Aplikacija za nadgledanje i upravljanje računa u banci

Haubrich, Kristijan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

APLIKACIJA ZA NADGLEDANJE I UPRAVLJANJE RAČUNA U BANCI

Diplomski rad

Kristijan Haubrich

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Kristijan Haubrich
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D-1204R, 07.10.2021.
JMBAG:	0165074858
Mentor:	doc. dr. sc. Tomislav Galba
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	doc. dr. sc. Tomislav Galba
Član Povjerenstva 2:	izv. prof. dr. sc. Alfonzo Baumgartner
Naslov diplomskog rada:	Aplikacija za nadgledanje i upravljanje računa u banci
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Potrebno je napraviti aplikaciju koja će omogućiti nadgledanje i upravljanje računima u banci. Postojale bi dvije uloge (upravitelj i korisnik). Upravljanje i nadgledanje korisničkih računa, bankovnih računa i transakcija te brisanje i dodavanje istih su mogućnosti upravitelja dok korisnik može odraditi transakciju i pogledati stanje svojih računa i transakcija. Aplikacija bi trebala biti smještena unutar Docker kontejnera dok bi API bio izrađen koristeći SpringBoot programski okvir. Korisnički dio aplikacije trebao bi biti napravljen koristeći ReactNative programski okvir.
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	25.06.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	5.7.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	11.07.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 11.07.2024.

Ime i prezime Pristupnika:	Kristijan Haubrich
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1204R, 07.10.2021.
Turnitin podudaranje [%]:	11

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za nadgledanje i upravljanje računa u banci**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Galba

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD 1

1.1. Zadatak diplomskog rada 1

2. PREGLED SLIČNIH RJEŠENJA 3

2.1. Erste mBanking 3

2.2. Addiko Mobile Hrvatska 4

2.3. PBZ mobilno bankarstvo 4

2.4. m-zaba 5

2.5. mojaRBA – moje financije 6

3. KORIŠTENE TEHNOLOGIJE 8

3.1. Docker 8

3.2. Liquibase 9

3.3. Spring Boot 10

3.4. JWT Token 11

3.5. React Native 12

3.6. VSC – Visual Studio Code 13

3.7. Intelij 14

3.8. Ostalo 15

4. POSTUPAK IZRADE APLIKACIJE ZA NADGLEDANJE I UPRAVLJANJE RAČUNA U BANCI 18

4.1. Struktura datoteka 18

4.2. Sigurnost 20

4.2.1. Uloge i dopuštenja 20

4.2.2. Tokeni 21

4.2.3. Konfiguracija sigurnosti 23

4.3. Registracija i prijava 23

4.3.1. *Users* 24

4.3.2. *BankManager* 25

4.3.3. *Client* 25

4.3.4. Postupak registracije 26

4.3.5. Postupak prijave 27

4.4. Bankovni menadžer 29

4.4.1. *Account* 29

4.4.2. *CreditCard* 31
4.4.3. Zaslon profila 31
4.4.4. Zaslon postavki 32
4.4.5. Zaslon profila klijenta 32
4.4.6. Zaslon za pravljenje novog bankovnog računa 32
4.4.7. Zaslon bankovnog računa 33
4.4.8. Zaslon za dodavanje kreditne kartice 34
4.5. Klijent 34
4.6. Transakcije 35
4.6.1. *Transaction* 35
4.6.2. Uplata 36
4.6.3. Isplata 37
4.6.4. Isplata na drugi račun 38
5. PRIKAZ APLIKACIJE ZA NADGLEDANJE I UPRAVLJANJE RAČUNA U BANCI 39
6. ZAKLJUČAK 53
LITERATURA 54
SAŽETAK 56
ABSTRACT 57
ŽIVOTOPIS 58

1. UVOD

Tema ovog diplomskog rada jest izrada aplikacije za nadgledanje računa u banci. Zadatak aplikacije je omogućiti korisniku registraciju i prijavu u aplikaciju preko JWT (eng. *JSON Web Token*) tokena te nakon toga korištenje aplikacije za provjeru stanja računa i dodavanje transakcija. Osim korisnika koji bi imali svoje bankovne račune postoje i korisnici koji nadgledaju bankovne račune (bankovni menadžeri). Oni imaju popis svojih klijenata. Bankovni menadžeri mogu pristupiti listi svojih klijenata, dodavati im nove bankovne račune, dodavati kreditnu karticu ili raditi transakcije na bilo kojem njihovom bankovnom računu te obrisati profil klijenta u aplikaciji.

Aplikacija je namijenjena svim bankama kao demo sustav koji bi mogli koristiti njihovi klijenti kao i zaposlenici u svrhu osnovnog pregleda računa. Izvedba aplikacije može se podijeliti na izradu baze podataka i pristupnih točaka (eng. *endpoints*) preko programskog sučelja aplikacije (eng. *API - Application Programming Interface*) i izradu aplikacije koja služi za interakciju s korisnicima i koristi navedeno programsko sučelje. Docker spremnik (eng. *container*) koristi se za spremanje baze podataka. Za migraciju podataka iz modela Java klase u tablicu baze podataka koristi se *Liquibase*. API je izrađen pomoću *Spring Boot* Java programskog okvira. Aplikacija za interakciju s korisnicima napravljena je u *ReactNative* programskom okviru gdje je *kôd* pisan u *JavaScript* programskom jeziku. Struktura diplomskog rada sastoji se od pregleda sličnih rješenja, teorijske podloge, postupka izrade aplikacije, prikaz izgleda aplikacije te zaključak. U pregledu sličnih rješenja opisano je pet rješenja za kupnju i prodaju koja imaju sličnosti s aplikacijom koja se izrađuje. Teorijska podloga je kratki opis tehnologija korištenih u izradi aplikacije. Izrada aplikacije predočava i objašnjava funkcionalnosti aplikacije te kako su one izvedene. U prikazu izgleda aplikacije prikazuje se izgled aplikacije na uređaju.

