Utjecaj izmjena Zakona o obnovljivim izvorima energije i visokoučinkovitoj kogeneraciji na FNE

Kocijan, Filip

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:627510

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I

INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PRIMJENA BAZE PODATAKA U SUSTAVU ZA TESTIRANJE PROGRAMSKE PODRŠKE U AUTOMOBILSKOJ INDUSTRIJI

Diplomski rad

Luka Kopić

Osijek, 2022.

SADRŽAJ

1. UVOD 1
2. TEHNOLOGIJE KORIŠTENE ZA IZRADU BAZE PODATAKA I APLIKACIJE 3
2.1. Postojeće tvrtke i aplikacije za testiranje proizvoda 3
2.2. Pohrana podataka 4
2.3. Vremenski nizovi, odzivi i primjene 7
2.4. Ograničenja korištenih tehnologija 9
3. PRIMJENA BAZE PODATAKA U SUSTAVU ZA TESTIRANJE PROGRAMSKE PODRŠKE U AUTOMOBILSKOJ INDUSTRIJI 10
3.1. Zahtjevi baze podataka 10
3.2. Dizajn baze podataka 10
3.3. Implementacija baze podataka 11
3.4. Aplikacijsko programsko sučelje (API) 13
3.5. Implementacija API-ja 13
3.5.1. POST metoda 15
3.5.2. GET metoda 16
3.5.3. DELETE metoda 18
3.5.4. PUT metoda 19
4. ISPITIVANJE FUNKCIONALNOSTI API-JA I BAZE PODATAKA 21
4.1. Ručna testiranja 21
4.2. Automatska testiranja 22
5. ZAKLJUČAK 26
LITERATURA 27
SAŽETAK 28
ABSTRACT 29
ŽIVOTOPIS 30
PRILOZI 31

1. UVOD

Automobilska je industrija grana industrije koja trenutno jako brzo napreduje. Zahtijevaju se nove funkcionalnosti (npr. sustavi za pomoć vozaču u vožnji: automatsko kočenje, automatsko parkiranje, nadzor vozača) za što je potrebno dodatno sklopovlje (senzori, aktuatori, ugradbeni računalni sustavi) koji zahtijevaju specijalnu programsku podršku. Razvijene sustave, komponente i programsku podršku potrebno je temeljito ispitati i provjeriti jesu li u skladu s postavljenim zahtjevima. Iscrpnim testiranjima dobiva se programska podrška koja je pouzdanija, sigurnija i ima bolje performanse.

Testiranja u automobilskoj industriji na početku sadrže samo jednu komponentu na kojoj se vrše pojedinačni testovi (engl. *unit tests*). Nakon odrađenih pojedinačnih testova potrebno je testirati rad više komponenti u isto vrijeme kako bi se provjerila komunikacija i moguće greške između povezivanja istih. Na kraju izvršavaju se testiranja svih komponenata zajedno gdje se uspoređuju dobiveni i očekivani rezultati trenutne verzije cjelokupnog proizvoda. Rezultati testa mogu biti prolaz (engl. *passed*), preskočen (engl. *skipped*) i pad (engl. *failed*). Ako su očekivani i dobiveni rezultati jednaki rezultat je prolaz, ako su dobiveni rezultati različiti od očekivanog proizvodi pad, a rezultat je preskočen ako se test nije uopće izvršio. Testovi mogu biti izvršavani automatski ili ručno (engl. *manual*). Automatski testovi izvršavaju se pomoću prethodno napisanih skripti koje provjeravaju sve potrebne zahtjeve i šalju konačan rezultat kao povratnu informaciju. Ručne testove izvodi čovjek (tester) koji dobije popis potrebnih testova za izvršavanje i u ovisnosti o dobivenim i očekivanim rezultatima označava jedan od tri ponuđena rezultata.

Većina današnjih aplikacija podatke nastale tijekom korištenja pohranjuje u neku vrstu baze podataka. Kako u ostatku industrije tako i u testiranju automobilske programske podrške najčešće se koriste komercijalno dostupni sustavi za upravljanje bazama podataka. Dobivene podatke o testovima, korisnicima, testiranom hardveru ili softveru potrebno je pohraniti na računalo u strukturiranom obliku zbog lakšeg obrađivanja i prikazivanja podataka. Iz tog je razloga vrlo pogodno koristiti gotove baze podataka poput MySQL baze podataka. Uobičajeni način korištenja baze podataka u mrežnom načinu rada jest upotreba aplikacijskih programskih sučelja (engl. *application programming interface* - API).

Zbog sve veće cijene hardvera otežana je kupovina i opremanje prostorija za testiranje te dolazi do sve veće potrebe za udaljenim testiranjima. Udaljena testiranja omogućavaju korisniku spajanje na testnu stanicu koja se ne nalazi na istoj lokaciji kao i tester te povećava ukupno vrijeme

iskorištenosti testne stanice i smanjuje potrebno vrijeme za ponovnim postavljanjem testnog okruženja. U okviru ovog rada dizajnirana je i implementirana baza podataka s pripadajućim API-em za primjenu u provođenju udaljenog ispitivanja programske podrške. Razvijeno sučelje i baza dio su većeg sustava koji je integriran u web aplikaciju, a zadaća aplikacije je omogućiti testerima ispitivanje, unos i pregled rezultata testirane programske podrške u situacijama kada se ne nalaze na istoj fizičkoj lokaciji kao i sama oprema na kojoj se provode testovi.

Na početku rada opisane su tehnologije korištene za izradu baze podataka i aplikacije, postojeća rješenja problema i usporedbe između NoSQL i SQL baze podataka. U trećem poglavlju prikazani su dizajn i implementacija baze podataka i aplikacijsko programskog sučelja (API). U četvrtom je poglavlju izvršeno je ispitivanje i testiranje pojedinih dijelova i cjelokupnog sustava te na kraju je dan zaključak cijelog rada.

2. TEHNOLOGIJE KORIŠTENE ZA IZRADU BAZE PODATAKA I APLIKACIJE

U ovom poglavlju uspoređena su slična postojeća rješenja problema u dizajniranju, implementaciji i održavanju baze podataka i API-ja za testiranje programske podrške na udaljenoj lokaciji, razlika između SQL i NoSQL baze podataka, vremenske relacije i potencijalni problemi kod izrade baze podataka i web aplikacije. Zbog sve učestalijeg korištenja zaštite i privatnosti podataka postojeća javna rješenja u automobilskoj industriji teško je pronaći. Zbog toga su navedene i analizirane slične tvrtke i web aplikacije od kojih neke nisu u domeni automobilske industrije. U ovom radu za izradu baze podataka i web aplikacije za testiranje automobilske programske podrške na udaljenoj lokaciji korišteno je nekoliko različitih programskih jezika, postojećih biblioteka i programa. Baza podataka kreirana je pomoću MySQL Workbench 8.0 u MySQL [1] programskom jeziku. U bazu podataka unesene su potrebne tablice, relacije, procedure i okidači (engl. *triggers*). Cjelokupna web aplikacija kreirana je pomoću Python Flask [2], Python Django [3], HTML [4] (engl. *HyperText Markup Language*), JavaScript [5] i CSS [6] (engl. *Cascading Style Sheets*) programskih jezika. Za izradu korisničkog sučelja (engl. u*ser interface*) korišteni su HTML, JavaScript i CSS, dok su za pozadinsku komunikaciju između korisničkog sučelja i baze podataka (engl. *backend*) korišteni Python Django i Python Flask.

2.1. Postojeće tvrtke i aplikacije za testiranje proizvoda

1. Automotive Testing and Development Services, Inc - tvrtka koja je osnovana 1989. godine od strane Olsen Engineering. Izvode širok obujam testova u automotiv industriji među kojima je najpopularnije mjerenje količine ispušnih plinova [7].

2. SGS - tvrtka koja se bavi testiranjem cjelokupnog vozila, motora, komponenti, načinom sastavljanja dijelova, elektronike, goriva, ulja i raznih mehaničkih i kemijskih vrsta testiranja [8].

3. UserTesting - aplikacija za testiranje raznih proizvoda uključujući elektroniku, automobile, softver i hardver. UserTesting je korisnička testna platforma osnovana 2007. godine. Korisnik prolazi razne testove ovisno o demografskim standardima i životnim navikama. Ako se konačni rezultati poklapaju s traženim rezultatima, korisnik može postati službeni tester. Testiranja se mogu održavati bilo gdje u cijelom svijetu. Primjeri testiranih proizvoda: Ford automobili, CBS-ov portal dnevnih novosti, Adobe softver, Facebook funkcionalnosti [9].

4. Pinecone Research - aplikacija za testiranje glazbe, sporta, kućnih navika, filmova, televizijskih emisija, zdravstvenih i kozmetičkih proizvoda. Nakon prijave korisnik prima kratki upitnik koji omogućava tvrtki prikaz prioriteta i životnih navika korisnika. Korisnik dobiva bodove za svaki ispunjeni upitnik koji traje 15 do 20 minuta te bodove može zamijeniti za novčane kupone. Primjeri testiranih proizvoda: Amazon prime, TV emisije, Mars čokoladica, časopis New York [10].
5. Brooks - aplikacija za testiranje sportske obuće i opreme. Zbog popularnosti testa i web aplikacije program je često popunjen. Korisnici testiraju kvalitetu, udobnost, izdržljivost i estetski izgled opreme te istu dobiju na poklon kao način isplate. Dobivene rezultate unose na potrebna mjesta u web aplikaciji [11].

Usporede li se trenutna postojeća rješenja sa zahtjevima web aplikacije i baze podataka može se primijetiti kako prve dvije tvrtke (Automotive Testing and Development Services i SGS) imaju razvijena slična rješenja za probleme u automobilskoj industriji, dok su preostale tri aplikacije slične po načinima testiranja (testiranje izvršeno na udaljenoj lokaciji). UserTesting i Pinecone Research fokusiraju se na *unit* i integracijskim testovima dok se u slučaju Brooks može pronaći veliki broj *ad-hoc* testova. U *ad-hoc* testu korisnik testira opremu na proizvoljan način što može biti korisno u slučaju otkrivanja grešaka koje još nisu istražene.

