корисници који су имали искуства са алатима за визуелизацију неуронских мрежа су одговарали на

питање да ли је могуће да овакав алат замени постојеће, који су развијени као десктоп апликације и одговори су били подељени. Два корисника су се изјаснили да би радије наставили са коришћењем постојећих алата, док је један корисник рекао како види перспективу у оваквом типу алата.

5. ЗАКЉУЧАК

У склопу овог рада представљена је веб апликација за визуелизацију обучавања неуронске мреже у реалном времену, која је развијена у три дела: неуронска мрежа, сокет сервер и кориснички интерфејс. Сокет сервер је развијен у ЈаваСкрипт програмском језику, кошћењем Сокет ИО технологије за комуникацију у реалном времену. Неуронска мрежа је развијена у Пајтон програмском језику користећи Тензерфлоу библиотеку, али и Сокет ИО Пајтон клијент библиотеку како би се омогућила конекција на сокет сервер. Кориснички интерфејс је развијен у склопу библиотеке Риект ЈС, а коришћене су и библиотеке Ситоскејп, Чарт ЈС, Ричартс и Виктори за визуелизацију података.

Главна мотивација за развијање оваквог алата као веб апликације јесте омогућавање лакше и шире примене алата, без нужне потребе да се неуронска мрежа налази на истој машини где и алат, што такође утиче на то да корисникова машина не мора бити натпросечних перформанси.

Апликација је развијена на такав начин да корисницима пружа могућност бирања различитих алата за визуелизацију, како би се подстакло тестирање различитих алата и на тај начин, у перспективи, дошло до идеалног алата сачињеног од више мањих.
Приликом развоја апликације, највећи непревазиђени изазов јесте нешто слабије перформансе корисничког интерфејса, кад су у оптицају веће количине података које се требају приказати. Стога, први наредни кораци у развоју овог алата били би пре свега тражење библиотеке за визуелизацију графа која је боље оптимизована, а вероватно и замене главне библиотеке Риект ЈС, за неку са бољим перформансама, као што су Вју (енгл. *Vue*) или Лептос (енгл. *Leptos*).

6. ЛИТЕРАТУРА

[1] Fundamentals of Data Visualization,
урл: https://clauswilke.com/dataviz/
[Приступљено 15.9.2024.]

[2] Socket.IO – Official website,
урл: https://socket.io/
[Приступљено 15.9.2024.]

[3] TensorFlow – Official website,
урл: https://www.tensorflow.org/
[Приступљено 15.9.2024.]

[4] Python – Official website,
урл: https://www.python.org/
[Приступљено 15.9.2024.]

[5] React – Official website,
урл: https://react.dev/
[Приступљено 15.9.2024.]

[6] Vizzu – 40 types of Data Visualization Charts and Graphs,
урл: https://www.vizzu.io/blog/data-visualization-types
[Приступљено 15.9.2024.]

[7] Cytoscape.js – Official website,
урл: https://js.cytoscape.org/
[Приступљено 15.9.2024.]

[8] Sigma.js – Official website,
урл: https://www.sigmajs.org/
[Приступљено 15.9.2024.]

[9] Chart.js – Official website,
урл: https://www.chartjs.org/docs/latest/
[Приступљено 15.9.2024.]

[10] Recharts – Official website,
урл: https://recharts.org
[Приступљено 15.9.2024.]

[11] Victory – Official website,
урл: https://commerce.nearform.com/open-source/victory/docs/victory-chart/
[Приступљено 15.9.2024.]

Биографија:

Дане Милишић рођен је 9. априла 2000. године у Новом Саду. Завршио је основну школу „Доситеј Обрадовић“ у Новом Саду, а затим друштвено-језички смер гимназије „Исидора Секулић“ у Новом Саду. Школске 2019/2020. године уписује основне академске студије Информационог инжењеринга на Факултету техничких наука у Новом Саду. Положио је све испите прописане планом и програмом, а потом и дипломирао у септембру 2023. године, са просечном оценом студија 8.88. Школске 2023/2024. године уписује мастер академске студије Примењених рачунарских наука и информатике на Факултету техничких наука у Новом Саду. Положио је све испите прописане планом и програмом на поменутим студијама у склопу модула Мултимедија и рачунарске игре.