U trećem poglavlju opisane su korištene tehnologije. Četvrto poglavlje opisuje način rada i postupak izrade aplikacije. Prikaz aplikacije i izgled svih opisanih elemenata prikazuje peto poglavlje. U šestom poglavlju nalazi se zaključak o aplikaciji i cijelom projektu.

1.1. Zadatak diplomskog rada

Kratko opisati način rada aplikacije. Izraditi aplikaciju koja treba omogućiti korisnicima prijavu i registraciju kao bankovni menadžer ili kao klijent. Bankovnom menadžeru omogućuje se pristup klijentima koji su kod njega stvorili račun. On može klijentu dodati ili obrisati novi

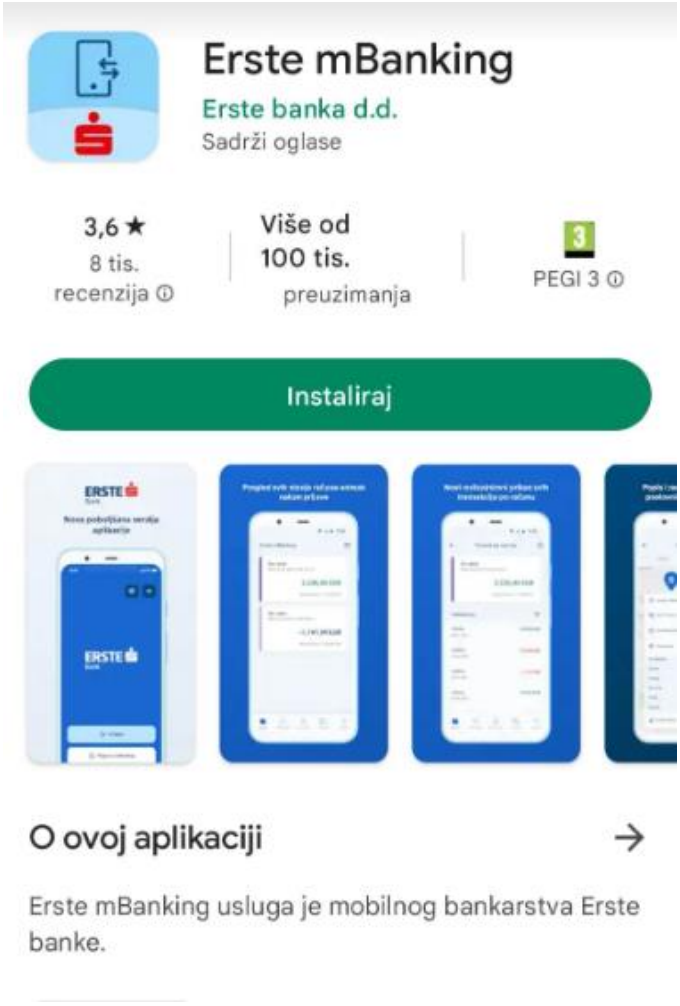
bankovni račun. Na bankovnom računu klijenta menadžer može dodati transakcije kao i kreditnu karticu u slučaju zahtjeva klijenta. Klijentu je omogućen pregled svih njegovih bankovnih računa kao i transakcija. Klijent ne može obrisati ni dodati sebi bankovni račun ali može izvršavati transakcije. Svaki korisnik aplikacije može sebi naknadno promijeniti lozinku. Brisanje profila od bankovnog menadžera ili klijenta u mogućnosti je samo bankovnom menadžeru.

2. PREGLED SLIČNIH RJEŠENJA

Dolaskom novih tehnologija razvila su se i rješenja koja ljudima štede novac i vrijeme. S obzirom na to, danas postoji veliki broj rješenja za Internet bankarstvo, a neka od njih će se opisati u sklopu ovog poglavlja.

