Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/27BE26Makan

PLATFORMA ZA KONTROLU VERZIJA I SARADNJU ZASNOVANA NA AWS USLUGAMA

VERSION CONTROL AND COLLABORATION PLATFORM BASED ON AWS SERVICES

Albert Makan, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu opisana je implementacija i specifikacija arhitekture platforme za kontrolu verzija i saradnju koja je postavljena na AWS.*

Ključne reči: *Kontrola verzija, Node.js, Next.js, Docker, AWS, Git server, MongoDB*

Abstract – *This paper describes the implementation and specification of the architecture of a version control and collaboration platform that is deployed on AWS.*

Keywords: *Version control, Node.js, Next.js, Docker, AWS, Git server, MongoDB*

1. UVOD

U eri brzog razvoja softverskih rešenja, efikasno upravljanje verzijama softvera postaje kritičan deo procesa razvoja. Ovaj rad detaljno istražuje implementaciju sistema za kontrolu verzija i saradnju, koji je zasnovan na *AWS* platformi. Rad ima za cilj da prikaže praktičnu primenu specifičnih tehnologija i arhitekture, i kako se ovi alati koriste da bi se postigli konkretni funkcionalni ciljevi sistema.

Primarni korisnici sistema su programeri, ali i drugi koji učestvuju u isporuci softverskog rešenja. Registracija na sistem i prijava su preduslov korisniku za korišćenje svih funkcionalnosti koje nudi aplikacija. Korisnici mogu da kreiraju, pregledaju i upravljaju repozitorijumima za svoje projekte. Repozitorijum može da bude javni ili privatni.

Platforma omogućuje korisnicima da prate promene u kodu, kreiraju grane za razvoj, izvrše spajanje (*merge*) promena iz različitih grana i imaju uvid u prethodne verzije projekta. Korisnik može otvoriti zahtev spajanja (*pull request*) kako bi predstavili svoje promene, a ostali saradnici na repozitorijumu mogu pregledati, komentarisati i prihvatiti te promene.

Korisnici mogu da kreiraju, dodeljuju i prate zadatke (*issues*), greške i druge probleme povezane sa projektima i da postave ciljeve i rokove za svoje projekte. Svaki cilj se definiše kao "*milestone*" i može se koristiti za grupisanje zadataka.

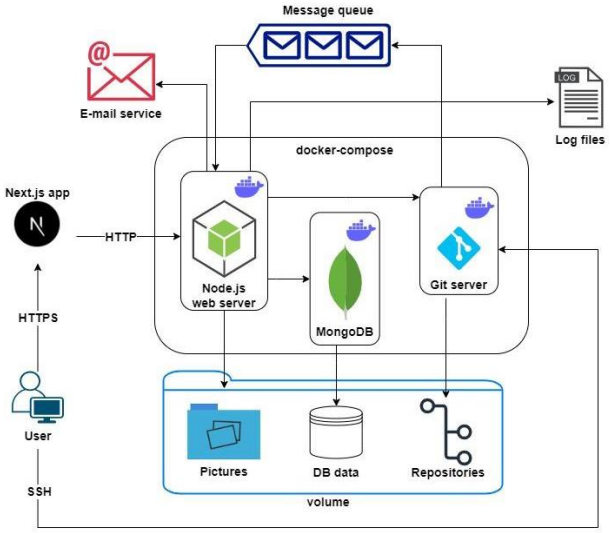
Korisnici imaju uvid u razne statistike i grafikone u vezi sa repozitorijumom, kao što su broj predatih izmena koda (engl. *commit*), broj dodatih i obrisanih linija koda tokom vremena, i po saradniku.

Korisnici mogu označiti zvezdom (engl. *star*) repozitorijume koji su im interesantni ili korisni i mogu kopirati (engl. *fork*) repozitorijume drugih korisnika.

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

2. SPECIFIKACIJA SISTEMA

Šema arhitekture sistema je prikazana na slici 1. Serverska i klijentska strana su razvijeni kao odvojene aplikacije. Za funkcionisanje sistema neophodno je da serverska aplikacija komunicira sa bazom podataka i sa git serverom. Git server šalje podatke o novim komitovima preko reda poruke. Korisnik pristupa aplikaciji kroz klijentsku stranu i komunicira sa git serverom putem *SSH* protokola.