2.2. Pohrana podataka

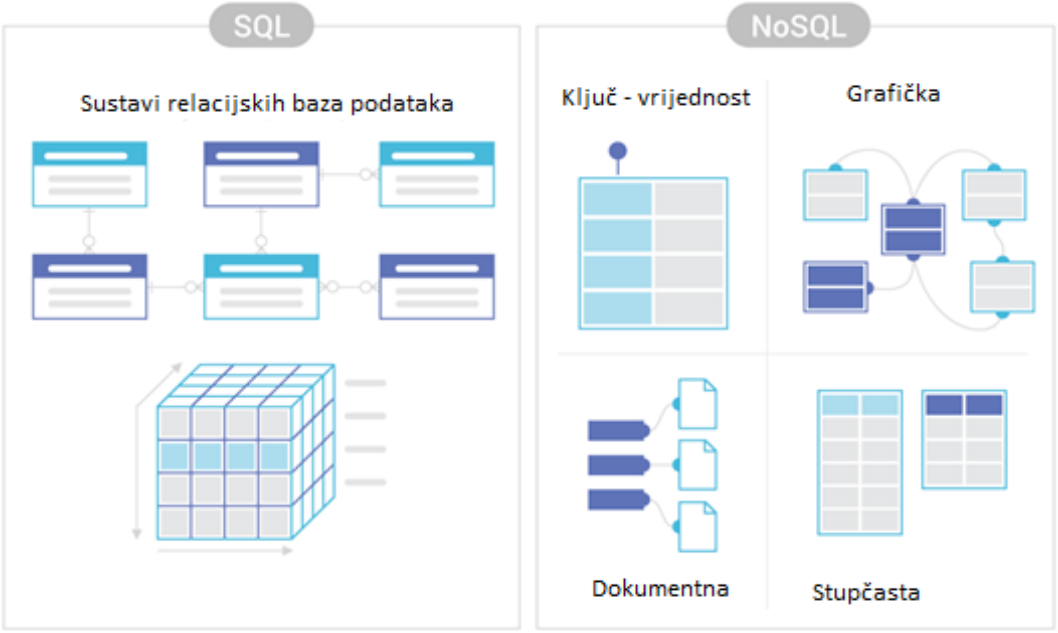
Trenutno postoje dva temeljna modela baze podataka (NoSQL i SQL). Za izradu ovog rada potrebno je odabrati koji će se model koristiti. Relacijske su baze podataka baze gdje su tablice i podaci u tablicama vezani relacijama između njih. Strukturirane su tako da se osigura ažurnost pohranjenih podataka, visoka sigurnost, nadzor pristupa podacima, mogućnost postavljanja više upita uz nekoliko kriterija u istom trenutku, najmanja redundancija i trajno očuvanje integriteta pohranjenih podataka. SQL je programski jezik koji se koristi za kreiranje i upravljanje relacijskim bazama podataka. NoSQL je model baze podataka koji ne koristi relacije i redundancije već automatski pohranjuje podatke na više memorijskih poslužitelja ako je potrebno. Taj način rada omogućava umetanje podataka u bazu podataka bez potrebe za prethodno definiranom shemom što omogućava rad bez prekida. Najveće razlike između SQL i NoSQL su jezik, skalabilnost, struktura, svojstva i podrška.

Razlike u jeziku: SQL programski jezik postoji 40 godina. Siguran je i svestran jezik koji ima široku upotrebu, ponajviše za kompleksne upite. Međutim SQL zahtjeva od korisnika striktne predefinirane tablične sheme i više vremena je potrebno za organiziranje strukture prije samog

početka korištenja. NoSQL koristi dinamičnu shemu uz veliku fleksibilnost. Manje se vremena posvećuje planiranju zbog potpune slobode dodavanja novih tablica i redova u bazu podataka. Međutim NoSQL ne sadrži većinu standarda koje SQL pruža, stoga kompleksniji upiti mogu biti zahtjevni za izvršiti.

Razlike u skalabilnosti: SQL baze podataka povećavaju kapacitet pohrane vertikalno na način da dodaju dodatne količine procesne snage postojećeg hardvera. NoSQL baze podataka rastu horizontalno, dodavanjem novih servera ili čvorova.

Razlike u strukturi: Shema SQL baze podatka uvijek predstavlja relacijske veze, tablične podatke uz pravila o dosljednosti i integritetu. NoSQL baze podataka obično pripadaju u jednu od četiri općih kategorija. Stupčasto orijentirana baza podataka transponira relacijsku bazu podataka usmjerenu na redove te omogućava pohranu velike količine podataka i pojedinačne zapise s promjenjivim atributima (Slika 2.1.).



Sl. 2.1. *Grafička usporedba SQL i NoSQL baza podataka*

U stupčastoj se bazi podaci pohranjuju od vrha do dna dok se u rednoj bazi pohranjuju s lijeva na desno (Slika 2.2.). Ključ – vrijednost model baze podataka koji pohranjuje podatke u obliku parova gdje je prvi član ključ, a drugi vrijednost koja je pridružena ključu. Dokumentna je baza podataka polustrukturna baza podataka dizajnirana za pohranu, prikazivanje i upravljanje podacima koji su različiti za svaki podatak. Grafička baza podataka dodaje koncept relacijske baze

podataka, direktnu vezu između objekata što omogućuje brzu pretragu i dobru povezanost podataka.

Sl. 2.2. *Usporedba stupčaste i baze podataka usmjerene na redove*

Razlike u svojstvima: Na visokim razinama i strukturama SQL i NoSQL imaju razdvojena pravila izvršavanja zahtjeva. Relacijske baze podataka moraju poštivati „ACID“ svojstva. Atomičnost (engl. *Atomicity*) - svi zahtjevi moraju biti uspješni ili neuspješni, ne smiju postojati djelomično izvršeni zahtjevi čak i u slučaju sistemskih grešaka. Konzistentnost (engl. *Consistency*) - u svakom koraku baza podataka može doći iz jednog validnog stanja samo u drugo validno stanje. Izolacija (engl. *Isolation*) - istovremene radnje ne smiju utjecati međusobno jedna na drugu. Konačni rezultat više istovremenih radnji mora biti jednak rezultatu dobivenom izvođenjem pojedinačnih radnji. Durabilnost (engl. *Durability*) - sve završene radnje na bazi podataka ostaju pohranjene i nepromijenjene u slučaju sistemskih grešaka. NoSQL baze podataka pridržavaju se „CAP“ teorema. CAP teorem kaže kako u bilo kojoj distribuiranoj bazi podataka samo dva svojstva mogu biti prisutna u istom trenutku. Dosljednost (engl. *Consistency*) - svaki zahtjev prima najnoviji rezultat ili grešku. Dostupnost (engl. *Availability*) - svaki zahtjev prima rezultat bez greške, neovisno je li primljeni podataka najnoviji podatak iz baze podataka. Tolerancija podjele (engl. *Partition tolerance*) - sustav mora nastaviti raditi unatoč kašnjenju ili gubitku podataka između čvorova.

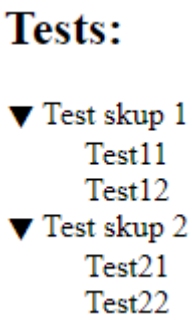
Razlike u podršci: SQL baze podataka predstavljaju veliku zajednicu, stabilne baze kodova i dokazane standarde. Različiti primjeri objavljeni su na internetu i stručnjaci mogu pomoći korisniku koji kreira novu bazu podataka za svoju primjenu. NoSQL tehnologije brzo se prihvaćaju, ali zajednica korisnika i programera trenutno broji malo članova i rastavljena je na frakcije. Međutim mnogo je SQL programskih jezika u vlasništvu velikih kompanija ili s njima

povezano, dok NoSQL zajednica koristi otvorene sustave i pomoć pri učenju i prilagođavanju sustavu novim korisnicima.

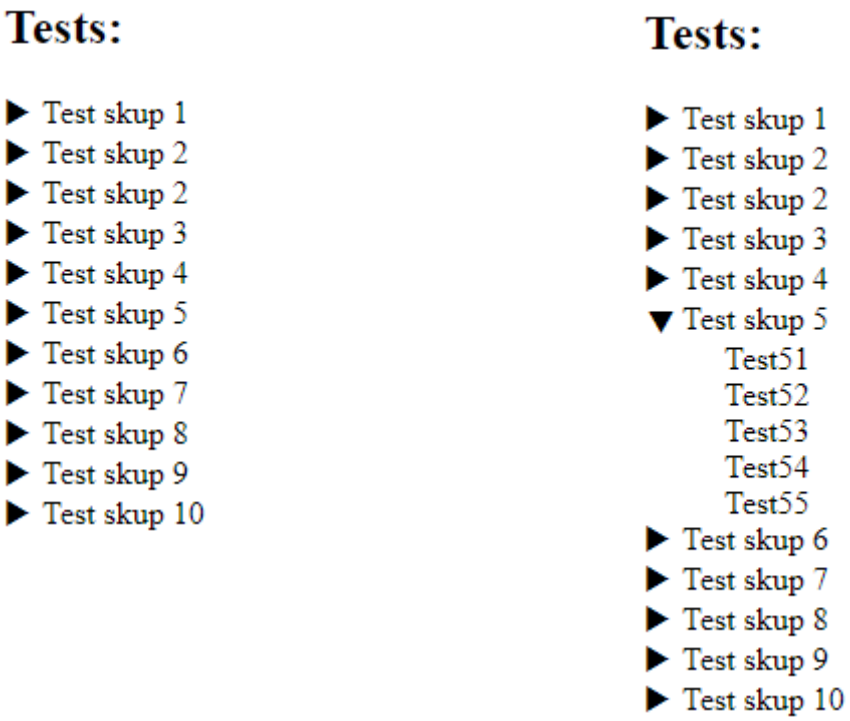
Općenito NoSQL baze podataka češće se koriste za: grafičku i hijerarhičnu strukturu podataka, podatke koji su veliki i količina im se brzo povećava, poslove koji rastu ekstremno brzo, ali nedostaje dobra shematska organizacija strukture za pohranu podataka. Društvene mreže, upravljanje online sadržajem, analiza toka podataka i mobilne aplikacije preferiraju NoSQL baze podataka. SQL je više primjeren za manje količine podataka, sustave gdje je konzistentnost ključna i modele gdje su prije početka sheme i tablice potpuno definirane. Male tvrtke, internet trgovine i transakcijske agencije koriste SQL. SQL baze podataka su u konačnici relacijske, imaju bolju podršku i otpornije su na greške. U ovom slučaju veću prednost imaju SQL baze podataka zbog definiranih konačnih ciljeva i potreba aplikacije te male mogućnosti za izmjenom konačnog proizvoda.

2.3. Vremenski nizovi, odzivi i primjene

Brzina slanja i primanja povratne informacije ključan je faktor uz sigurnost web aplikacije i baze podataka. Web aplikacija mora podržati određen broj istovremenih korisnika bez greške, prekida u radu ili prikazivanja i pohrane pogrešnih podataka. Slanje podataka iz baze u aplikaciju može se ostvariti u cjelini ili po dijelovima. Ako je zahtjev jednostavan i ne zahtijeva veću količinu resursa, izvršava se u potpunosti prema slici 2.3. dok se veći zahtjevi rastavljaju na dijelove te šalju i preuzimaju prilikom korisnikovog pritiska na točno određene dijelove web aplikacije. Npr. korisnik treba izvršiti 10 različitih skupova testova od kojih svaki sadrži pet testova. Korisnik na početku prima samo podatke o skupovima, a podatke o testovima ne prima. Nakon odabira jednog od ponuđenih skupova korisnik prima samo podatke o testovima iz tog skupa, dok ostale testove ne prima kako bi se ubrzao i optimizirao rad aplikacije (Slika 2.4.).



Sl. 2.3. *Prikaz svih testova kod malog broja testova*



Sl. 2.4. *Prikaz testova tek nakon odabira željenog skupa*

Slanje podataka može se izvršiti u drugom smjeru, iz aplikacije u bazu podataka. Poslani podaci mogu se slati periodično u točno definiranim intervalima ili nakon nekog neočekivanog podražaja kako bi se izvršile dodatne provjere, mjerenja i uspostavilo radi li sustav bez greške. Primjer podataka koji se pohranjuju u bazu podataka i obrađuju u realnom vremenu su podaci s različitih senzora u automobilu, prikupljanje prosječnih, ukupnih i trenutnih vrijednosti (npr. broj okretaja motora, potrošnja goriva, temperatura) te izračun stanja i potencijalnih problema kako bi se korisniku poslala odgovarajuća povratna informacija. Također moderni automobili sudjeluju u skupljanju podataka o prometnim nesrećama, međusobno komuniciraju (*V2X* komunikacija) i omogućuju tvrtkama kreiranje baze podataka o cestama i zgradama. Koriste se za izradu najmodernijih mapa koje omogućuju dodatne razine razvoja autonomne vožnje. U ovom slučaju aplikacija trenutno nije predviđena za pohranu vremenskih relacija nego za administrativne potrebe i izvršavanje točno određenih testova.