2.1. Erste mBanking

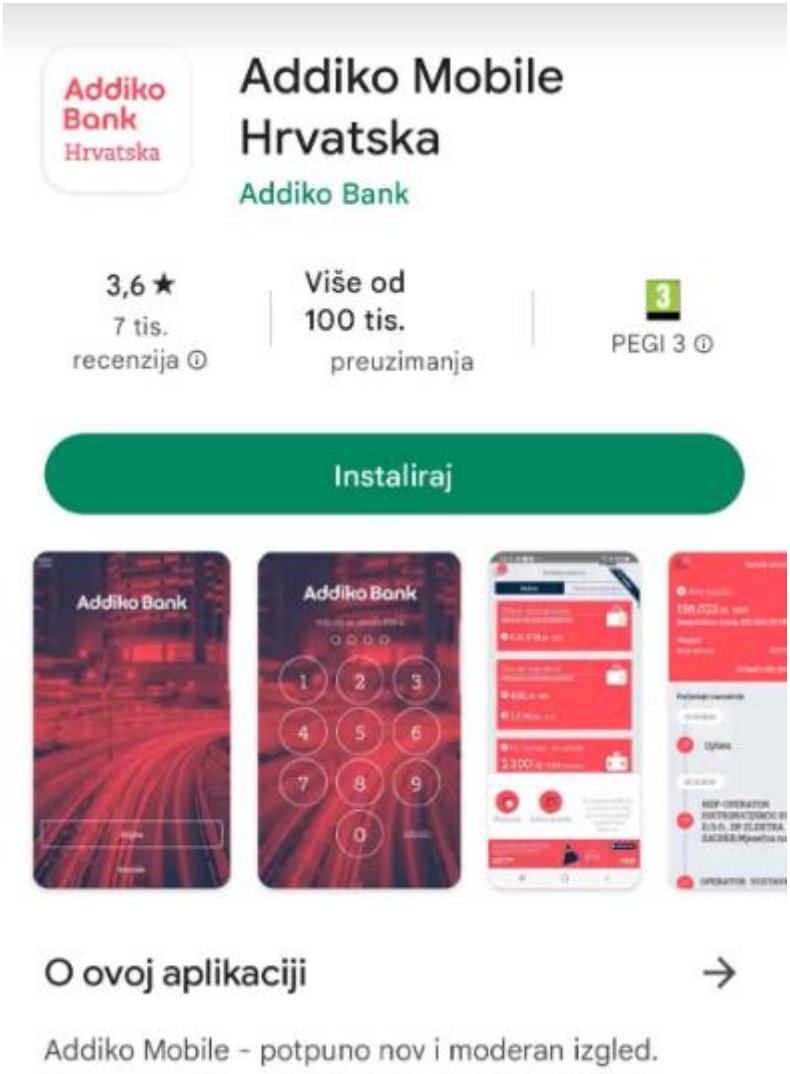
Erste mBanking (**Slika 2.1.**) je mobilna aplikacija za mobilno bankarstvo Erste banke. Ima funkcionalnosti poput pregledavanja stanja po računima, kontroliranja prometa i zadane transakcije, pregledavanja i naručivanja izvoda s mjesta i u vrijeme koje odgovara korisniku, a pritom i uštedjeti zahvaljujući nižim naknadama.



Slika 2.1. Prikaz rješenja Erste mBanking

2.2. Addiko Mobile Hrvatska

Addiko Mobile Hrvatska (**Slika 2.2.**) je mobilna aplikacija za mobilno bankarstvo Addiko Banke za klijente u Hrvatskoj. Neke od funkcionalnosti aplikacije su otključavanje aplikacije i potvrđivanje plaćanja uslugom biometrije, otiskom prsta ili skeniranjem lica, dijeljenje IBAN-a preko 2D bar koda, izvršavanje i pregled transakcija i slično.

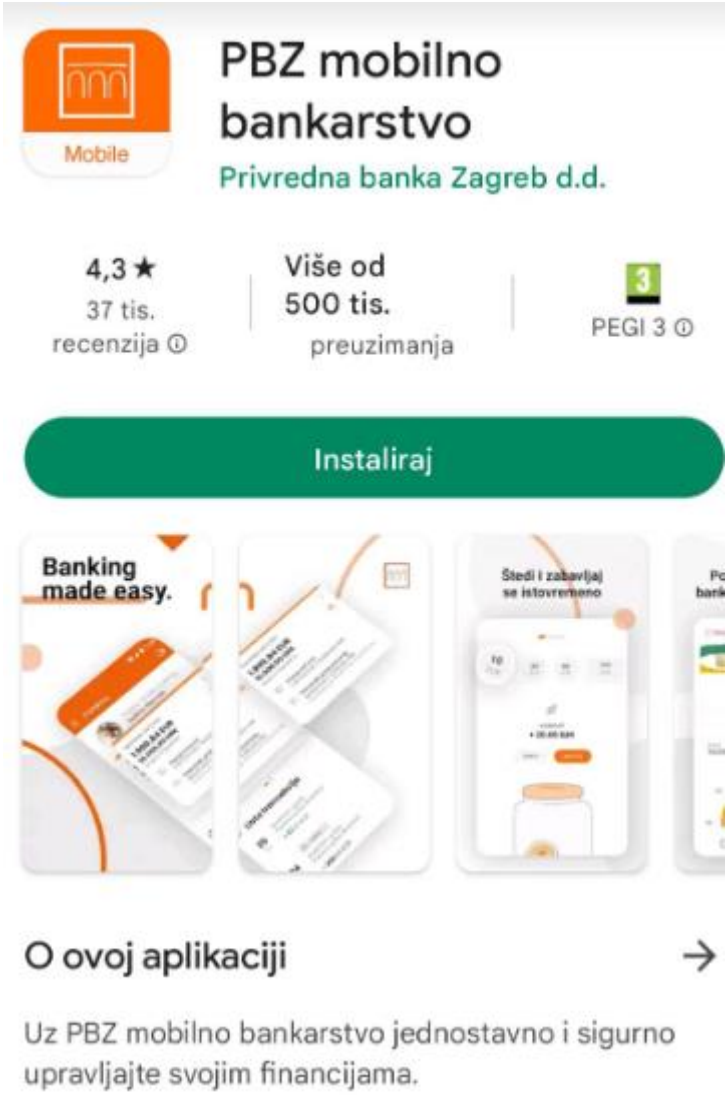


Slika 2.2. Pregled rješenja Addiko Mobile Hrvatska na Trg Play-u

2.3. PBZ mobilno bankarstvo

PBZ mobilno bankarstvo (**Slika 2.3.**) je mobilna aplikacija za upravljanje i pregled financija klijenta PBZ banke u Hrvatskoj. Neke od funkcionalnosti su brzo i jednostavno plaćanje