Slika 1. *Šema arhitekture sistema*

Osnovni entiteti sistema su *User* (korisnik), *Repository* (repozitorijun), *Issue* (zadatak), *PullRequest* (zahtev za povlačenje), *Milestone* (etapa), *Label* (labela). *Star* (zvezda) i *Collaborator* (saradnik) su entiteti veze između repozitorijuma i korisnika.

Događaji u sistemu su modelovani po principima *Event Sourcing*-a koji spada u *Domain Driven Design* šablone. Svaka akcija korisnika se modeluje kao događaj. Događaji su nepromenjivi i predstavljaju jedini izvor istine. *Issue*-i i *Pull Request*-ovi su koreni agregata (*Aggregate Roots*), promena njihovog stanja je podstaknuta događajem. Sami događaji ne određuju kako će biti obrađeni; ta odgovornost leži kod agregata kojima

pripadaju. Još jedna bitna karakteristika agregata je sposobnost rekonstrukcije trenutnog stanja na osnovu događaja. Svaki događaj nasleđuje *BaseEvent* klasu.

3. IMPLEMENTACIJA

U ovom delu je predstavljena implementacija svakog dela aplikacije, odnosno opisani su potrebni koraci za integraciju *AWS* servisa. Pri implementaciji aplikacija je razdvojena na tri dela: na serversku stranu, git server, i klijentsku stranu.

3.1. Serverska strana

Serverska strana aplikacije je realizovana preko *TypeScript* programskog jezika i *Node.js* okruženja. Za skladištenje podataka korišćena je *Mongo* baza podataka.

Typescript [1] je programski jezik koji proširuje *JavaScript* dodajući mu statičku tipizaciju, omogućavajući programerima da otkriju i isprave greške već tokom faze razvoja. To čini serversku stranu aplikacije sigurnijom i lakšom za održivanje. *Node.js* [2] je izvršno okruženje koje omogućava izradu serverskih aplikacija u *JavaScript*-u. Njegova asinhrona arhitektura omogućava efikasno upravljanje istovremenim zahtevima, poboljšavajući performanse serverske strane.

MongoDB [3] je baza podataka orijentisana prema dokumentima koja koristi *BSON* (*Binary JSON*) dokumente za skladištenje podataka. Fleksibilnost ovog *NoSQL* pristupa olakšava rad sa promenljivim i kompleksnim podacima.

Kod aplikacije organizovan je po funkcionalnostima (*features*), gde svaka funkcionalnost sadrži svoje modele, repozitorijume, servise i rute. Ovakav pristup olakšava razdvajanje odgovornosti i čini kod manje izmešanim. U okviru ove organizacije, svaka funkcionalnost ima svoj sopstveni direktorijum u okviru aplikacije. Svaka funkcionalnost ima svoj model koji definiše strukturu podataka ili objekata povezanih sa tom funkcionalnošću. Repozitorijum služi za interakciju sa bazom podataka, korišćenjem "*mongodb*" biblioteke. Servis sadrži poslovnu logiku povezanu sa određenom funkcionalnošću. Svaki *feature* ima svoje definisane rute koje određuju kako će aplikacija odgovarati na različite *HTTP* zahteve.

U glavnom fajlu sve rute koje podržavaju različite funkcionalnosti se povezuju zajedno. Ovde su dodate odgovarajuće midlver funkcije, kao što su *authorize*, *findRepository* i *errorHandler*, i pokreće se *express* aplikacija.

3.2. Git server

Git server pruža centralizovano skladište za Git repozitorijume i omogućuje korisnicima da kloniraju, izmenjuju i sinhronizuju svoj kod sa drugima. Git server je organizovan kroz *Node.js* aplikaciju, *SSH* servera i *Linux* fajl sistema i smešten u *Docker* [4] kontejner.