2.4. Ograničenja korištenih tehnologija

Najveći problemi koji se javljaju u odabranim tehnologijama za izradu web aplikacije su brzina odziva i broj paralelnih korisnika. Python Flask development server koji je zadano postavljen ne podržava izvođenje više od jednog istovremenog zahtjeva zbog čega dolazi do prekida u radu API-ja prilikom izvođenja više istovremena zahtjeva. Rješenje je zadanog problema višenitno procesiranje, prelazak na produkcijski server, dodavanje više paralelnih radnika (engl. *workera*) na server ili prelazak na Gunicorn ili WSGI server. Višenitno procesiranje omogućava se naredbom `app.run(threaded = True)` te omogućava više istovremenih procesa ovisno o broju aktivnih niti. WSGI server zbog mogućnosti dodavanja radnika omogućava više istovremenih procesa bez prekida rada aplikacije. Maksimalni je broj radnika za ovaj slučaj ((2 * (broj jezgri procesora)) – 1) što omogućava veći broj istovremenih procesa.

3. PRIMJENA BAZE PODATAKA U SUSTAVU ZA TESTIRANJE PROGRAMSKE PODRŠKE U AUTOMOBILSKOJ INDUSTRIJI

U ovom poglavlju opisani su zahtjevi na bazu podataka na temelju kojih je dizajniran konačni izgled baze. Implementirane su potrebne tablice i relacije između njih kako bi baza imala potrebnu povezanost. Omogućeno je dodavanje, pregled, uređivanje i brisanje podataka iz baze pomoću Python Flask programskog jezika u kojem je napravljen API.

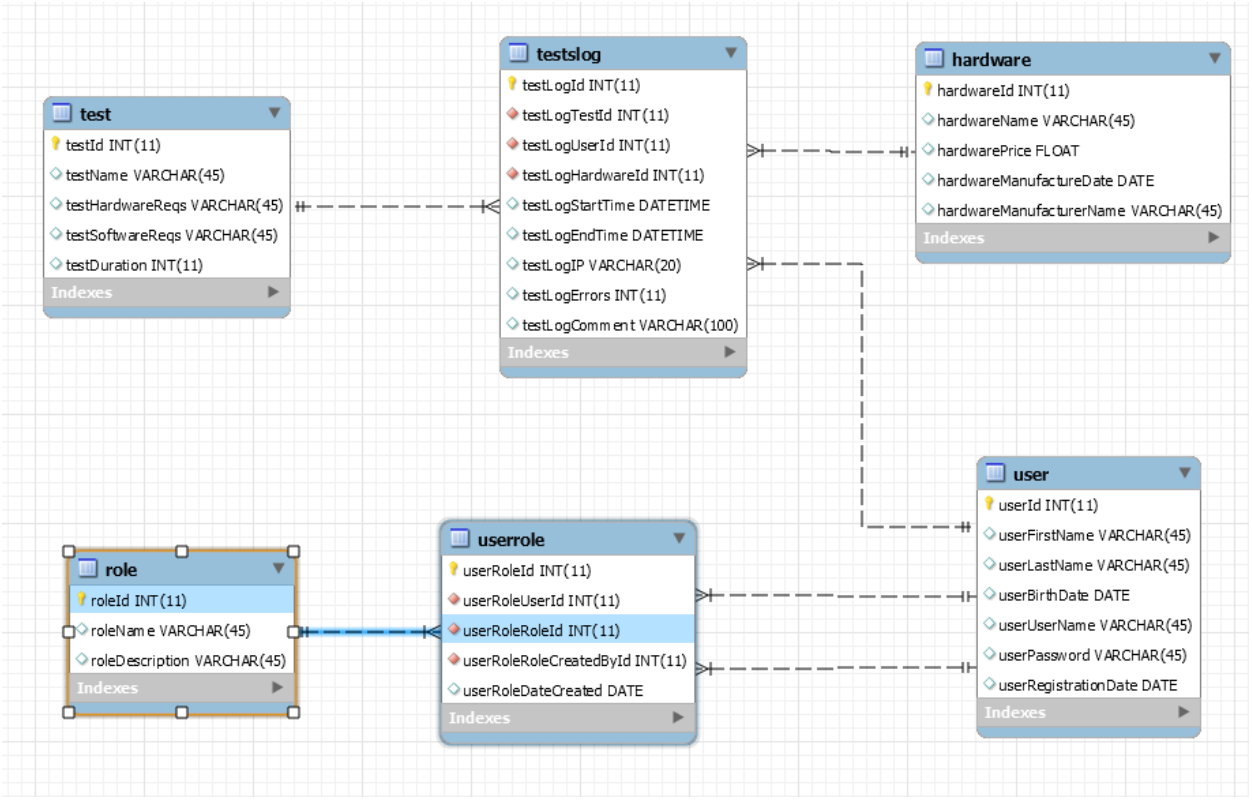
3.1. Zahtjevi baze podataka

U ovom radu korištena je MySQL baza podataka koja pripada u grupu SQL relacijskih baza podataka. Stoga bilo je potrebno napraviti što bolji dizajn iste kako bi se smanjila potreba za korekcijama i doradama na postojećem formatu baze. Baza ima mogućnost jednostavnog i brzog rukovanja podacima zbog jednostavnosti i preciznosti. U bazu trebaju biti pohranjeni podaci o korisnicima, testovima, ulogama koje su dodijeljene korisniku, hardveru na kojem je izvršeno testiranje i rezultatima testova.

3.2. Dizajn baze podataka

Za vrijeme dizajna potrebno je uzeti u obzir sve trenutne zahtjeve: mogućnost pohrane podataka o korisnicima, testovima, rezultatima testova, dodjeljivanje testova i opreme nad kojom se vrši testiranje, prikaz konačnih vremena po testovima te potencijalne buduće zahtjeve sustava kako bi se dizajnirala sveobuhvatna baza uz što manju potrebu za naknadnim modifikacijama. Dodatna jedna tablica ili čak jedan redak može utjecati na potrebu za promjenom velike količine dizajna baze podataka ako nije napravljeno po pravilima. Korisnik može imati jednu ili više uloga: administrator, menadžer, tester i preglednik. Administrator ima sve ovlasti u aplikaciji, menadžer upravlja testerima, dodjeljuje i pregledava konačne rezultate testova te i sam može izvršavati testove. Tester može pregledavati i izvršavati samo dodijeljene testove. Preglednik može samo pogledati konačne rezultate, grafove i tablice izvršenih testova. Uz ime hardvera, potrebni su podaci okvirna cijena, datum proizvodnje hardvera i ime proizvođača. Uzimajući sve navedeno u obzir, baza je dizajnirana tako da se sastoji od 6 tablica koje su međusobno povezane po pravilima o dizajnu i implementaciji relacijskih baza podataka (Slika 3.1.). Ovakvim dizajnom omogućavaju se jednostavna pohrana i pretraživanje konačnih podataka o rezultatima testova iz tablice `testlog`

uz sve potrebne podatke o testeru (zbog poveznice na tablicu *user* preko stranog ključa na *userId*), testovima (poveznica na tablicu *test* preko stranog ključa na *testId*) i hardveru na kojem je izvršeno testiranje (poveznica na tablicu *hardware* preko stranog ključa na *hardwareId*).



Sl. 3.1. *Dijagram baze podataka*

3.3. Implementacija baze podataka

Za potpuno funkcionalnu bazu podataka potrebno je implementirati prethodno kreiranu shemu baze podataka. Svaki korisnik mora imati jedinstveni korisnički broj (id) po kojem će se razlikovati od drugog. Za prijavu u sustav potrebni su ime, prezime, korisničko ime, datum rođenja, zaporka i datum izvršenja registracije u bazu podataka (Slika 3.2). U vezivnoj tablici korisnika i uloga (Slika 3.3.) unose se uloge dodijeljene korisniku, tko je dodijelio ulogu i datum dodjele za lakše praćenje i evidenciju. Svaki hardver na kojem se izvršava testiranje mora također biti pohranjen u bazu podataka. Predviđeni testovi unose se u bazu podataka uz ime testa, potrebnom verzijom hardvera, potrebnom verzijom softvera i predviđenim trajanjem testa. Rezultati i dodijeljeni testovi pohranjuju se u tablicu *testlog*. *Testlog* sadrži id testa kojeg se izvršava, id korisnika kojem

je test dodijeljen, id hardvera na kojem se izvršava testiranje, početno vrijeme izvršavanja, konačno vrijeme izvršavanja, IP adresu uređaja na kojem je izvršeno testiranje, broj grešaka i komentar testera na izvršeni test. Prije izvršavanja predviđenog testa zadnjih pet polja (početno i konačno vrijeme izvršavanja, IP testera, broj grešaka i komentar testera) automatski su stavljeni na NULL (prazno polje) kako bi se znalo da test nije izvršen (Slika 3.4.).

```
CREATE TABLE `user` (
  `userId` int(11) NOT NULL AUTO_INCREMENT,
  `userFirstName` varchar(45) DEFAULT NULL,
  `userLastName` varchar(45) DEFAULT NULL,
  `userBirthDate` date DEFAULT NULL,
  `userUserName` varchar(45) DEFAULT NULL,
  `userPassword` varchar(45) DEFAULT NULL,
  `userRegistrationDate` date DEFAULT NULL,
  PRIMARY KEY (`userId`)
)
CREATE TABLE `role` (
  `roleId` int(11) NOT NULL AUTO_INCREMENT,
  `roleName` varchar(45) DEFAULT NULL,
  `roleDescription` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`roleId`)
)
```

Sl. 3.2. *Kreiranje tablice korisnika i tablice uloga u bazi podataka*

```
CREATE TABLE `userrole` (
  `userRoleId` int(11) NOT NULL AUTO_INCREMENT,
  `userRoleUserId` int(11) NOT NULL,
  `userRoleRoleId` int(11) NOT NULL,
  `userRoleRoleCreatedById` int(11) NOT NULL,
  `userRoleDateCreated` date DEFAULT NULL,
  PRIMARY KEY (`userRoleId`),
  KEY `userRoleRoleIdFK_idx` (`userRoleRoleId`),
  KEY `userRoleUserIdFK_idx` (`userRoleUserId`),
  KEY `userRoleRoleCreatedByIdFK_idx` (`userRoleRoleCreatedById`),
  CONSTRAINT `userRoleRoleCreatedByIdFK` FOREIGN KEY
(`userRoleRoleCreatedById`) REFERENCES `user` (`userId`),
  CONSTRAINT `userRoleRoleIdFK` FOREIGN KEY (`userRoleRoleId`)
REFERENCES `role` (`roleId`),
  CONSTRAINT `userRoleUserIdFK` FOREIGN KEY (`userRoleUserId`)
REFERENCES `user` (`userId`)
)
```