контакт: milisic@uns.ac.rs

Зборник радова Факултета техничких наука, Нови Сад

UDK: 4.4
DOI: https://doi.org/10.24867/30BE27Simic

РАЗВОЈ АРХИТЕКТУРЕ БЕЗ СЕРВЕРА УПОТРЕБОМ ПЛАТФОРМЕ АМАЗОН ВЕБ СЕРВИСА

DEVELOPMENT OF A SERVERLESS ARHITECTURE USING THE AMAZON WEB SERVICES PLATFORM

Јован Симић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Рад се бави развојем апликације за резервисање смештаја употребом serverless модела тј. архитектуре без сервера. Поред три главна модела услуга (IaaS, PaaS, SaaS) обрађени су и све популарнији концепти serverless и FaaS услуга. Задатак рада је да представи једно од могућих архитектурних решења користећи предности услуга које нам облак пружа и тиме смањи недостатке традиционалних система. Описан је начин имплементације и како се ресурси на Амазоновом облаку (Amazon Web Services) креирају уз помоћ IaC (Infrastructure as Code) концепта.*

Кључне речи: *serverless, AWS, FaaS, IaC, CDK*

Abstract – *This thesis addresses the development of an accommodation reservation application using the serverless model. In addition to the three main service models (IaaS, PaaS, SaaS), the increasingly popular concepts of serverless and FaaS (Function as a Service) services are also covered. The task of the paper is to present one of the possible architectural solutions by utilizing the advantages of cloud services, thereby reducing the drawbacks of traditional systems. The method of implementation is described, as well as how resources on Amazon's cloud (Amazon Web Services) are created using the IaC (Infrastructure as Code) concept.*

Keywords*: serverless, AWS, FaaS, IaC, CDK*

1. УВОД

Традиционални приступ развијања апликација подразумева управљање и контролу рачунарским ресурсима у оквиру физичких центара организације. Сва опрема је одговорност организације, потребно ју је обезбедити унапред, конфигурисати, водити рачуна о ажурирању компоненти што изискује додатни напор и време [1]. Такође је неопходно размотрити и редудантност у случају покретања критичних апликација, осигуравајући бизнис функционалности увек доступним, уз одређене стратегије обнове после катастрофе (енгл. *Disaster Recovery*).

Све претходно наведено је створило добар темељ за развој рачунарства у облаку (енгл. *Cloud Computing*), који је донео револуцију омогућавајући компанијама да складиште податке и користе ресурсе (сервери, мрежа, софтверске апликације) путем удаљених центара који се налазе изван њихове инфраструктуре. Оно што нам доноси јесу већа прилагодљивост, скалабилност и смањење трошкова јер су ресурси постали доступни као услуга - *IaaS, PaaS, SaaS*, а не као физички хардвер. Додавање тј. уклањање ресурса и неопходних сервиса на захтев обезбеђује већу флексибилност у постављању жељене инфраструктуре или мењању постојеће, скалирајући потребе бизниса уз минималне напоре [2].

Овај рад се бави демонстрацијом имплементације serverless решења употребом платформе Амазон Веб Сервиса на примеру резервација смештаја.

2. ТЕОРИЈСКИ КОНЦЕПТИ

2.1. Историја рачунарства у облаку

Развој рачунарства у облаку започео је у шездесетим годинама прошлог века, када су усвојени основни концепти који дефинишу облак**.** Значајни напредак у имплементацији облака десио се 1970. године када је компанија IBM представила своју прву виртуелну машину - VM/370. У 1980. годинама, са брзим развојем рачунарских технологија, индустрије и компаније су тражиле начине да повежу своје рачунаре и омогуће приступ дељеним подацима. Појава виртуелних приватних мрежа (енгл. *Virtual Private Network – VPN*) омогућила је ову комуникацију и сарађивање. Појава Amazon Web Services (AWS) на почетку 21. века представља важан моменат у развоју рачунарства у облаку, отварајући пут за његову широку примену.