Fajl sistem gde se čuvaju repozitorijumi organizovan je na sledeći način: u direktorijumu "*/home*", korenskom direktorijumu za sve korisnike, svaki korisnik ima svoj poddirektorijum sa svojim repozitorijumima. Fajl "*.ssh/authorized_keys*" sadrži javne SSH ključeve korisnika za autentifikaciju bez lozinke. Korisnički direktorijumi sadrže repozitorijume, svaki sa "*.git*" folderom, "*collaborators.txt*" fajlom sa spiskom saradnika, i "*public*" fajlom koji označava javni repozitorijum. Kreirani su *Linux* korisnici za svakog korisnika i grupe za svaki repozitorijum sa pravima pristupa. Vlasnik i grupa imaju pun pristup, dok ostali imaju pristup čitanja samo ako je repozitorijum javni.

Ključna komponenta git servera je *API* koji je implementiran kao *Node.js express* aplikacija. Ovaj *API* pruža krajnje tačke koje pozivaju odgovarajuće funkcije koje izvršavaju git i linux komande nad repozitorijumima. Aplikacija sadrži funkcije za kreiranje i brisanje repozitorijuma, za dobavljanje *commit*, *tree* i *blob* objekata, za rad sa granama i tagovima, za spajanje grana i dodavanje korisnika i kolaboratora. *API* nema mehanizam za autorizaciju korisnika, zato svaki zahtev ide kroz serversku stranu.

Pristup udaljenim Git repozitorijumima putem operacija poput *clone*, *pull* i *push* je moguć samo putem *SSH* protokola. Git koristi *SSH* za autentikaciju korisnika prilikom operacija prema udaljenom repozitorijumu, koristeći *SSH* ključeve. Javni ključ se čuva na serveru, a privatni na lokalnom računaru. Pokretanje *SSH* procesa na Git serveru se postiže korišćenjem *sshd* (*OpenSSH Daemon*). Za autentikaciju i pristup udaljenom Git repozitorijumu pomoću *SSH* ključeva, potrebno je generisati *SSH* ključ na lokalnom računaru, konfigurisati *SSH* agenta, dodati privatni ključ u agenta, i postaviti javni ključ na server putem klijentske aplikacije.

Git omogućava da se pozovu proizvoljno napisane skripte kada se dogode određene važne akcije. Postoje dve grupe ovih „hvataljki“ za događaje (engl. *hooks*) [5]: one na klijentskoj i one na serverskoj strani. Čuvaju se u "*.git/hooks*" direktorijumu. Git *post-receive* je skripta koji se izvršava nakon što se Git repozitorijum na udaljenom serveru uspešno ažurira. U ovoj aplikaciji *post-receive* se koristi za slanje podataka o novim komitovima na red poruka. Serverska strana će pročitati te podatke sa reda, proveriti da li se referencira neki *issue* u komit poruci, i dodati *IssueReferencedEvent* događaj ako jeste.

Git server, kao i serverska strana smešten je u *Docker* kontejner da bismo obezbedili izolaciju aplikacije od ostatka sistema. Ovo sprečava potencijalne konflikte i probleme sa resursima. Kontejneri se lako mogu kreirati i uništiti, što olakšava reprodukciju i skaliranje Git servera prema potrebama. Kontejneri omogućavaju pakovanje svih zavisnosti koje Git server zahteva, uključujući operativni sistem, Git softver, *OpenSSH*, i konfiguracije što pojednostavljuje upravljanje. Za postizanje ove svrhe, koristi se *Dockerfile* koji definiše kontejner oko Git servera.

3.3. Klijentska strana

Klijentsku stranu čini *Next.js* aplikacija. Za korisnički interfejs su korišćeni *Tailwind CSS*, *Headless UI* i *Heroicons*.

Next.js [6] radni okvir za razvoj web aplikacija zasnovanih na React biblioteci. *Next.js* dolazi sa jednostavnim i konvencijama vođenim sistemom rutiranja, što olakšava definisanje ruta i navigaciju između stranica. Omogućava brzo učitavanje promena u realnom vremenu tokom razvoja aplikacije bez potrebe za osvežavanjem stranice.

Tailwind CSS [7] je *CSS* radni okvir koji koristi jednostavne klase za stilizovanje *HTML*-a. Ova fleksibilnost i prilagodljivost omogućava brži i efikasniji razvoj klijentske strane.