Sl. 3.3. *Kreiranje vezivne tablice korisnika i uloga u bazi podataka*

```
CREATE TABLE `testslog` (
  `testLogId` int(11) NOT NULL AUTO_INCREMENT,
  `testLogTestId` int(11) NOT NULL,
  `testLogUserId` int(11) NOT NULL,
  `testLogHardwareId` int(11) NOT NULL,
  `testLogStartTime` datetime DEFAULT NULL,
  `testLogEndTime` datetime DEFAULT NULL,
```

```
  `testLogIP` varchar(20) DEFAULT NULL,
  `testLogErrors` int(11) DEFAULT NULL,
  `testLogComment` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`testLogId`),
  KEY `testHardwareIdFK_idx` (`testLogHardwareId`),
  KEY `tesTestIdFK_idx` (`testLogTestId`),
  KEY `testLogUserIdFK_idx` (`testLogUserId`),
  CONSTRAINT `testLogHardwareIdFK` FOREIGN KEY (`testLogHardwareId`)
REFERENCES `hardware` (`harwareId`),
  CONSTRAINT `testLogTestIdFK` FOREIGN KEY (`testLogTestId`)
REFERENCES `test` (`testId`),
  CONSTRAINT `testLogUserIdFK` FOREIGN KEY (`testLogUserId`)
REFERENCES `user` (`userId`)
)
```

Sl. 3.4. *Kreiranje tablice s dodijeljenim testovima*

3.4. Aplikacijsko programsko sučelje (API)

Komunikacija između baze podataka i sustava izvršena je pomoću Python Flask programskog jezika i Python SQLAlchemy [12] skupa alata (engl. *toolkit*). Poslani i primljeni podaci šalju se u JSON (JavaScript Object Notation) formatu. Python Flask ima mogućnost izvršavanja *request* metoda pomoću kojih obrađuje podatke primljene iz baze podataka ili iz sustava te ih pohranjuje i(li) šalje na potrebno mjesto ovisno o vrsti zadatka. Kreirani model baze podataka potrebno je također kreirati u SQLAlchemy modelu kako bi Python mogao ispravno obrađivati podatke. Ponovni dizajn modela nije potreban zbog toga što se koristi potpuno isti dizajn iz baze podataka. SQLAlchemy sadrži `sqlcodegen` koji je automatizirani alat za generiranje modela za MySQL, OurSQL, PostgreSQL i SQLite programske jezike. Pomoću naredbe kreiraju se potrebni modeli i pohranjuju u željenu datoteku.

```
sqlacodegen
mysql+pymysql://korisnik:zaporka@ipServera:port/imeBazePodataka>naziv
Datoteke.py
```

3.5. Implementacija API-ja

Nakon izvršene `sqlcodegen` naredbe kreira se datoteka, u ovom slučaju rtrk.py, koja sadrži sve potrebne informacije o bazi podataka, tablicama, redovima i vezama između tablica. Tablice (korisnici, uloge,...) pohranjuju se kao klase, dok su redovi iz tablice atributi klase u SQLAlchemy datoteci (rtrk.py). Na slikama 3.5, 3.6 i 3.7 prikazane su SQLAlchemy klase dobivene iz tablica na slikama 3.2. i 3.3. kreiranih u bazi podataka na slici 3.1..

```
class User(Base):
    __tablename__ = 'user'
    userId = Column(INTEGER(11), primary_key=True)
    userFirstName = Column(String(45))
    userLastName = Column(String(45))
    userBirthDate = Column(Date)
    userUserName = Column(String(45))
    userPassword = Column(String(45))
    userRegistrationDate = Column(Date)
    testslog = relationship('Testslog', back_populates='user')
    userrole = relationship('Userrole',
foreign_keys='[Userrole.userRoleRoleCreatedById]',
back_populates='user')
    userrole_ = relationship('Userrole',
foreign_keys='[Userrole.userRoleUserId]', back_populates='user_')
```

Sl. 3.5. *Sqlalchemy model tablice korisnika*

```
class Role(Base):
    __tablename__ = 'role'

    roleId = Column(INTEGER(11), primary_key=True)
    roleName = Column(String(45))
    roleDescription = Column(String(45))

    rolepermission = relationship('Rolepermission',
back_populates='role')
    userrole = relationship('Userrole', back_populates='role')
```

Sl. 3.6. *Sqlalchemy model tablice uloga*

```
class Userrole(Base):
    __tablename__ = 'userrole'
    __table_args__ = (
        ForeignKeyConstraint(['userRoleRoleCreatedById'],
['user.userId'], name='userRoleRoleCreatedByIdFK'),
        ForeignKeyConstraint(['userRoleRoleId'], ['role.roleId'],
name='userRoleRoleIdFK'),
        ForeignKeyConstraint(['userRoleUserId'], ['user.userId'],
name='userRoleUserIdFK')
    )
    userRoleId = Column(INTEGER(11), primary_key=True)
    userRoleUserId = Column(ForeignKey('user.userId'),
nullable=False, index=True)
    userRoleRoleId = Column(ForeignKey('role.roleId'),
nullable=False, index=True)
    userRoleRoleCreatedById = Column(ForeignKey('user.userId'),
nullable=False, index=True)
    userRoleDateCreated = Column(Date)
    user = relationship('User',
foreign_keys=[userRoleRoleCreatedById], back_populates='userrole')
    role = relationship('Role', back_populates='userrole')
```

```
    user_ = relationship('User', foreign_keys=[userRoleUserId],
back_populates='userrole_')
```

Sl. 3.7. *Sqlalchemy model vezivne tablice korisnika i uloga*

Za potpunu funkcionalnost sustava potrebno je implementirati *backend* dio za komunikaciju s bazom podataka. Python Flask korišten je za implementaciju, povezivanje s bazom te kreiranje i prekidanje korisničkih sesija. Prvo je potrebno kreirati *engine* pomoću `create_engine(„putanjaDoBazePodataka")`, povezati se na bazu podataka (`engine.connect()`), napraviti korisničku sesiju te nakon izvršavanja potrebnih zahtjeva prekinuti korisničku sesiju kako bi se oslobodila aplikacija za buduće zahtjeve (Slika 3.8.). Za potpunu funkcionalnost sustava potrebne su četiri metode: POST, GET, DELETE i PUT kojima su omogućene operacije čitanja, pisanja, uređivanja i brisanja (engl. *create, read, update, delete – CRUD*) nad bazom podataka. API sadrži funkcije za dodavanje korisnika, testova, hardvera i rezultata testova, uređivanje podataka o korisniku, testu, hardveru i rezultatu testa te pregled i brisanje svih navedenih podataka iz baze podataka (P.3.1.).

```
engine = create_engine("mysql+pymysql://root:@localhost/rtrk")
engine.connect()
DB_Session = sessionmaker(bind=engine)
db_session = DB_Session()
app = Flask(__name__)
db_session.close()
```

Sl. 3.8. *Kreiranje, pokretanje i prekidanje sesije na bazu podataka*

3.5.1. POST metoda

POST metoda temelji se na INSERT naredbi u SQL programskom jeziku koja služi za dodavanje novih podataka (redaka) u postojeće tablice iz baze podataka. U ovom slučaju, što ne mora uvijek biti slučaj u praksi, prema slici 3.9. podaci se šalju u JSON formatu, a primljeni podatak obrađuje se koristeći `request.json[]` funkciju. Zbog dodatne sigurnosti, korisničke zaporke ne pohranjuju se u originalnom formatu (passDiplomski), nego se koristi dodatna razina sigurnosti i kriptiranja. U ovom slučaju korišten je MD5 (engl. *Message-digest algorithm*) algoritam za kriptiranje podataka. MD5 je kriptografska funkcija koja prima poruku proizvoljne duljine, a za povratnu informaciju vraća ispis fiksne duljine. Dodana je još jedna dodatna razina sigurnosti pomoću proizvoljne sigurnosne riječi („TTTechAuto“ u ovom slučaju) koja se dodaje na kraj korisničke zaporke kako bi se dodatno osigurala i otežala mogućnost probijanja zaporke. Konačni je niz znakova, koji je zbroj korisničke zaporke i sigurnosne riječi (u ovom primjeru

„passDiplomskiTTTechAuto“). MD5 algoritam je jednosmjerni postupak koji omogućava dodatnu sigurnost ako neovlaštena osoba dođe u posjed baze podataka. Algoritam pretvara konačni niz znakova u novi zapis (npr. 8cede9abe6df05ea3dcad447400f9d31), pohranjuje u varijablu `hashedPassword` koja se pohranjuje u bazu podataka. Pomoću SQLAlchemy korisnički podaci pohranjuju se u objekt koji predstavlja novog korisnika te su spremni za dodavanje u bazu podataka. Ako je sve uspješno dodano dobiva se povratna informacija „0“ koja potvrđuje kako nema grešaka i sve je potpuno izvršeno, dok u bilo kojem drugom slučaju šalje se „1“ koja predstavlja negativnu povratnu informaciju i grešku u dodavanju korisnika (P.3.1.1). Nakon uspješno dodanog korisnika pojavljuje se novi zapis u bazi podataka s podacima tog istog korisnika (Tablica 3.1.).

```
{
    "userFirstName": "imeDiplomski",
    "userLastName": "prezimeDiplomski",
    "userPassword": "passDiplomski",
    "userUserName": "usernameDiplomski",
    "userBirthDate": "1988-02-27"
}
```

Sl. 3.9. *Primjer poslanih podataka zapisanih u strukturiranom obliku*

Tab. 3.1. *Primjer pohrane korisničkih podataka u bazi*

userId	userFirstName	userLastName	userBirthDate	userUserName	userPassword	userRegistrationDate
10	imeDiplomski	prezimeDiplomski	1988-02-27	usernameDiplomski	e75122eeff68c9dd11a4c1693b3e5d95	2022-07-19

3.5.2. GET metoda

GET metoda temelji se na SELECT naredbi u SQL programskom jeziku koja služi za pregled i izdvajanje željenih podataka iz baze podataka. U ovom slučaju povezivanjem na API poveznicu `'/user'` korisniku se prikazuju svi trenutno registrirani korisnici u bazi podataka te njihovi uneseni podaci. Pretraživanje podataka iz željene tablice u bazi podataka izvršava se korištenjem naredbe `nazivSesije.query(nazivModela.nazivTablice).all()` (P.3.1.2.). Konačna povratna informacija šalje rječnik (engl. *dictionary*) u JSON formatu zbog lakšeg obrađivanja podataka u *frontend* dijelu sustava (Slika 3.10.). Ako trenutno nema niti jednog korisnika unutar baze podataka, povratna informacija bit će prazan rječnik.

```
{
    "user": [
        {
            "userFirstName": "Luka",
            "userId": 4,
            "userLastName": "Kopic",
            "userPassword": "b604863610b2ede6a92c7d4384a49999",
            "userRegistrationDate": "Mon, 20 Dec 2021 00:00:00 GMT",
            "userUserName": "koparica"
        },
        {
            "userFirstName": "imeDiplomski",
            "userId": 10,
            "userLastName": "prezimeDiplomski",
            "userPassword": "8cede9abe6df05ea3dcad447400f9d31",
            "userRegistrationDate": "Tue, 19 Jul 2022 00:00:00 GMT",
            "userUserName": "usernameDiplomski"
        }
    ]
}
```


Sl. 3.10. *Primjer primljenih podataka o korisnicima*

Pomoću GET metode moguće je primiti podatke iz više različitih (odvojenih ili povezanih) tablica iz baze podataka te ih filtrirati ovisno o traženim uvjetima. U ovom slučaju korištena je funkcija `'userRole/<userId>'` koja omogućava pretraživanje i pregled dodijeljenih prava postojećim korisnicima (P.3.1.3.). Prvi korak isti je kao u jednostavnoj GET metodi (*query*), nakon čega se izvršava filtriranje podataka po zadanom korisničkom id-u kako bi se izdvojili samo podaci koji su vezani za korisnika od interesa. Filtriranje se vrši naredbom `filter(nazivModela.nazivTablice.userId == predaniUserId).first()`. `first()` služi za simultano izdvajanje samo prvog korisnika s traženim id-om iz baze podataka. Trenutna funkcija koristi `first()` zbog jedinstvenog korisničkog id-a u bazi podataka te nije potrebno pretraživati ostale korisnike što dovodi do uštede vremena. Nakon što se korisnik pronađe prelazi se na vezivnu tablicu između korisnika i uloga kako bi se pronašle sve uloge traženog korisnika (Tablica 3.2.). Filtriranje postojećih uloga korisnika izvršava se naredbom `nazivSesije.query(nazivModela.nazivVezivneTablice).filter(nazivModela.NazivVezivneTablice.userRoleUserId == predaniUserId).all()`. U ovom slučaju korišten je `all()` zbog toga što korisnik može imati više od jedne uloge (tester i gledatelj u ovom slučaju za korisnika s id-em 10 prema tablici 3.2.). Na kraju svi podaci spajaju se u jedan JSON zapis i dobiva se tražena povratna vrijednost (Slika 3.11.). Također, može se dogoditi kako traženi korisnik još nema niti jednu dodijeljenu ulogu. Tada je povratna vrijednost `User <ime prezime`

korisnika> doesn't have any roles!. Na kraju, ako traženi id ne postoji u bazi podataka povratna je vrijednost *Not existing user!*, kako bi se uočilo da ne postoji korisnik s traženim id-em.