2.2. Инфраструктура као сервис

Инфраструктура као сервис (енгл. *Infrastructure as a Service - IaaS*) представља сервисе на мрежи који обезбеђују високо апстрактне програме, ослоњене на инфраструктуру која се налази испод површине. Овај модел апстрахује детаље инфраструктуре, укључујући физичке ресурсе, партиционисање података, скалирање и прављење резервних копија (енгл. *backup*). Корисници омогућеним приступом на захтев добијају инфраструктуру коју су закупили код провајдера, укључујући сервере, меморију за складиштење и мрежне ресурсе. Инфраструктура као сервис омогућава већу флексибилност и скалабилност у поређењу са традиционалним приступом [3].

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Жељко Вуковић, доцент.

2.3. Платформа као сервис

Платформа као сервис (енгл. *Platform as a Service* – PaaS) омогућава корисницима да развијају и креирају апликације у облаку користећи програмске језике, библиотеке и софтверске алате које пружа провајдер [4]. У овом моделу, корисници не управљају инфраструктуром, већ имају контролу над развијеним апликацијама и могућности за подешавање окружења.

2.4. Софтвер као сервис

На највишем нивоу апстракције, модел софтвер као сервис (енгл. *Software as a Service* – SaaS) омогућава корисницима да интерагују са већ готовим апликацијама које нуди провајдер, обично уз месечну надокнаду. Вендор управља апликацијом и свим аспектима испод ње, а оне су доступне као десктоп верзије, преко веб претраживача или мобилног телефона.

2.5. Serverless рачунарство и архитектуре

Serverless рачунарство представља нови модел у облаку, у коме провајдери динамички алоцирају ресурсе на захтев корисника, преузимајући одговорност за одржавање сервера. Иако сама реч "serverless" може деловати конфузно и сугерише да сервери не постоје, у стварности они и даље постоје, али су корисницима невидљиви и не захтевају управљање, конфигурисање или скалирање.

Овај модел наплате заснива се на коришћењу ресурса (енгл. *pay-as-you-go*), те корисници плаћају само онолико колико користе, без фиксних или унапред дефинисаних трошкова. Овакво апстраховање инфраструктуре омогућава развојним тимовима да се фокусирају на код и логику апликације [5].

2.6. Функција као сервис

Функција као сервис представља подскуп serverless модела. Ова парадигма се најчешће примењује у рачунарству вођеног догађајима, где се функција покреће када се деси одређени догађај, као што је пристигла порука у реду или HTTP захтев. Када се функција развије, њено скалирање на горе и доле се врши аутоматски.

Током периода када нема захтева, сервер се гаси и ослобађа ресурсе за друге функције. Као и код других serverless имплементација, провајдер не наплаћује услуге када функција није активна [6].

2.7. Амазон веб сервиси

Амазон веб сервиси (енгл. *Amazon Web Services – AWS*) представљају једну од најпопуларнијих платформи за рачунарство у облаку, развијену под брендом компаније Амазон, која обухвата разноврсне услуге које подржавају претходно поменуте моделе облака – IaaS, PaaS, SaaS.

Ове услуге су представљене кроз различите сервисе за складиштење и обраду података, изнајмљивање рачунарских ресурса, управљање мрежним компонентама, безбедност, аналитику, вештачку интелигенцију, управљање великим обимима података (Big Data) и друго [7].

3. ПРЕГЛЕД НАЈЗНАЧАЈНИЈИХ АМАЗОН ВЕБ СЕРВИСА

3.1. Amazon Simple Storage Service (S3)

S3 представља један од првих сервиса који су били доступни јавности. Као што сам назив сервиса каже, главна сврха је складиштење објеката у такозване бакете (енгл. *bucket*), који морају имати глобално јединствен назив, независно од налога на ком је креиран. Различити су случајеви коришћења S3 сервиса – на пример, за језера података (енгл. *data lakes*), архивирање, чување резервних копија, објављивање садржаја статичких веб сајтова, опоравак од катастрофе,…

3.2. AWS Lambda

Једна од првих асоцијација када се говори о serverless решењима на AWS платформи, реч је о познатом Lambda сервису. Функције које се окидају по потреби, без обавезе управљања серверима или провизионисања истих. Скалирају се аутоматски спрам пристижућих захтева, али их карактерише кратко извршавање, ограничено временом.