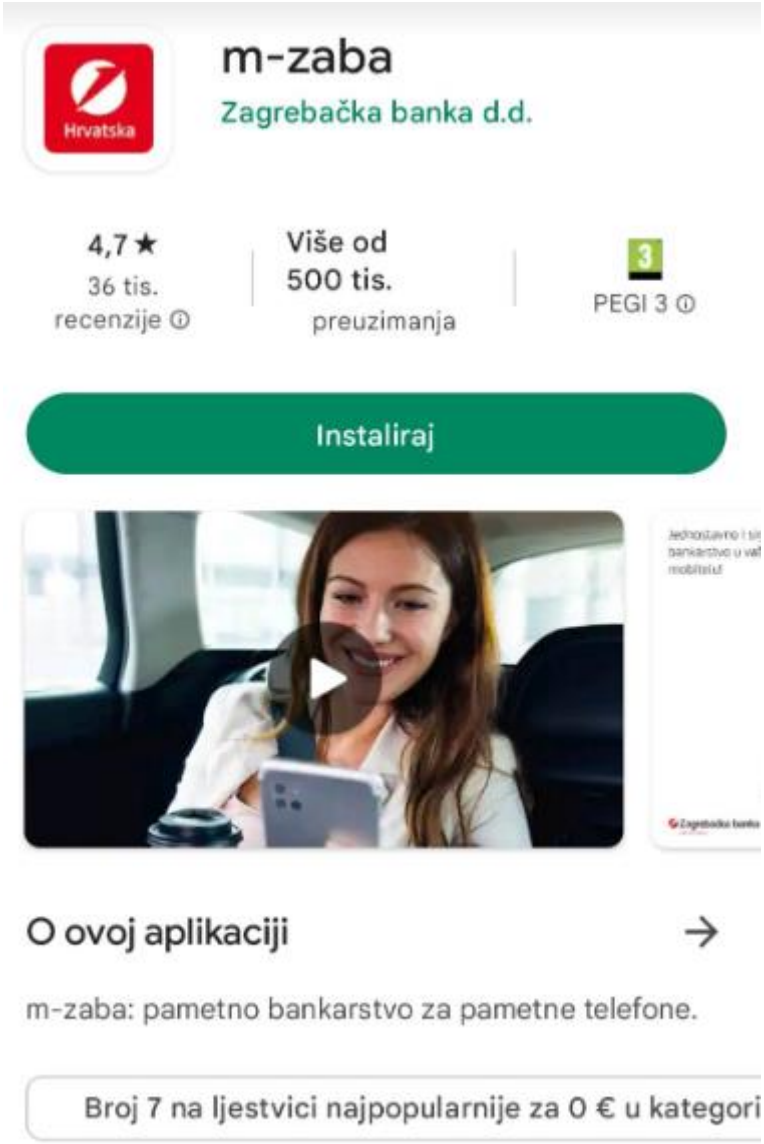
kontaktima s mobilnog telefona, podizanje gotovine na PBZ bankomatu bez kartice pomoću generiranog koda, pristupanjem portalu *e-Građani* pomoću tokena, uvid i upravljanje transakcijama i slično.



Slika 2.3. Pregled rješenja PBZ mobilno bankarstvo na Trg Play-u

2.4. m-zaba

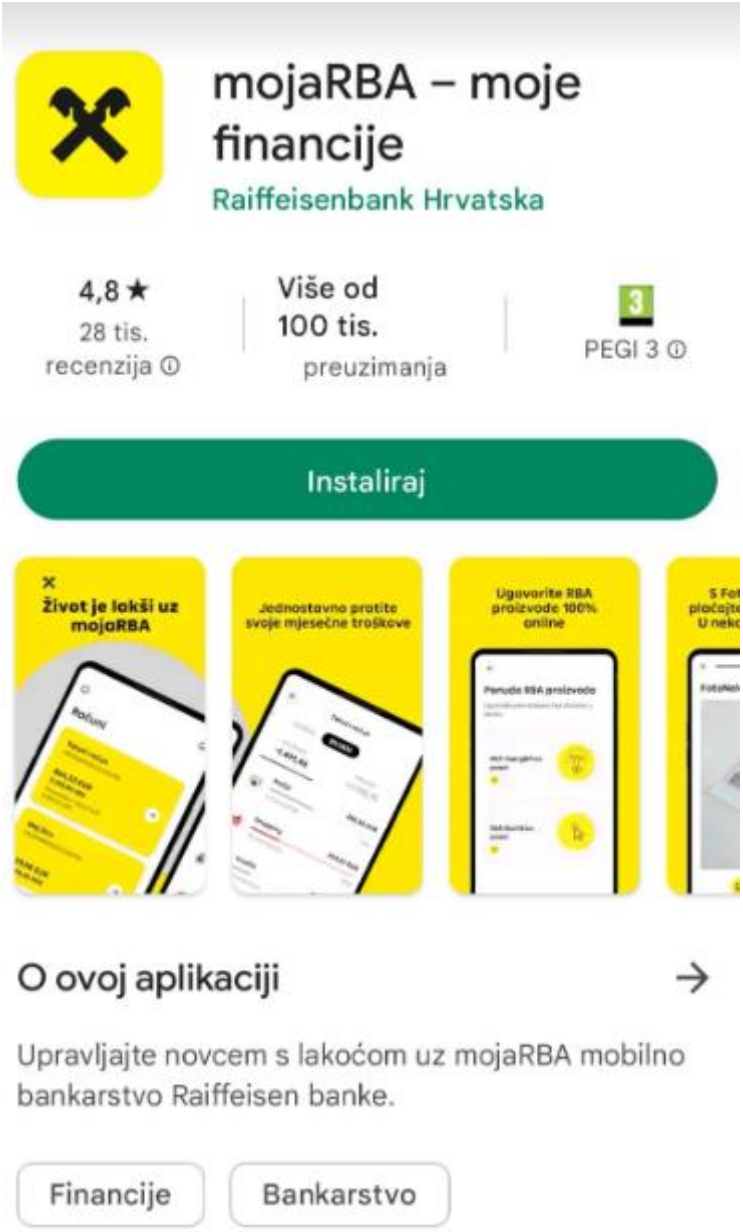
m-zaba (**Slika 2.4.**) je aplikacija koja pruža usluge upravljanja i pristupa financijama klijentima Zagrebačke banke. Neke od funkcionalnosti su plaćanja računa preko skeniranja bar koda, uvid u stanje računa i transakcije, ugovaranje kredita bez odlaska u poslovnicu, beskontaktno plaćanje pametnim telefonom i slično.