Tab. 3.2. *Primjer podataka o ulogama i dodijeljene uloge korisniku*

roleId	roleName	roleDescription
1	Admin	User got all privileges
2	Manager	User manages all testers
3	Tester	User executes given tests
4	Viewer	User can only view test results

userRoleId	userRoleUserId	userRoleRoleId	userRoleRoleCreatedById	userRoleDateCreated
1	3	1	3	2022-02-27
2	4	1	4	2022-02-27
3	5	2	4	2022-02-27
4	10	3	4	2022-02-27
5	10	4	4	2022-02-27


```
{
    "user": [
        {
            "roles": [
                {
                    "roleDescription": "User executes given tests",
                    "roleGivenBy": "Luka Kopic(koparica)",
                    "roleGivenDate": "Sun, 27 Feb 2022 00:00:00 GMT",
                    "roleName": "Tester"
                },
                {
                    "roleDescription": "User can only view test results",
                    "roleGivenBy": "Luka Kopic(koparica)",
                    "roleGivenDate": "Sun, 27 Feb 2022 00:00:00 GMT",
                    "roleName": "Viewer"
                }
            ],
            "userFirstName": "imeDiplomski",
            "userLastName": "prezimeDiplomski"
        }
    ]
}
```


Sl. 3.11. *Primjer primljenih podataka za postojećeg korisnika koji ima dodane uloge*

3.5.3. DELETE metoda

Svaki podatak u bazi treba imati mogućnost brisanja. Brisanje se izvršava DELETE metodom. U nastavku dan je primjer brisanja pojedinog korisnika iz baze podataka. Adresa traženog poziva jest *'/user/<userId>'*. Kao u prethodnoj GET metodi, prvo se izvršava filtriranje željenog korisnika iz baze podataka pomoću id korisnika. Ako korisnik sa željenim id-em ne postoji šalje se povratna informacija *Not existing user!*. Python Flask naredbom *filtriraniKorisnik.delete()* pokušava obrisati postojećeg korisnika. Za uspješno izbrisanog korisnika šalje se „0“ kao povratna informacija dok za prisutnost bilo kakve pogreške „1“ (P.3.1.4.).

3.5.4. PUT metoda

Za kraj potrebno je omogućiti uređivanje postojećih podataka. Uređivanja se izvršavaju PUT metodom, UPDATE u SQL programskom jeziku. U ovom primjeru prikazan je postupak promjene korisničke zaporke. Adresa je funkcije `'/user/<userId>'` (P.3.1.5.). Nakon što korisnik unese trenutnu i željenu novu zaporku podaci se šalju u JSON formatu (Slika 3.12.). Prvi je korak jednostavno filtriranje korisnika i traženje postoji li korisnik s predanim id-em. Za postojećeg korisnika provjerava se je li predana trenutna zaporka točna. Provjera se vrši na način da se predanoj zaporki dodaje „TTTechAuto“, ukupni niz znakova kriptira pomoću MD5 algoritma i uspoređuje s hash zapisom zaporke iz baze podataka. Ako se zaporke ne podudaraju, korisnik dobiva povratnu informaciju `Wrong password!` te ne može promijeniti zaporku. Postupak kriptiranja nove zaporke identičan je kao kod dodavanja novog korisnika. Nakon kriptiranja nove zaporke vrši se izmjena polja hash zaporke te se pokušava urediti polje postojeće hash zaporke trenutnog korisnika u bazi podataka (Tablica 3.3.). Za uspješno izvršen cijeli proces šalje se „0“ kao povratna informacija dok „1“ za bilo koju vrstu pogreške.

```
{
	"userPassword": "passDiplomski",
	"userNewPassword": "novaLozinka"
}
```

Sl 3.12. *Primjer poslanih podataka kod izmjene korisničke zaporke*

Tab. 3.3. *Primjer korisničkih podataka u bazi prije i poslije promjene zaporke*

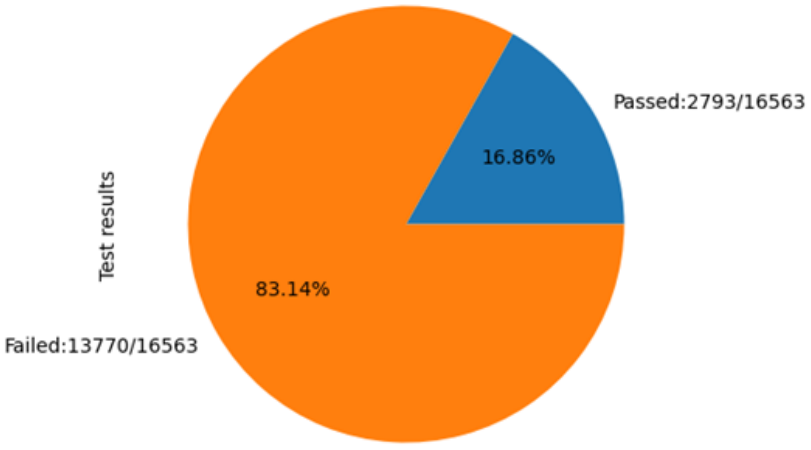
userId	userFirstName	userLastName	userBirthDate	userUserName	userPassword	userRegistrationDate
10	imeDiplomski	prezimeDiplomski	1988-02-27	usernameDiplomski	86ce2b7f7bea44513243653b3cd45660	2022-07-19
10	imeDiplomski	prezimeDiplomski	1988-02-27	usernameDiplomski	8cede9abe6df05ea3dcad447400f9d31	2022-07-19

Nakon dodanih korisnika i testova potrebno je testeru dodijeliti testove za izvršavanje i hardver na kojem će se testovi izvršiti. Menadžer pomoću funkcije iz API-ja pregledava sve postojeće testove, korisnike i hardver te dodjeljuje testove testerima. Nakon što je dodijelio testove poziva se funkcija za pohranu podataka o dodijeljenom testu, testeru i hardveru u bazu podataka u `testlog` tablicu. Tester također koristeći funkciju iz API-ja dobiva sve podatke o dodijeljenim testovima. Prilikom početka izvršavanja testa pohranjuje se početno vrijeme izvršavanja u varijablu `testLogStartTime`, Kada tester izvrši testiranje, upiše konačne rezultate i zaključa test

pohranjuje se konačno vrijeme izvršavanja u varijablu `testLogEndTime`. Nakon zaključanog testa poziva se funkcija za pohranu rezultata testa u bazu podataka. Radi lakšeg pregleda i obrade konačnih rezultata testova, dodijeljenih testera, vremena početka i kraja izvršenih testova, IP adresu s koje je tester izvršio test te hardver na kojem je testiranje izvršeno prikazani su u grafičkom (kružni dijagram) obliku te u MS Excel-u. Python Pandas biblioteka omogućava izvršavanje oba zahtjeva funkcija za ispis (engl. *export*) rezultata. Za ispis rezultata u MS Excel izvršava se filtriranje i obrada podataka kao u GET metodi te pohrana potrebnih podataka u nizove. Nakon toga željeni nizovi pohranjuju se u pojedinačne stupce MS Excel aplikacije pomoću `pandas.DataFrame({„nazivStupca1“:nazivNiza1,„nazivStupca2“:nazivNiza2, „nazivStupca3“:nazivNiza3}).to_excel(nazivDatoteke.xlsx,sheet_name=nazivLista,index=False)` (P.3.1.6.). Ako je test izvršen, prikazat će se svi potrebni podaci. Ako je test dodijeljen testeru, a nije izvršen, polja za početno vrijeme, krajnje vrijeme, rezultat, IP adresu testera i komentar bit će prazna (Tablica 3.4.). Također, korisnik može pogledati i preuzeti rezultate izvršenih testova, u ovom slučaju samo prolaz (engl. *passed*) i pad (engl. *failed*), iz web preglednika s adrese `'/testResultChart'` (P.3.1.7.). Python Flask filtrira sve rezultate testova te računa ukupan broj pozitivnih i negativnih rezultata te postotak u odnosu na ukupan broj testova. Pomoću `pandas` biblioteke kreira se kružni dijagram predanih veličina, naziva polja i naziva cjelokupnog dijagrama. Kreirani dijagram pohranjuje se u datoteku Preuzimanje (engl. *Downloads*) na korisničkom računalu te prikazuje u web pregledniku (Slika 3.13.).