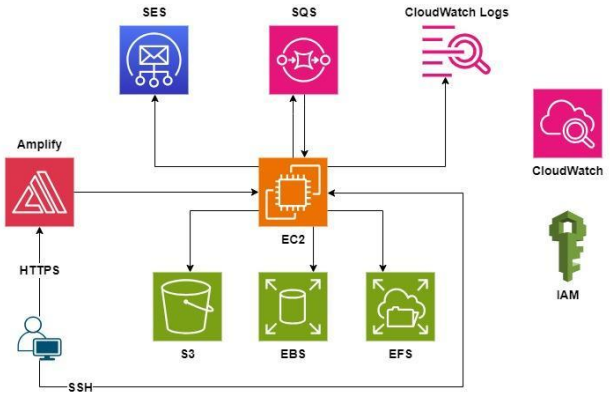
Za upravljanje podacima i stanjem aplikacije koristi se *React Query*, što omogućava jednostavno dohvatanje i keširanje podataka sa serverske strane, čineći aplikaciju bržom i efikasnijom. Takođe, *Context API* se koristi za globalno upravljanje stanjem i deljenje podataka između komponenti.

Kod aplikacije organizovan je u tri foldera: *core*, *features* i *pages*. Folder sa nazivom *core* sadrži osnovne komponente i funkcionalnosti koje su zajedničke za čitavu aplikaciju. U folderu sa nazivom *features* komponente su grupisane po funkcionalnostima. Svaki podfolder za određenu funkcionalnost sadrži komponente specifične za tu funkcionalnost (stranice, kartice, tabele, forme...), tipove podataka i funkcije koje komuniciraju sa serverskom stranom putem *axios* zahteva. Folder pod nazivom *pages* služi za čuvanje *Next.js* stranica. Svaka datoteka unutar ovog foldera predstavlja jednu rutu u aplikaciji.

3.3. AWS

Amazon Web Services (AWS) [8] je vodeći i široko korišćen *cloud computing* servis i platforma koju nudi kompanija *Amazon*. *AWS* pruža obimne i raznolike servise za računarstvo u oblaku, čime omogućava organizacijama i pojedincima da iznajme računarske resurse, skladišta podataka i različite servise putem Interneta.

Na slici 2 je prikazana struktura integrisanih servisa *AWS* platforme pri realizaciji aplikacije.



Slika 2. *Arhitektura integrisanih AWS servisa*

Serverska strana zajedno sa *Mongo* bazom podataka i Git serverom hostovana je na *Amazon Elastic Compute Cloud (EC2)* instanci, dok je klijenstka strana smeštena na *Amazon Amplify Hosting* platformi. Skladišta podataka *EC2* instance čini *Amazon Elastic Block Storage (EBS)* i *Amazon Elastic File System (EFS)*. Koristeći *AWS Software Development Kit (SDK)* verzije 3 za *JavaScript* [9], server ostrvaruje komunikaciju sa *Amazon Simple Storage Service (S3)* za skladištenje i dobavljanje slika korisnika, sa *Amazon Simple Email Service (SES)* za slanje imejl obaveštenja korisnicima, sa *Amazon Simple Queue Service (SQS)* za upravljanju redom poruka i sa *Amazon CloudWatch Logs* za praćenje i upravljanje logovima sistema. Aplikacija upravlja pravima pristupa koristeći *Amazon Identity and Access Management (IAM)* korisnika, što obezbeđuje strogu kontrolu nad pristupom *AWS* resursima. *Amazon CloudWatch* nadgleda i beleži performanse, aktivnosti i razne metrike svakog od *AWS* servisa u sistemu.

EC2 instanca kreirana je na *Ubuntu t3.micro* tipu sa *VPC* okruženjem, sigurnosnom grupom, *EBS* diskom i *EFS* fajl sistemom. Nakon pokretanja, pristup instanci se ostvaruje putem *SSH* klijenta. *AWS CodeDeploy* agent i *Docker* i *Docker Compose* se instaliraju, pripremajući instancu za automatsko postavljanje aplikacije i kontejnerizaciju.

IAM uloga je povezana sa *EC2* instancom, omogućavajući serverskoj strani upotrebu *AWS* servisa *(S3, SES, SQS, CloudWatch Logs*) bez prosleđivanja pristupnih ključeva u samoj aplikaciji. Ova praksa povećava sigurnost i pojednostavljuje upravljanje permisijama.