Slika 2.4. Pregled rješenja m-zaba na Trg Play-u

2.5. mojaRBA – moje financije

mojaRBA – moje financije (**Slika 2.5.**) je mobilna aplikacija za upravljanje i pregled financija klijenta. Neke od funkcionalnosti su personalizirani pregled računa, skrivanje određenih računa s liste transakcija, plaćanje računa preko skeniranja bar koda, izvršavanje i uvid u transakcije, praćenje i stvaranje štednje i slično.



Slika 2.5. Pregled rješenja mojaRBA – moje financije na Trg Play-u

Za kraj je bitno reći da sva navedena rješenja imaju funkcionalnosti koje dijele s aplikacijom ovog diplomskog rada kao što su izvršavanje i pregled transakcija kao i uvid u cijeli bankovni račun. Rješenja također sadrže i puno funkcionalnosti (plaćanje računa skeniranjem bar koda, štednja, povlačenje sredstava iz bankomata uz pomoć tokena i slično) koje bi se mogle dodati u aplikaciju ovog diplomskog rada u slučaju daljnjeg razvoja.

3. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju objašnjene su sve tehnologije koje će biti korištene u izradi aplikacije. Pored opisa tehnologija objašnjeni su i razlozi korištenja u aplikaciji te njihova uloga.

3.1. Docker

Docker (**Slika 3.1.**) je softverska platforma koja omogućuje brzu izradu, testiranje i implementaciju aplikacija. *Docker* pakira softver u standardizirane jedinice koje se nazivaju spremnici. Spremnici imaju sve što je softveru potrebno za pokretanje uključujući biblioteke, sistemske alate, *kôd* i vrijeme izvođenja. Pomoću *Docker-a* može se brzo implementirati i skalirati aplikacije u bilo koje okruženje i znati da će se *kôd* pokrenuti. [1]

Docker radi tako da pruža standardni način za pokretanje koda. *Docker* je operativni sustav za spremnike. Slično kao što virtualni stroj uklanja potrebu za izravnim upravljanjem hardverom poslužitelja, kontejneri uklanjaju potrebu za izravnim upravljanjem operativnog sustava poslužitelja. *Docker* je instaliran na svakom poslužitelju i pruža jednostavne naredbe koje se mogu koristiti za izgradnju, pokretanje ili zaustavljanje spremnika [1]

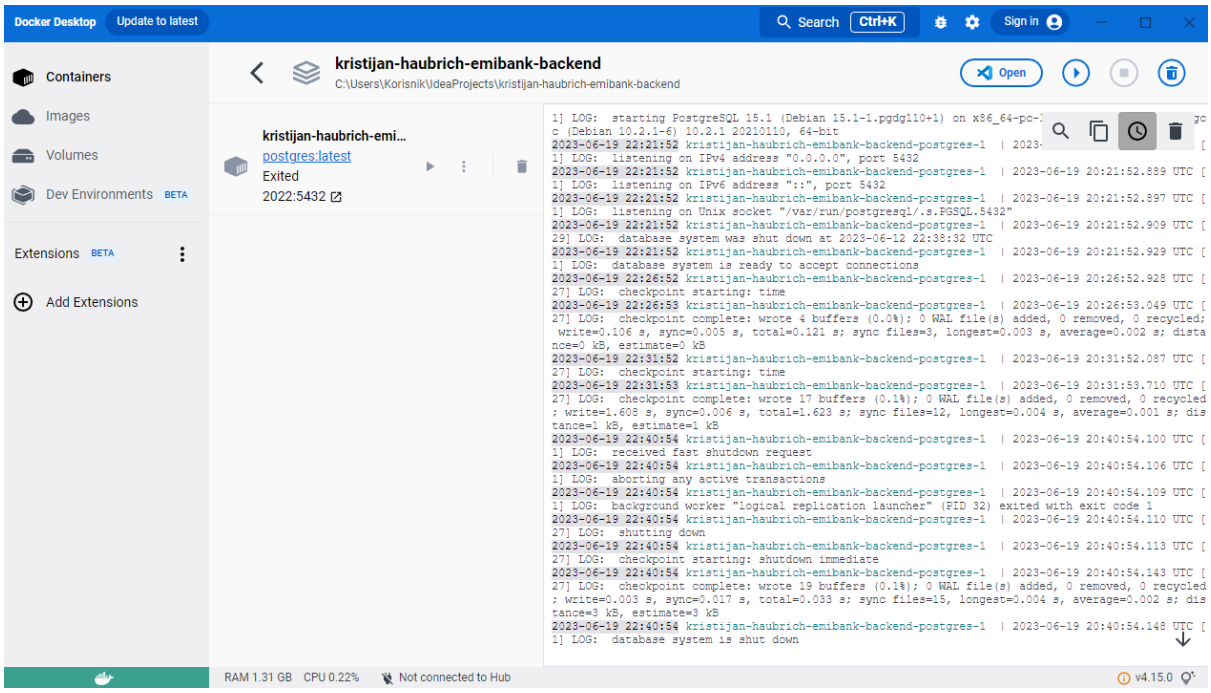
Docker je moćan alat i mnoge su mogućnosti rada s njim:

