Zbornik radova Fakulteta tehničkih nauka, Novi Sad

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

FAGL – JEZIK SPECIFIČAN ZA DOMEN IMPLEMENTACIJE VEB APLIKACIJA

FAGL – A DOMAIN-SPECIFIC LANGUAGE FOR WEB APPLICATION IMPLEMENTATION

Lazar Pavlović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

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

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

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

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

1. UVOD

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

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

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

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

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

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

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

2. ARHITEKTURA I DIZAJN REŠENJA

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

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

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

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

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

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

3. KONKRETNA SINTAKSA FAGL JEZIKA

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

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

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

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

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

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

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

4. STUDIJA SLUČAJA

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

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

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

5. POREĐENJE SA DRUGIM REŠENJIMA

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

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

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

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

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

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

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

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

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

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

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

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

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

6. ZAKLJUČAK

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

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

7. LITERATURA

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

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

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

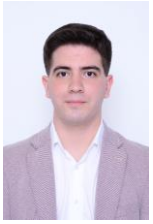
[4] A. Suljkanović, B. Milosavljević, V. Inđić, and I. Dejanović, "Developing microservice-based applications using the silvera domain-specific language," *Applied Sciences*, vol. 12, no. 13, p. 6679, 2022.

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

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

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

Kratka biografija:



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