S3 se koristi u sistemu u dva slučaja sa kreiranjem dva *bucket*-a. *Bucket* "*siithub-images*" čuva slike korisnika, dok *CodePipeline* servis koristi drugi bucket za skladištenje koda aplikacije. Serverska strana koristi "*@aws-sdk/client-s3*" biblioteku za komunikaciju sa *S3* servisom, a "*multer*" i "*multer-s3*" za upload slika. *Bucket*-i nisu javno dostupni preko *URL*-a.

SES se koristi za slanje pozivnica za saradnju na repozitorijumu putem imejla. Korisnici, nakon registracije na aplikaciju, dobijaju mejl pod nazivom "*Amazon Web Services – Email Address Verification Request*" sa linkom za verifikaciju svoje imejl adrese. Ovaj proces osigurava validnost imejl adrese, povećavajući bezbednost i pouzdanost komunikacije putem imejla. Pozivnice za saradnju šalju se koristeći prethodno kreirani šablon "*CollaboratorInvitation*". Serverska strana koristi "*@aws-sdk/client-ses*" biblioteku za komunikaciju sa *SES* servisom.

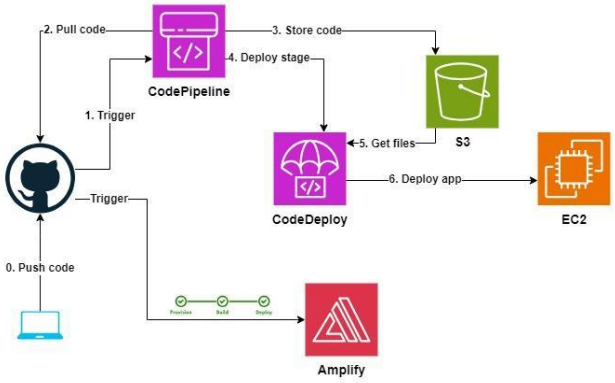
SQS se koristi za efikasnu obradu poruka o novim komitovima u sistemu. Nakon izvršavanja *post-receive* „hvatača događaja“ na Git serveru, skripta aktivira slanje poruka o novim komitovima na *SQS* red. Serverska aplikacija je konfigurisana da konzumira, čita, obrađuje i briše poruke sa *SQS* reda. Git server i serverska strana komuniciraju sa *SQS* servisom koristeći "*@aws-sdk/client-sqs*" biblioteku, dok se za konzumiranje poruka koristi "*sqs-consumer*" biblioteka. *Consumer* instanca je pokrenuta prilikom startovanja *express* aplikacije.

CloudWatch Logs koristi se za efikasno praćenje i čuvanje aplikacionih logova u sistemu sa dve kreirane log grupe. Grupa "*siithub-logs*" čuva logove serverske strane aplikacije, dok druga log grupa dolazi od *Amplify* servisa. Logiranje na serverskoj strani koristi "*winston*" i "*winston-cloudwatch*" biblioteke za kreiranje i slanje logova u *CloudWatch Logs*. Svaki dan se kreira novi log tok (engl. *stream*), olakšavajući pristup i pregled logova po datumu. Strukturirane log poruke omogućavaju jednostavno pretraživanje i analizu informacija.

4. CI/CD

U ovom delu opisan je način kako se *GitHub*, centralno mesto za upravljanje izvornim kodom, povezuje sa *AWS* servisima kako bi se omogućila glatka interakcija i

isporuka promena. Na slici 3 prikazana je šema interakcije između *GitHub* repozitorijuma, *AWS CodePipeline*, *CodeDeploy*, *Amazon S3* skladišta, *Amazon EC2* servera i *AWS Amplify Hosting* platforme. Cilj je jasan: kada se desi promena na glavnoj (*master*) grani repozitorijuma, automatizovani *CI/CD* proces će osigurati da se kod automatski isporuči na *EC2* i *Amplify Hosting* platformu.