Brza, dosljedna isporuka aplikacija - *Docker* usmjerava životni ciklus razvoja dopuštajući programerima da rade u standardiziranim okruženjima koristeći lokalne spremnike koji pružaju aplikacije i usluge. Spremnici su izvrsni za kontinuiranu integraciju i radne tijekove kontinuirane isporuke (eng. *CI/CD – (continuous integration and continuous deployment)*).[2]

Responzivna implementacija i skaliranje – *Docker*-ova platforma temeljena na kontejnerima omogućuje visoko prenosiva radna opterećenja. *Docker* spremnici mogu se izvoditi na lokalnom prijenosnom računalu programera, na fizičkim ili virtualnim strojevima u podatkovnom centru, na pružateljima usluga u oblaku ili u mješavini okruženja. *Docker*-ova prenosivost i lagana priroda također olakšavaju dinamičko upravljanje radnim opterećenjima, skaliranjem ili rušenjem aplikacija i usluga prema poslovnim potrebama, u gotovo stvarnom vremenu. [2]

Pokretanje više radnih opterećenja na istom hardveru - *Docker* je lagan i brz. Pruža održivu i isplativu alternativu virtualnim strojevima koji se temelje na hipervizoru, tako da se može koristiti više kapaciteta poslužitelja za postizanje poslovnih ciljeva. *Docker* je savršen za

okruženja visoke gustoće i za male i srednje implementacije gdje treba učiniti više s manje resursa.[2]



Slika 3.1. Izgled Docker korisničkog sučelja za prikaz spremnika baze korištene u radu

Docker spremnik je u ovom radu korišten za spremanje baze podataka. Baza podataka je tipa *PostgreSql* i u nju se spremaju svi podaci o aplikaciji. Da bi kreirali spremnik s bazom napravljena je datoteka *docker-compose.yaml* koja sadrži sve informacije o spremniku. Jednostavnom direktivom *docker compose up* unutar terminala spremnik na *Dockeru* se stvara ili ako već postoji pokrene se i konfigurira za početak rada. Nakon pokretanja spremnika baza je spremna za primanje i slanje podataka.

3.2. Liquibase

Liquibase je rješenje za upravljanje promjenama sheme baze podataka otvorenog koda koje omogućuje jednostavno upravljanje revizijama promjena baze podataka. *Liquibase* svima koji su uključeni u proces izdavanja aplikacije olakšava:

1. Uklanjanje pogrešaka i kašnjenja prilikom izdavanja baza podataka.
2. Implementiranje i vraćanje promjena za određene verzije bez potrebe o znanju za onim što je već implementirano

3. Implementiranje promjene baze podataka i aplikacije zajedno kako bi uvijek bile sinkronizirane[3]

Ručne promjene baze podataka usporavaju sposobnost isporuke i sposobnost tima da surađuje. Svaki ručni pregled i provjera svake promjene baze podataka u svakoj skripti košta organizaciju dragocjenog vremena i novca. U međuvremenu, loše promjene sheme i dalje se provlače do proizvodnje—ugrožavajući rad aplikacije i izlažući podatke riziku od krađe. *Liquibase* je stvoren kako bi riješio važan, ali vremenski zahtjevan zadatak ažuriranja sheme baze podataka. Automatizira proces i čini bazu podataka sigurnijom, lakšom za reviziju i usklađenom.[4]

Liquibase je u ovom radu korišten za promjene baze koje se naprave tijekom definiranja entiteta baze i početnih uvjeta za bazu u *SpringBoot*-u. *Kôd* je pisan u XML-u. Promjena u bazi može biti promjena tablice, dodavanje primarnih, stranih ili jedinstvenih ključeva ili pak čisto dodavanje podataka u tablice prije korištenja aplikacije (npr. uvjetovanje pristupa korisnicima određenim krajnjim točkama preko uloga i dozvola (eng. *roles and permissions*). Za svaku novu promjenu kreirana je nova XML datoteka i imenovana je na sljedeći način *v1.x.x – naziv_promjene.xml* gdje se umjesto x stavlja verzija koja je u skladu s dosadašnjim dodavanjima u bazu (npr. v1.0.0. , v1.0.1. ,…). Pozivom direktive *includeAll* pozvat će se po redu sve promjene i ako postoji neka nova promjena izvršit će se, a u suprotnom promjene će se samo provjeriti jesu li u skladu s dosada izvršenim.