3.3. Amazon DynamoDB

DynamoDB представља serverless NoSQL базу података, без потребе за одржавањем, високо доступна са копирањем у вишеструким зонама доступности. Дизајниран је да подржи огромна оптерећења скалирајући се аутоматски колико је потребно. Подржава милионе захтева, трилионе редова и стотине терабајта складишта. Нема потребе за креирањем базе података, она је већ ту и спремна за употребу, све што је потребно је креирати тебелу, што је и карактеристично за DynamoDB. Свака табела има примарни кључ који треба да се одреди у време креирања табеле. Подаци се чувају у табели као ставке (енгл. *items*).

3.4. Simple Queue Service (SQS)

SQS је један од најстаријих Амазон веб сервиса, чије су услуге постале јавно доступне и на располагању корисницима још од 2004. године [8]. Користи се да би се апликације раздвојиле и нема ограниченост кад је у питању скалабилност. Испорука порука је гарантована бар једанпут (енгл. *at least once delivery*), али могу се појавити дупликати. Такође није гарантовано да ће поруке бити сортиране по редоследу (енгл. *best effort ordering*).

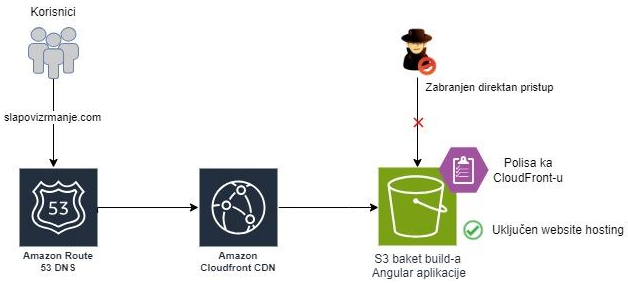
3.5. Route 53

Route 53 представља високо доступан и скалабилан сервис који се бави именима домена (енгл. *Domain Name System – DNS*). Преусмерава крајње кориснике до циљане дестинације тј. жељеног ресурса. Да би ово било могуће, људски читљиве називе као на пример *www.example.com* је потребно превести у адресу Интернет протокола (енгл. *Internel Protocol Address – IP address*) [9]. Ово није једина карактеристика Route 53-а. Могуће је регистровати називе домена за вашу веб апликацију, као и проверити доступност ваших ресурса.

4. НАМЕНА СИСТЕМА И АРХИТЕКТУРА РЕШЕЊА

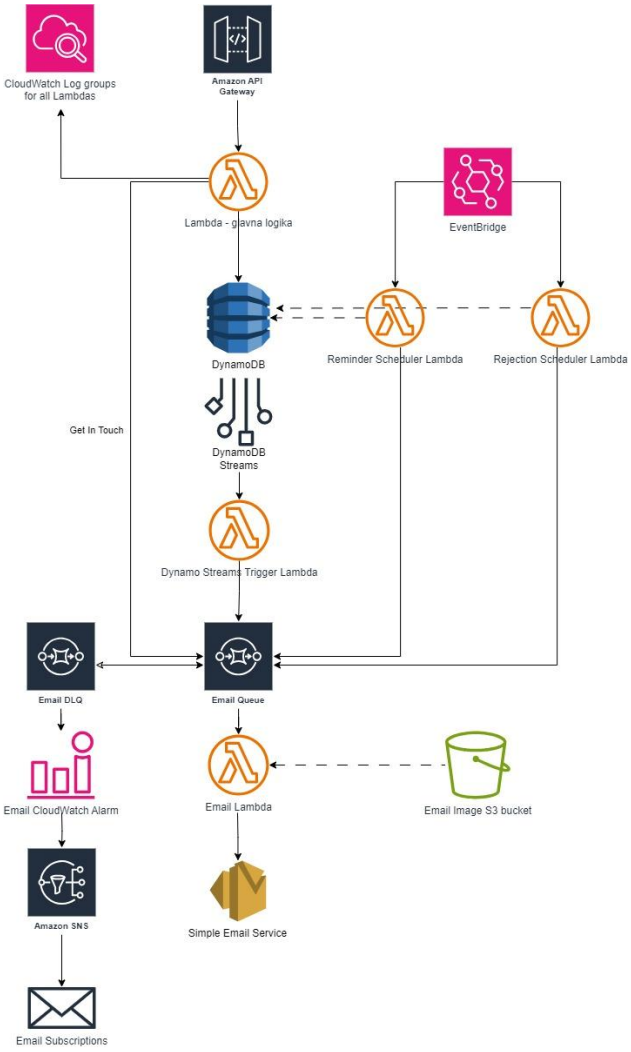
Намена постојања овог система јесте пре свега да олакша заказивање термина смештаја у ресторану “Слапови Зрмање”. До сада није постојало неко аутоматизовано решење. Ни ово не представља потпуно аутоматизовано решење, због захтева и ограничења клијента, али му олакшава интеракцију са корисницима.