Tab. 3.4. *Primjer ispisa podataka o testovima u MS Excel*

	A	B	C	D	E	F	G	H	I
1	testLogId	test(id)	user(id)	hardware(id)	startTime	endTime	ip	errors	comment
2	1	Basic test(1)	imeDiplomski prezimeDiplomski(10)	Handle(1)	2022-07-20 14:05:06	2022-07-20 14:06:12	1.1.1.1	1	Failed start
3	2	Basic test(1)	imeDiplomski prezimeDiplomski(10)	Handle(1)	2022-07-20 14:07:03	2022-07-20 14:09:15	1.1.1.1	0	
4	3	Basic test(1)	imeDiplomski prezimeDiplomski(10)	Handle(1)					
5	4	Basic test(1)	imeDiplomski prezimeDiplomski(10)	Handle(1)					



Sl. 3.13. *Primjer primljene povratne informacija prethodne funkcije*

4. ISPITIVANJE FUNKCIONALNOSTI API-JA I BAZE PODATAKA

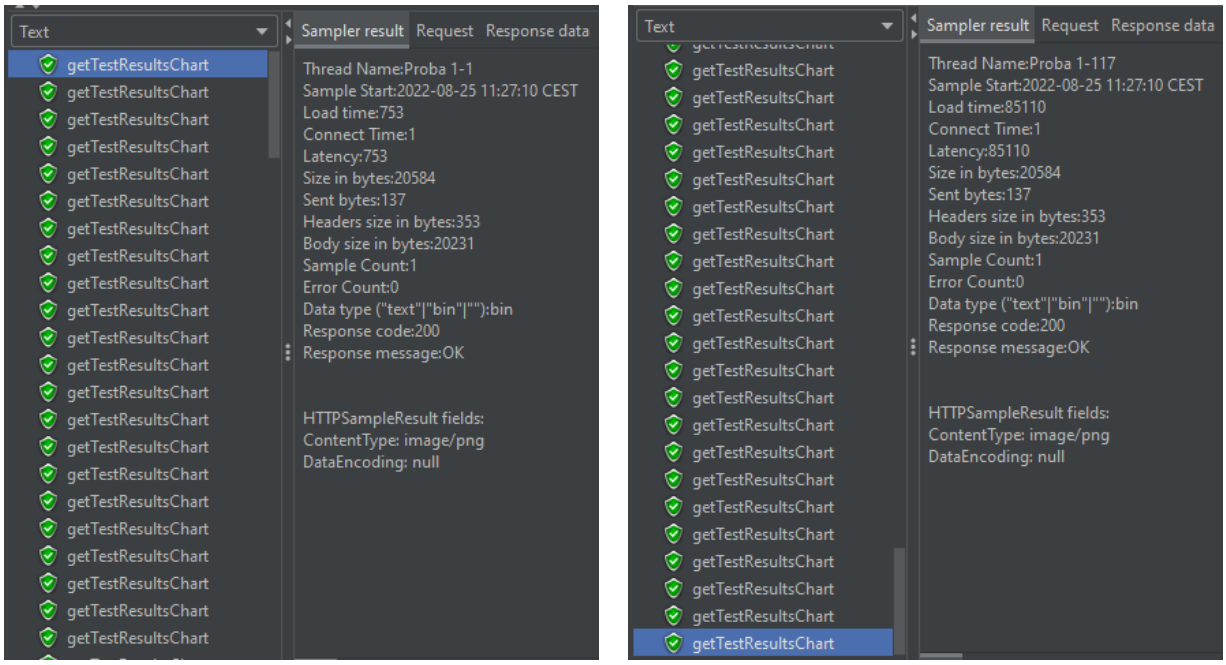
Na kraju svakog dijela razvoja sustava i baze podataka potrebno je izvršiti veliku količinu testova kako bi pronašli i uklonili što veći broj grešaka i problema. U ovom dijelu rada predstavljeni su rezultati izvršenih pojedinačnih (engl. *unit*), integracijskih, cjelokupnih i ad-hoc (nasumična testiranja bez plana) manualnih i automatskih testiranja. Svi su pronađeni problemi uklonjeni i ispravljeni te su baza podataka i API spremni za objavljivanje prve verzije na kojoj će se simultano raditi nadogradnje i ispravci primijećenih problema. U pojedinačnom testu testirani su najmanji dijelovi koda, API-ja i funkcionalnosti. Nakon što su svi pojedinačni testovi prošli s pozitivnom ocjenom prešlo se na integracijsko testiranje. U integracijskom testiranju testirano je nekoliko povezanih cjelina, komunikacija između istih, protok podataka te jesu li svi dijelovi obradili, pročitali i(li) pohranili željene podatke bez greške. Poslije završetka testiranja dijelova sustava izvršeno je testiranje cjelokupnog sustava, gdje se nije odvajao niti jedan pojedinačni dio, nego je testirano sve kao cjelina. Za kraj, izvršeno je ad-hoc testiranje gdje su testeri, programeri i ostali korisnici izvršili testiranje bez plana i programa te pokušali pronaći što više neotkrivenih pogrešaka. Testiranja su izvršena na sustavu koji sadrži: Intel Core i7-6700 3.40GHz 4c/8t uz Intel HD Graphics 530, 16GB DDR4 2133MHz i ADATA SX8200PNP 256GB SSD, Windows 10 operacijski sustav, MySQL Community Server (GPL) 5.7.31, Python 3.9.10, Flask 2.0.3, Werkzeug 2.0.3.

4.1. Ručna testiranja

Ručna (engl. *manual*) su testiranja testiranja koje izvršava čovjek (tester, programer, korisnik) po zadanim koracima nakon kojih uspoređuje dobivene rezultate s predviđenim rezultatima. Ako se dobiveni rezultat razlikuje od predviđenog rezultata smatra se da je test pao. Tada je potrebno napisati konačni rezultat, način kako se došlo do greške te kratki komentar testera. U suprotnom je test prošao i potvrdio valjanost testiranog dijela proizvoda. U ovom slučaju izvršeni su pojedinačni, integracijski, cjelokupni i ad-hoc testovi. Testiranja su izvršena direktno u sustavu ili pomoću aplikacije Postman [13] koja simulira rad sustava bez potrebe za izradom korisničkog sučelja aplikacije. Primjer pojedinačnog testa odnosio se na dodavanje novog i brisanje postojećeg korisnika, integracijski test izvršavanje dodijeljenih testova i unošenje rezultata dok je cjelokupno testiranje bilo sve od početka kreiranja korisnika, testova, izvršavanja do generiranja i obrađivanja rezultata. Nakon izvršenih otprilike 50 testova uklonjeno je 10 grešaka.

4.2. Automatska testiranja

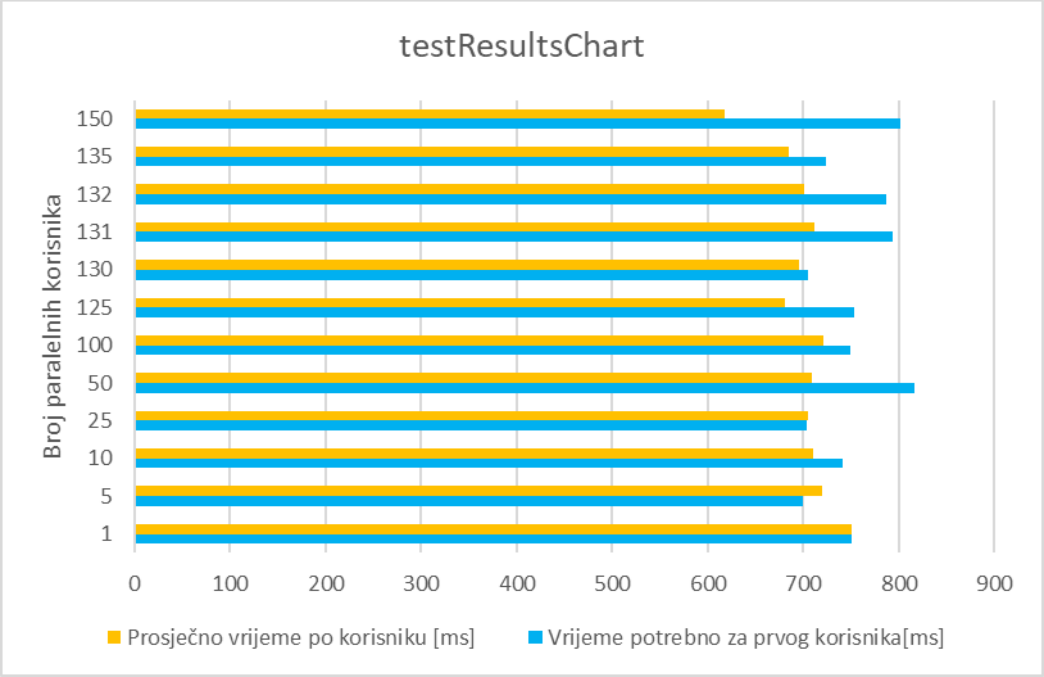
Automatska testiranja izvršena su u aplikaciji Apache JMeter [14] koja omogućava simuliranje većeg broja korisnika i različitih zahtjeva na API-je (Slika 4.1.). Zbog bržeg rada i odziva baze podataka nego Python Flask-a izvršeni testovi proučavali su maksimalnu izdržljivost Python Flask API-ja i lokalnog servera s obzirom na broj istovremenih korisnika. Za potrebe rada izvršena su testiranja dvije funkcije: `'/testResultsChart'` i `'userRole/<userId>'`. U sučelju aplikacije Apache JMeter dodjeljeni su parametri za broj istovremenih korisnika kojim se želi opteretiti testirani API, adresa funkcije, vremenski razmak između dva zahtjeva te broj iteracija izvođenja testiranja. Nakon izvršenih zahtjeva u aplikaciji, primile su se informacije o uspješnosti zahtjeva, vremenskom izvršavanja i povratnoj informaciji funkcije. U tablici 4.1. i slikama 4.2.a), 4.2.b), 4.3.a) i 4.3.b) predstavljeni su postignuti rezultati prethodnih testiranja. Funkcija za grafički prikaz mnogo je veća i zahtjevnija od funkcije za prikaz uloga korisnika, stoga je i broj paralelnih korisnika bez pojave greške manji te vrijeme izvršavanja po pojedinačnom korisniku veći. Prva funkcija (grafički prikaz rezultata u kružnom dijagramu) podržava maksimalno 130 istovremenih zahtjeva različitih korisnika uz prosječno vrijeme izvršavanja zahtjeva po korisniku od 704,83 ms. Funkcija za prikaz uloga podržava maksimalno 350 istovremenih zahtjeva različitih korisnika uz prosječno vrijeme izvršavanja zahtjeva po korisniku od 10,22 ms. Prilikom 350 istovremenih zahtjeva na drugu funkciju javlja se granični slučaj. U graničnom slučaju funkcija se može izvršiti bez greške, ali se mogu pojaviti i greške (dvije i tri u ovom slučaju). Prosječno vrijeme izvršavanja zahtjeva po korisniku odnosi se samo na testove izvršene bez grešaka. U slučaju istovremenog povezivanja više od maksimalnog broja dopuštenih korisnika pojavljuje se trenutna povratna informacija koja prikazuje grešku. Zbog trenutnog odziva (0 sekundi umjesto 704,83 ms za prvu funkciju) prosječno vrijeme izvršavanja funkcije po korisniku smanjuje se. Iz dobivenih rezultata može se zaključiti da prosječno vrijeme po korisniku i vrijeme potrebno za prvog korisnika ne ovisi o broju korisnika dok se povećanjem broja korisnika povećava ukupno vrijeme izvršavanja i broj grešaka.