3.3. Spring Boot

Java Spring Framework (Spring Framework) popularan je okvir otvorenog koda na razini poduzeća za stvaranje samostalnih aplikacija proizvodne razine koje se izvode na Java Virtual Machine (JVM). Java Spring Boot (Spring Boot) je alat koji čini razvoj web aplikacija i mikroservisa uz Spring Framework bržim i lakšim kroz tri osnovne mogućnosti: automatska konfiguracija, samostalan pristup konfiguraciji i mogućnost izrade samostalnih aplikacija. Ove značajke rade zajedno kako bi pružile alat koji omogućuje postavljanje aplikacije temeljene na *Springu* uz minimalnu konfiguraciju i postavljanje. Auto konfiguracija znači da se aplikacije inicijaliziraju s unaprijed postavljenim ovisnostima koje se ne moraju ručno konfigurirati. Budući da Java Spring Boot dolazi s ugrađenim mogućnostima automatske konfiguracije, automatski konfigurira i temeljni Spring Framework i pakete trećih strana na temelju korisničkih postavki (i na temelju najboljih praksi, što pomaže u izbjegavanju pogrešaka). Iako se mogu poništiti ove zadane postavke nakon dovršetka inicijalizacije, značajka automatske

konfiguracije Java Spring Boot-a omogućuje brz početak razvijanja aplikacije temeljene na Springu i smanjuje mogućnost ljudskih pogrešaka.[5]

Spring Boot koristi samouvjeren pristup dodavanju i konfiguriranju početnih ovisnosti, na temelju potreba projekta. Slijedeći vlastitu prosudbu, Spring Boot odabire koje će pakete instalirati i koje će zadane vrijednosti koristiti, umjesto da od korisnika traži da sam donosi sve te odluke i sve postavlja ručno. Potrebe projekta mogu se definirati tijekom procesa inicijalizacije, tijekom kojeg se bira između više početnih ovisnosti— nazvanih Spring Starters—koje pokrivaju tipične slučajeve upotrebe. Spring Boot Initializr pokreće se ispunjavanjem jednostavnog web obrasca, bez ikakvog kodiranja. Spring Boot uključuje više od 50 Spring Startera, a dostupno je još puno drugih pokretača.[5]

Spring Boot pomaže programerima u stvaranju aplikacija koje se jednostavno pokreću. Točnije, omogućuje stvaranje samostalnih aplikacija koje se pokreću same, bez oslanjanja na vanjski web poslužitelj, ugradnjom web poslužitelja kao što je Tomcat ili Netty u aplikaciju tijekom procesa inicijalizacije. Kao rezultat toga, može se pokrenuti aplikacija na bilo kojoj platformi jednostavnim pritiskom na naredbu *Pokreni*. Može se isključiti ovu značajku za izradu aplikacija bez ugrađenog web poslužitelja.[5]

SpringBoot se koristio u ovom radu za pisanje API-a (eng. Aplication Programming Interface) aplikacije. U to spada definiranje svih pristupnih točki kojima može neki korisnik pristupati, definiranje sučelja preko kojeg se pristupa određenom entitetu u bazi (korišten je *JPA repository*), pisanje svih entiteta u bazi i njihovih atributa, pisanje servisa za svaki kontroler u kojem se implementira posao prilikom pristupa nekoj pristupnoj točki , definiranje klasa koje preslikavaju druge klase (eng. *mapper*) pomoću kojih se najčešće preslikavalo entitete u objekte za prijenos podataka (eng. *DTO - Data Transfer Object*) te definiranje sigurnosti pristupa API-u i implementacije provjere pristupa pristupnim točkama preko JWT tokena.

3.4. JWT Token

JSON Web Token (JWT) je otvoreni standard (RFC 7519) koji definira kompaktan i samostalan način za siguran prijenos informacija između strana kao JSON objekt. Ovi se podaci mogu provjeriti i vjerovati jer su digitalno potpisani. JWT-ovi se mogu potpisati korištenjem tajne (s HMAC algoritmom) ili para javnih/privatnih ključeva pomoću RSA ili ECDSA. Potpisani tokeni mogu potvrditi integritet zahtjeva sadržanih u njemu, dok šifrirani tokeni skrivaju te zahtjeve od drugih strana. Kada se tokeni potpisuju korištenjem parova

javnih/privatnih ključeva, potpis također potvrđuje da je samo strana koja drži privatni ključ ona koja ga je potpisala. Token se može koristiti za autorizaciju ili za razmjenu informacija.[6]

Autorizacija: Ovo je najčešći scenarij za korištenje JWT-a. Nakon što se korisnik prijavi, svaki sljedeći zahtjev uključit će JWT, dopuštajući korisniku pristup rutama, uslugama i resursima koji su dopušteni s tim tokenom. *Single Sign On* je značajka koja danas naširoko koristi JWT, zbog svojih malih troškova i mogućnosti da se lako koristi na različitim domenama.[6]

Razmjena informacija: JSON web tokeni dobar su način sigurnog prijenosa informacija između strana. Budući da se JWT-ovi mogu potpisati—na primjer, korištenjem parova javnih/privatnih ključeva—sigurno je da su pošiljatelji oni za koje se predstavljaju. Osim toga, budući da se potpis izračunava korištenjem zaglavlja i nosivosti, također može se provjeriti da sadržaj nije bio neovlašteno mijenjan.[6]

JWT token je korišten u radu za autorizaciju korisnika. Jednom kada se korisnik prijavi dobije svoj pristupni token (eng. *access token*) i svoj obnavljajući token (eng. refresh token). Trajanje pristupnog tokena je svega 3 sata dok je trajanje obnavljajućeg tokena 20 dana. Svaki zahtjev prema bazi se presretne i provjerava se rok tokena. Ako je tokenu istekao rok onda se pomoću obnavljajućeg token dohvati novi pristupni token koji može koristiti korisnik. Kada istekne i obnavljajući token korisnika se preusmjerava na ponovnu prijavu.