Slika 3. *Interakcija između GitHub-a i AWS servisa*

Za postavljanje i pokretanje serverske strane aplikacije na *EC2* instanci, koristi se *Docker Compose*, *appspec.yml* i odgovarajuće skripte. U *docker-compose.yml* su definisana tri servisa (*siithub-git-server*, *mongo*, *siithub-backend*) i jedna mreža (*application*) za povezivanje. Volumeni koriste povezani *Elastic File System (EFS)* za deljenje podataka između kontejnera i instanci. *appspec.yml* [10] definiše korake postavljanja i pokretanja aplikacije na *EC2* instanci. Različite faze (*hooks*) kao što su *ApplicationStop*, *BeforeInstall*, *AfterInstall*, *ApplicationStart* imaju specifične skripte ili komande, u ovom slučaju *ApplicationStop* izvršava "*docker-compose down*", *BeforeInstall* osigurava da postoji folder za instalaciju, *AfterInstall* izvršava "*docker-compose build*" i "*docker image prune -f*", a *ApplicationStart* izvršava "*docker-compose up -d*". Ovaj *appspec.yml* fajl treba smestiti u korenski direktorijum repozitorijuma.

CodeDeploy konfiguracija uključuje kreiranje aplikacije i *deployment* grupe, dok *CodePipeline* konfiguracija podrazumeva kreiranje jednog *pipeline*-a. U *CodePipeline*, odabiremo *GitHub* kao izvor koda, povezujemo se sa *GitHub* nalogom, biramo repozitorijum i granu. Faza izgradnje se preskače, direktno isporučujući izvorni kod na *AWS* infrastrukturu. Zatim konfigurišemo fazu isporuke, odabirući *CodeDeploy* aplikaciju i *deployment* grupu za isporuku.

Za postavljanje klijentskog dela aplikacije na *AWS Amplify*, prvo je potrebno konfigurisati *amplify.yml* fajl kako bi se definisali koraci za izgradnju i isporuku. Ovaj fajl je potrebno smestiti u korenski direktorijum repozitorijuma. Nakon toga, potrebno je integrisati *Amplify* sa *GitHub* repozitorijumom kako bi automatski pratili promene u kodu. *Amplify* ima sposobnost da detektuje da je aplikacija izgrađena pomoću *Next.js* okvira, što omogućava pravilnu konfiguraciju tokom postupka isporuke.

5. ZAKLJUČAK

Ovaj rad predstavio je implementaciju sistema za kontrolu verzija i saradnju zasnovanog na *AWS* platformi. Kroz detaljan pregled serverske i klijentske strane i git servera, kao i procesa kontinuirane integracije i isporuke, sistem je predstavljen kao moćan alat za efikasno upravljanje verzijama i olakšavanje saradnje unutar timova.

Dalji razvoj ovog sistema bi trebao uključiti izdvajanje autentikacije u *AWS Cogito* za bolju upravljivost identiteta korisnika. Pored toga, unapređenje infrastrukture kroz korišćenje alata poput *AWS CDK* i *CloudFormation*-a bi značajno unapredilo upravljanje resursima i automatsko održavanje infrastrukture. Osim toga, prebacivanje sa *MongoDB* unutar kontejnera na potpuno upravljano rešenje na *AWS*-u obezbediće unapređene performanse, skalabilnost i upravljanje podacima.

U zaključku, ova platforma za kontrolu verzija i saradnju zasnovana na *AWS*-u je postavljena kao efikasan alat za upravljanje projektima i saradnju. Mogući dalji razvoj sistema, posebno u vezi sa skalabilnošću i infrastrukturom, može znatno unaprediti njegove performanse i upotrebnu vrednost, čineći ga još efikasnijim i konkurentnijim rešenjem.

6. LITERATURA

[1] TypeScript - https://www.typescriptlang.org/
[2] Node.js - https://nodejs.org/en/about
[3] MongoDB - https://www.mongodb.com/
[4] Docker - https://www.docker.com/resources/what-container/
[5] Git Hooks - https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks
[6] Next.js - https://nextjs.org/
[7] Tailwind CSS - https://tailwindcss.com/
[8] What is AWS - https://aws.amazon.com/what-is-aws
[9] AWS SDK for JavaScript v3 - https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/introduction/
[10] CodeDeploy AppSpec file reference - https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file.html

Kratka biografija:

Albert Makan rođen je u Zrenjaninu 1999. godine. Godine 2018 upisao je Fakultet tehničkih nauka u Novom Sadu, smer Softversko inženjerstvo i informacione tehnologije. 2022. godine diplomirao je na osnovnim studijama i upisao master studije na istom smeru.
kontakt: makanalbert@gmail.com