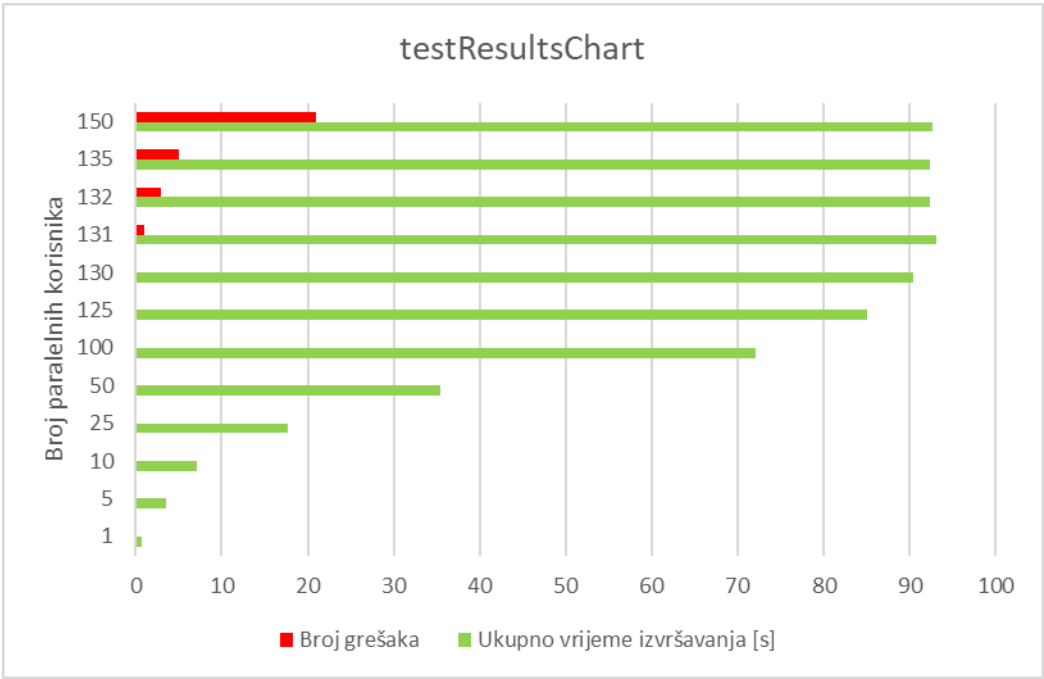
Sl. 4.1. *Prikaz rezultata izvršenog testa 125 istovremenih korisnika funkcije '/testResultsChart'*

Tab 4.1. *Rezultati testova opterećenosti funkcija*

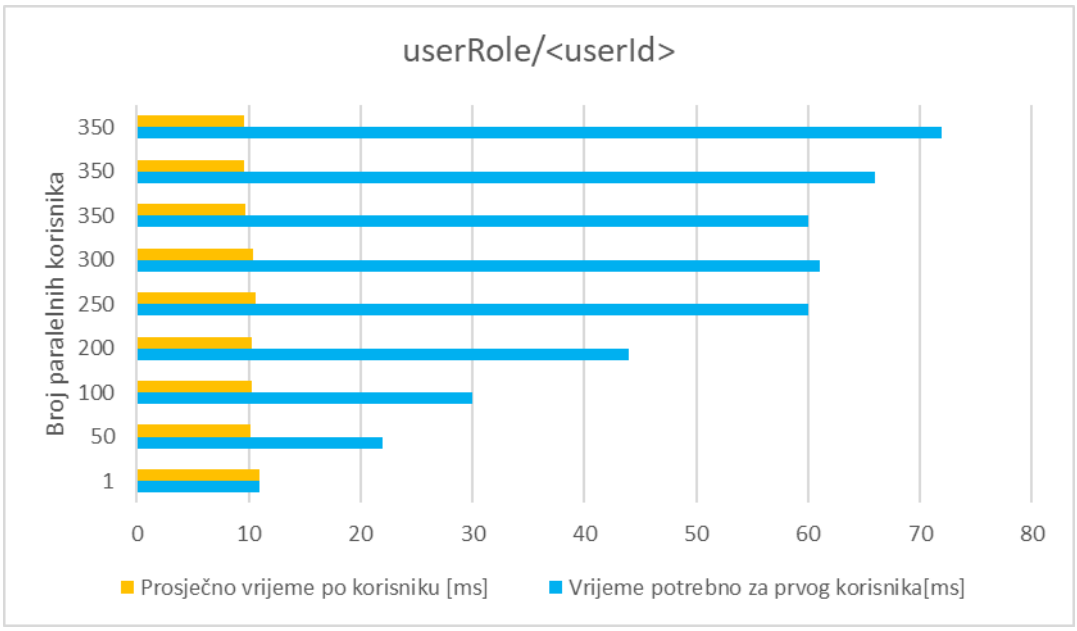
Naziv API-ja	Broj paralelnih korisnika	Vrijeme potrebno za prvog korisnika[ms]	Ukupno vrijeme izvršavanja [ms]	Prosječno vrijeme po korisniku [ms]	Broj grešaka
testResultsChart	1	750	750	750	0
testResultsChart	5	700	3600	720	0
testResultsChart	10	741	7100	710	0
testResultsChart	25	703	17629	705,16	0
testResultsChart	50	816	35419	708,38	0
testResultsChart	100	749	72124	721,24	0
testResultsChart	125	799	87264	698,112	0
testResultsChart	130	705	90470	695,923	0
testResultsChart	131	794	93160	711,145	1
testResultsChart	132	787	92420	700,152	3
testResultsChart	135	723	92425	684,63	5
testResultsChart	150	802	92650	617,667	21
userRole/<userId>	1	11	11	11	0
userRole/<userId>	50	22	511	10,22	0
userRole/<userId>	100	30	1030	10,3	0
userRole/<userId>	200	44	2067	10,335	0
userRole/<userId>	250	60	2645	10,58	0
userRole/<userId>	300	61	3118	10,393	0
userRole/<userId>	350	60	3402	9,72	0
userRole/<userId>	350	66	3339	9,54	2
userRole/<userId>	350	72	3340	9,54286	3



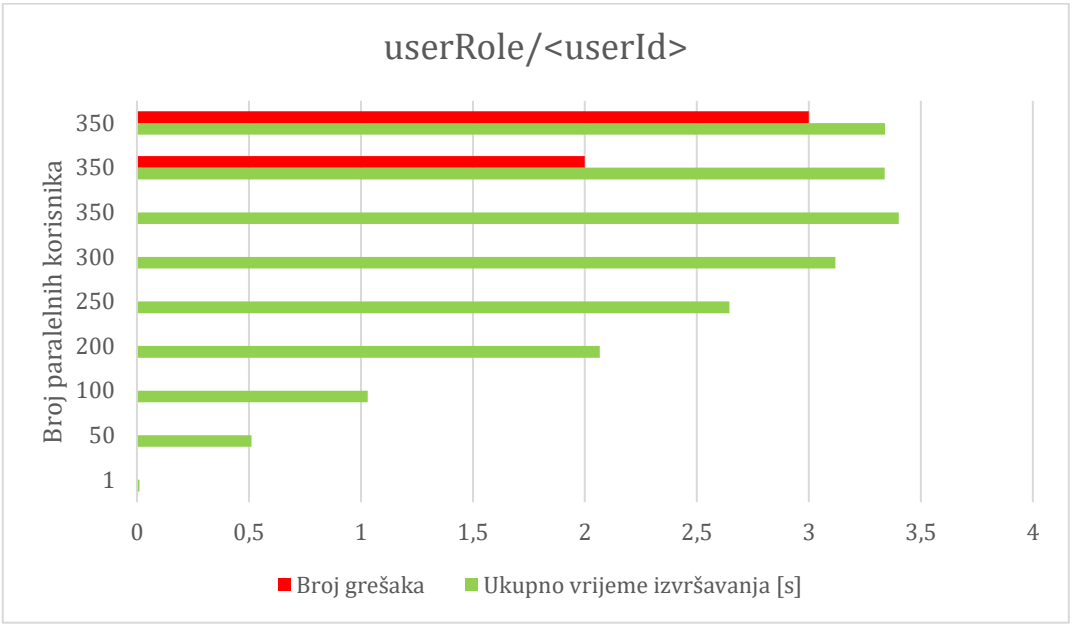
Sl. 4.2.a) *Grafički prikaz rezultata testiranja funkcije '/testResultChart': prosječno vrijeme po korisniku i vrijeme potrebno za prvog korisnika ovisno o broju različitih korisnika*



Sl. 4.2.b) *Grafički prikaz rezultata testiranja funkcije '/testResultChart': broj grešaka i ukupno vrijeme izvršavanja ovisno o broju različitih korisnika*



Sl. 4.3.a) *Grafički prikaz rezultata testiranja funkcije '/userRole/<userId>': prosječno vrijeme po korisniku i vrijeme potrebno za prvog korisnika ovisno o broju različitih korisnika*



Sl. 4.3.b) *Grafički prikaz rezultata testiranja funkcije '/userRole/<userId>': broj grešaka i ukupno vrijeme izvršavanja ovisno o broju različitih korisnika*

5. ZAKLJUČAK

Zadatak ovog rada bio je izraditi bazu podataka koja će omogućiti unos, uređivanje i prikazivanje potrebnih podataka o korisnicima, testovima, hardveru i konačnim rezultatima testova. Podaci se uređuju pomoću web aplikacije te se moraju omogućiti ispis grafova, tablica i svih ostalih konačnih rezultata. Korisniku mora biti omogućena prijava u sustav te izmjena postojeće za novu zaporku. Baza podataka kreirana je u MySQL programskom jeziku dok su za potrebe izrade web aplikacije korišteni Python Flask, Python Django, SQLAlchemy, HTML, CSS i JavaScript. Sva komunikacija između baze podataka i web aplikacije realizirana je putem aplikacijskog programskog sučelja (engl. *Application Programming Interface* – API) u Python Flask programskom jeziku. Zahtjevi na bazu podataka ili na primljene podatke iz web aplikacije obrađuju se `request()` metodama, a poslani su podaci u oba smjera u JSON formatu. Sustav omogućava prijavu pojedinačnom korisniku, preusmjerenje na ekran ovisno o ulozi i promjenu trenutne zaporke za prijavu. Uz to, sustav omogućava jednostavno dodavanje korisnika, testova i rezultata te filtriranje, uređivanje, brisanje i prikazivanje konačnih rezultata u obliku kružnih dijagrama i MS Excel tablica. Sustav služi za testiranje (polu)proizvoda i programske podrške u automobilskoj industriji. Postoje četiri vrste uloge koje korisnik može imati: administrator, menadžer, tester i gledatelj. Administrator ima sve ovlasti i upravlja radom aplikacije. Menadžer upravlja testovima i testerima, dodjeljuje jedne drugima te može pregledavati trenutno stanje testova i konačne rezultate izvršenih testova. Tester može samo izvršiti i pregledati dodijeljene testove, dok gledatelj ima samo pravo pregleda konačnih rezultata izvršenih testova. Izvršeni testovi pohranjuju rezultat izvršavanja, komentar testera, IP adresu testera, vrijeme početka i kraja izvršavanja testiranja, hardver koji je testiran te podatke o testeru koji je izvršio testiranja. Nakon izvršenog velikog broja testova, ispravljenih grešaka i dodanih novih funkcionalnosti baza podataka i API spremni su za isporuku u prvoj javnoj verziji. Izvršena su pojedinačna, cjelokupna i *ad-hoc* ručna i automatska testiranja. Za testiranja korišteni su web preglednik, Postman i Apache JMeter aplikacije. U budućnosti moguće je dodati nove funkcionalnosti ako se sustav pokaže kao koristan i učinkovit.

LITERATURA

[1] MySQL, https://dev.mysql.com/doc/ , kolovoz 2022.

[2] Python Flask, https://flask.palletsprojects.com/en/2.1.x/ , kolovoz 2022.

[3] Python Django, https://docs.djangoproject.com/en/4.0/ , kolovoz 2022.

[4] HTML, https://www.w3schools.com/html/ , kolovoz 2022.

[5] JavaScript, https://www.w3schools.com/js/ , kolovoz 2022.

[6] CSS, https://www.w3schools.com/css/ , kolovoz 2022.

[7] Automotive Testing and Development Services, Inc., https://automotivetesting.com/ , kolovoz 2022.

[8] SGS, https://www.sgs.com/ , kolovoz 2022.

[9] UserTesting, https://www.usertesting.com/ , kolovoz 2022.

[10] Brooks, https://www.brooksrunning.com/ , kolovoz 2022.

[11] Pinecone Research, https://members.pineconeresearch.com/ , kolovoz 2022.

[12] SQLAlchemy, https://docs.sqlalchemy.org/en/14/ , kolovoz 2022.

[13] Postman, https://www.postman.com/ , kolovoz 2022.

[14] Apache JMeter, https://jmeter.apache.org/ , kolovoz 2022.