На слици 1. налази се дијаграм који описује клијентски део апликације, односно frontend. Апликација је имплементирана коришћењем Angular библиотеке. Код је потребно изградити и затим га поставити на S3 бакет. Потребно је активирати S3 website hosting и изменити полису бакета да дозволи приступ CloudFront-у. Креира се дистрибуција у CloudFront-у где се као извор тј. порекло подеси претходно креирани бакет. Апликација постаје јавно доступна путем изгенерисаног URL-а. За регистрацију домена би се користио Route 53, као и за услуге преусмеравања Интернет саобраћаја.



Слика 1. Frontend део система

Backend део система је приказан дијаграмом архитектуре решења видљивим на слици 2. Улазну тачку у овај део система представља API Gateway, који даље делегира позиве инициране од стране frontend-а ка Lambda функцији. За базу података је коришћен DynamoDB сервис, још један serverless сервис. Да би се омогућио приступ бази, функцији је потребно доделити неопходна права. Уз сјајну комбинацију са DynamoDB Streams, приликом уписивања нових захтева у базу, у приближно реалном времену су догађаји пропагирани активирањем следеће Lambda функције. Ова функција обрађује пристигле догађаје, формирајући SQS поруке које садрже неопходне информације за даљу обраду. За формирање е-поште је задужена посебна Lambda функција која чита поруке са реда, и уз помоћ прослеђених података као и метаподатака формира одговарајући тип и садржај поруке. Порука се потом шаље употребом Simple Email Service-а до жељене дестинације. Уколико дође до грешке приликом слања е-поште услед слабе Интернет конекције, или из неког другог разлога, биће покушано поново захваљујући retry карактеристици SQS-а. Приликом успешности слања, порука се брише са реда. Насупрот, поновним неуспехом из непознатих разлога, порука завршава у специјалном Dead Letter Queue-u. Услед нагомилавања порука из озбиљнијих разлога жељних пажње, када број порука пређе задати праг, активира се аларм који шаље обавештење на Simple Notification Service топик. Потом се претплатницима овог топика, који представља тим подршке, шаље порука путем е-поште.



Слика 2. Дијаграм архитектуре решења backend дела система.

Функционалности које се извршавају једанпут дневно се активирају заказаним радњама Event Bridge-а. Ови догађаји побуђују две посебне Lambda функције. Надлежности ових функција су подсећање клијента на предстојеће заказане термине, као и понуде алтернативних термина корисницима чији су захтеви одбијени. Оне читају филтриране податке из DynamoDB табеле, креирају SQS поруке и даље их шаљу на SQS ред.

5. ИМПЛЕМЕНТАЦИЈА РЕШЕЊА

Клијентска страна је реализована уз помоћ популарног Angular радног оквира. Заснива се на TypeScript језику, бесплатан је и представља single-page апликацију која трчи на Node.js. Апликација је доступна на Интернету захваљујући интеграцији два Амазонова сервиса – Amazon S3 и Amazon CloudFront. Комбинација ових сервиса је изузетно честа и може се рећи да представља стандард за испоруку frontend апликација и статичког садржаја на скалабилан и перформантан начин. У претходном поглављу је

графички приказана и описана међусобна повезаност компоненти и на који то начин се један захтев обрађује од самог почетка проласком кроз API Gateway, па све до чувања ентитета у DynamoDB табели и слања електронске поште. Главна логика ипак је смештена у оквиру Lambda функција уз помоћ Java програмског језика. Шаблон једне Lambda функције доступан је у листингу 1.

```
@Slf4j
@Component
@AllArgsConstructor
public class DynamoStreamTriggerComponent {

  private final AwsService awsService;
  private final ObjectMapper objectMapper;
  private final AccommodationMapper
accommodationMapper;

  public Function<DynamodbEvent,
DynamodbEvent> handleDynamoStreamEvent() {
    return dynamodbEvent -> {
       // The remaining code goes here
      return dynamodbEvent;
    };
  }
}
```

Листинг 1. Метода која се извршава при активацији Lambda функције.

AWS Cloud Development Kit - CDK представља радни оквир за дефинисање инфраструктурних ресурса користећи познате програмске језике, од који су тренутно подржани: TypeScript, JavaScript, Python, Java, C# и Go [10]. Постављање AWS инфраструктуре је подржано уз помоћ овог алата.

5. ЗАКЉУЧАК

Рад се бавио применом концепата рачунарства у облаку и serverless технологије као део њега**.** Реализација овог решења је постигнута интеграцијом различитих сервиса које пружа Amazon Web Services платформа. Главне предности система јесу ефикасност трошкова и оптимизација ресурса. Преласком на модел плаћања на основу потрошње може се лако оптимизовати алокација ресурса и смањити оперативни трошкови. Ова финансијска флексибилност подстиче инжењере да експериментишу без терета великих почетних инвестиција.

Са друге стране, све већа зависност од облака доноси и нове изазове у погледу безбедности и приватности. Поред тога, регулаторни оквири и законодавство ће се морати прилагођавати како би пратили брзи развој технологије и осигурали адекватну заштиту корисника. Пројекат описан у раду је само један од примера трансформације традиционалних архитектурних решења под утицајем рачунарства у облаку.

ЛИТЕРАТУРА

[1] „Traditional IT Deployment Models,“ [На мрежи]. Available: https://www.flackbox.com/traditional-it-deployment-models-on-prem-colo. [Последњи приступ May 2024].

[2] „What is cloud computing?,“ [На мрежи]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing. [Последњи приступ May 2024].

[3] „What are Iaas, Paas and Saas?,“ [На мрежи]. Available: https://www.ibm.com/topics/iaas-paas-saas. [Последњи приступ May 2024].

[4] P. Mell и T. Grance, „The NIST Definition of Cloud Computing,“ National Institute of Standards and Technology: U.S. Department of Commerce, 2011.

[5] R. Miller, „AWS Lambda Makes Serverless Applications A Reality,“ *TechCrunch*, 2015.

[6] B. King, „What is FaaS? Function as a Service explained,“ 1 February 2022. [На мрежи]. Available: https://www.digitalocean.com/blog/what-is-faas-function-as-a-service-explained. [Последњи приступ May 2024].

[7] N. Barney, „Amazon Web Services,“ [На мрежи]. Available: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services. [Последњи приступ May 2024].

[8] „A History of Amazon Web Services (AWS),“ [На мрежи]. Available: https://www.awsgeek.com/AWS-History/. [Последњи приступ May 2024].

[9] „What is an IP Address?,“ 5 September 2023. [На мрежи]. Available: https://www.geeksforgeeks.org/what-is-an-ip-address/. [Последњи приступ May 2024].

[10] „What is the AWS CDK?,“ [На мрежи]. Available: https://docs.aws.amazon.com/cdk/v2/guide/home.html. [Последњи приступ May 2024].

Кратка биографија:

Јован Симић рођен је 30. јануара 1999. године у Новом Саду. Факултет техничких наука у Новом Саду, студијски програм Софтверско инжењерство и информационе технологије уписао је 2018. године. Након завршених основних студија, 2022. године, уписао је мастер академске студије из исте области на смеру Електронско пословање.

контакт: jovansimic995@gmail.com