SAŽETAK

U ovom diplomskom radu opisana je primjena baze podataka u sustavu za testiranje programske podrške u automobilskoj industriji. Na početku rada opisane su tehnologije korištene za izradu baze podataka i web aplikacije, postojeća rješenja problema i usporedbe između NoSQL i SQL baze podataka te koji je model pogodniji za izvršavanje postojećih zahtjeva. U trećem poglavlju prikazani su dizajn i implementacija baze podataka i aplikacijsko programskog sučelja (API). Opisane su i prikazane metode za dodavanje, brisanje, pregled i uređivanje podataka iz baze podataka pomoću aplikacijsko programskog sučelja implementiranog u Python Flask programskom jeziku. Četvrto poglavlje sadrži testiranje dijelova i cjelokupnog sustava, tablični i grafički prikaz rezultata testova. Na kraju rada nalaze se zaključak cijelog rada i priloženi kodovi za implementaciju aplikacijsko programskog sučelja.

Ključne riječi: aplikacijsko programsko sučelje, automotiv, baza podataka, testiranje, web aplikacija

ABSTRACT

In this master thesis was described application of the database in the system for testing software support in automotive industry. At the beginning of the work, the technologies used to create the database and web application are described, existing solutions to problems and comparisons between NoSQL and SQL databases and which model is more suitable for the execution of existing requirements. The third chapter presents the design and implementation of the database and the application programming interface (API). Methods for adding, deleting, viewing and editing data from the database using the application programming interface implemented in the Python Flask programming language are described and presented. The fourth chapter contains testing of parts and the entire system, tabular and graphical presentation of test results. At the end of the thesis, there is a conclusion of the entire thesis and attached codes for the implementation of the application programming interface.

Key words: application programming interface, automotive, database, testing, web application

ŽIVOTOPIS

Luka Kopić rođen je 23. svibnja 1994. godine u Osijeku. 2013. godine završava Isusovačku klasičnu gimnaziju s pravom javnosti u Osijeku. 2018. godine upisuje preddiplomski sveučilišni studij smjer računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. 2020. godine završava preddiplomski studij te upisuje diplomski sveučilišni studij automobilskog računarstva i komunikacija.

PRILOZI

P.3.1. Popis svih API metoda i funkcija:

Naziv	**Metoda**	**Opis**
'/user'	GET	Ispis svih korisnika iz baze podataka
'/user'	POST	Dodavanje novog korisnika u bazu podataka
'/user/<userId>'	DELETE	Uklanjanje predanog korisnika iz baze podataka
'/user/<userId>'	PUT	Uređivanje podataka iz baze podataka o predanom korisniku
'/user/<userId>'	GET	Ispis svih podataka iz baze podataka o predanom korisniku
'/test'	GET	Ispis svih testova iz baze podataka
'/test'	POST	Dodavanje novog testa u bazu podataka
'/test/<testId>'	DELETE	Uklanjanje predanog testa iz baze podataka
'/test/<testId>'	PUT	Uređivanje podataka iz baze podataka o predanom testu
'/test/<testId>'	GET	Ispis svih podataka iz baze podataka o predanom testu
'/hardware'	GET	Ispis svih hardvera iz baze podataka
'/hardware'	POST	Dodavanje novog hardvera u bazu podataka
'/hardware/<hardwareId>'	DELETE	Uklanjanje predanog hardvera iz baze podataka
'/hardware/<hardwareId>'	PUT	Uređivanje podataka iz baze podataka o predanom hardveru
'/hardware/<hardwareId>'	GET	Ispis svih podataka iz baze podataka o predanom hardveru
'/userRole/<userId>'	GET	Ispis svih uloga korisnika iz baze podataka
'/userRole/<userId>'	POST	Dodavanje nove uloge korisniku u bazu podataka
'/userRole/<userId>'	DELETE	Uklanjanje predane uloge korisnika iz baze podataka
'/testLog'	GET	Ispis svih rezultata testova iz baze podataka
'/testLog'	POST	Dodavanje novog rezultata testa u bazu podataka
'/testLog/<testLogId>'	DELETE	Uklanjanje predanog rezultata testa iz baze podataka
'/testLog/<testLogId>'	PUT	Uređivanje podataka iz baze podataka o predanom rezultatu testa
'/testLog/<testLogId>'	GET	Ispis svih podataka iz baze podataka o predanom rezultatu testa

'/testResultsChart'	GET	Ispis rezultata svih izvršenih testova u kružni dijagram
'/exportResultsToExcel'	GET	Ispis svih podataka o dodijeljenim i izvršenim testovima u MS Excel

P.3.1.1. Implementacije POST metode za pohranu podataka o korisniku

```
@app.route('/user', methods=['POST'])
def postUser():
    userUserName = request.json['userUserName']
    userFirstName= request.json['userFirstName']
    userLastName= request.json['userLastName']
    userPassword= request.json['userPassword']
    userBirthDate = request.json['userBirthDate']
    userRegistrationDate = datetime.datetime.now()
    salt = "TTTechAuto"
    hashedPassword = hashlib.md5((userPassword +
salt).encode()).hexdigest()
    user = rtrk.User(userUserName = userUserName, userFirstName =
userFirstName, userLastName = userLastName, userRegistrationDate =
userRegistrationDate, userPassword = hashedPassword, userBirthDate =
userBirthDate)
    try:
        db_session.add(user)
        db_session.commit()
        db_session.close()
        return "0"
    except:
        db_session.close()
        return "1"
```

P.3.1.2. Implementacija GET metode za pregled podataka o korisnicima

```
@app.route('/user', methods=['GET'])
def getUser():
    users = db_session.query(rtrk.User).all()
    output = []
    for user in users:
        data = {'userId': user.userId,'userPassword':
user.userPassword,'userFirstName': user.userFirstName,'userLastName':
user.userLastName,'userUserName':
user.userUserName,'userRegistrationDate': user.userRegistrationDate}
        output.append(data)
    return{"user": output}
```

P.3.1.3. Implementacija GET metode za filtriranje uloga korisnika

```
@app.route('/userRole/<userId>', methods = ['GET'])
def getUserRole(userId):
    user = db_session.query(rtrk.User).filter(rtrk.User.userId ==
userId).first()
    if user:
        output = []
        userRoles =
db_session.query(rtrk.Userrole).filter(rtrk.Userrole.userRoleUserId
== userId).all()
        if userRoles:
            roleOutput = []
            for userRole in userRoles:
                roleGivenBy =
db_session.query(rtrk.User).filter(rtrk.User.userId ==
userRole.userRoleRoleCreatedById).first()
                roles =
db_session.query(rtrk.Role).filter(rtrk.Role.roleId ==
userRole.userRoleRoleId)
                for role in roles:
                    roleData = {'roleName': role.roleName,
'roleDescription': role.roleDescription, 'roleGivenDate':
userRole.userRoleDateCreated, 'roleGivenBy':
roleGivenBy.userFirstName + " " + roleGivenBy.userLastName + "(" +
roleGivenBy.userUserName + ")"}
                roleOutput.append(roleData)
        else:
            db_session.close()
            return "User %s doesn't have any roles!"
%(user.userFirstName + " " + user.userLastName)
        userData = {'userFirstName': user.userFirstName,
'userLastName': user.userLastName, 'roles': roleOutput}
        output.append(userData)
    else:
        db_session.close()
        return "Not existing user!"
    db_session.close()
    return{"user": output}
```

P.3.1.4. Implementacija DELETE metode za brisanje korisnika

```
@app.route('/user/<userId>', methods = ['DELETE'])
def deleteUser(userId):
    user = db_session.query(rtrk.User).filter(rtrk.User.userId ==
userId)
    if user:
        try:
            user.delete()
            db_session.commit()
            db_session.close()
            return "0"
        except:
            db_session.close()
```

```
            return "1"
    else:
        return "Not existing user!"
```

P.3.1.5. Implementacija PUT metode za uređivanje korisničke zaporke

```
@app.route('/user/<userId>', methods = ['PUT'])
def updateUser(userId):
    user = db_session.query(rtrk.User).filter(rtrk.User.userId ==
userId).first()
    if user:
        salt = "TTTechAuto"
        userPassword = hashlib.md5((request.json['userPassword'] +
salt).encode()).hexdigest()
        if userPassword == user.userPassword:
            newPassword =
hashlib.md5((request.json['userNewPassword'] +
salt).encode()).hexdigest()
            try:
                user.userPassword = newPassword
                db_session.commit()
                db_session.close
                return "0"
            except:
                db_session.close()
                return "1"
        return "Wrong password!"
    else:
        return "Not existing user!"
```

P.3.1.6. Implementacija metode za ispis svih podataka o dodijeljenim i izvršenim testovima u MS Excel

```
def exportResultsToExcel():
    testLogId = []
    tests = []
    users = []
    hardwares = []
    startTime = []
    endTime = []
    ip = []
    errors = []
    comment = []
    testResults = db_session.query(rtrk.Testslog).all()
    for testResult in testResults:
        user = db_session.query(rtrk.User).filter(rtrk.User.userId ==
testResult.testLogUserId).first()
        userData = str(user.userFirstName) + " " +
str(user.userLastName) + "(" + str(testResult.testLogUserId) + ")"
        test = db_session.query(rtrk.Test).filter(rtrk.Test.testId ==
testResult.testLogTestId).first()
        testData = str(test.testName) + "(" +
str(testResult.testLogTestId) + ")"
```

```
        hardware =
db_session.query(rtrk.Hardware).filter(rtrk.Hardware.hardwareId ==
testResult.testLogHardwareId).first()
        hardwareData = str(hardware.hardwareName) + "(" +
str(testResult.testLogHardwareId) + ")"
        users.append(userData)
        tests.append(testData)
        hardwares.append(hardwareData)
        testLogId.append(testResult.testLogId)
        startTime.append(testResult.testLogStartTime)
        endTime.append(testResult.testLogEndTime)
        ip.append(testResult.testLogIP)
        errors.append(testResult.testLogErrors)
        comment.append(testResult.testLogComment)
    data = pd.DataFrame({"testLogId":testLogId, "test(id)":tests,
"user(id)":users, "hardware(id)":hardwares, "startTime":startTime,
"endTime":endTime, "ip":ip, "errors":errors, "comment":comment})
    data.to_excel('testResults.xlsx', sheet_name='test1',
index=False)
    return "0"
```

P.3.1.7. Implementacija metode za preuzimanje dijagrama rezultata testova

```
@app.route('/testResultsChart', methods = ['GET'])
def getTestResultsChart():
    passedTests = 0
    failedTests = 0
    testResults = db_session.query(rtrk.Testslog).all()
    for testResult in testResults:
        if testResult.testLogErrors:
            failedTests += 1
        else:
            passedTests += 1
    labelPassed = "Passed:" + str(passedTests) + "/" +
str(passedTests + failedTests)
    labelFailed = "Failed:" + str(failedTests) + "/" +
str(passedTests + failedTests)
    s = pd.Series([passedTests, failedTests], name = "Test results")
    fig, ax = plt.subplots()
    s.plot.pie(figsize = (6,5), labels = [labelPassed, labelFailed],
autopct = '%1.2f%%')
    downloadsPath = str(Path.home() / "Downloads")
    fig.savefig(downloadsPath + "/testResultsPieChart.png")
    plt.cla()   #clears axis
    plt.clf()   #clears figure
    plt.close('all')
    return send_file(downloadsPath + "/testResultsPieChart.png",
mimetype='image/png')
```