3.5. React Native

React Native je JavaScript okvir za pisanje stvarnih, „nativno“ renderiranih mobilnih aplikacija za iOS i Android. Temelji se na React-u, Facebook-ovoj JavaScript biblioteci za izradu korisničkih sučelja, ali umjesto da cilja na preglednik, cilja na mobilne platforme. Drugim riječima: web programeri sada mogu pisati mobilne aplikacije koje izgledaju i osjećaju se doista „nativno“, sve iz udobnosti JavaScript biblioteke koja je već dobro poznata programerima. Osim toga, budući da se većina koda koji se napiše može dijeliti između platformi, React Native olakšava istovremeni razvoj za Android i iOS.[7]

React Native može se koristiti već u postojećim Android i iOS projektima ili može se stvoriti potpuno novu aplikaciju od nule. Primitive React iscrtavaju se prema korisničkom sučelju izvorne platforme, što znači da aplikacija koristi iste API-je izvorne platforme koje koriste druge aplikacije. React Native omogućuje stvaranje istinski nativnih aplikacija i ne ugrožava iskustva korisnika. Pruža temeljni skup izvornih komponenti neovisnih o platformi kao što su

Prikaz, Tekst i Slika koje se preslikavaju izravno na izvorne građevne blokove korisničkog sučelja platforme. Komponente Reacta omotavaju postojeći izvorni *kôd* i komuniciraju s izvornim API-ima putem React-ove deklarativne UI (eng. *User Interface* – korisničko sučelje) paradigme i JavaScripta. To omogućuje nativni razvoj aplikacija za potpuno nove timove programera i može omogućiti postojećim nativnim timovima da rade mnogo brže. Omogućava pregled promjena odmah nakon spremanja. Uz snagu JavaScripta, React Native omogućuje ponavljanje velikom brzinom. Nema čekanja da završe izvorne verzije. Spremi, vidi, ponovi.[8]

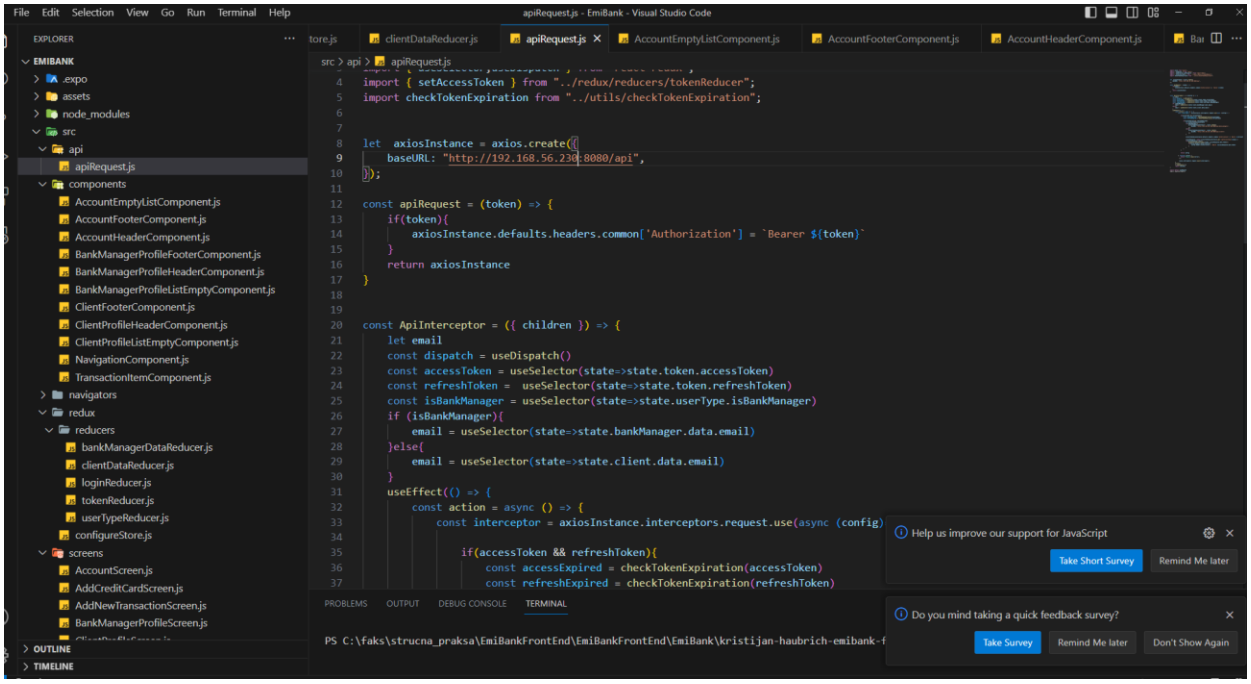
React Native je korišten u ovom radu za stvaranje aplikacije koja komunicira s korisnikom. Aplikacija prikazuje sve zaslone koje korisnik vidi. Dohvaća sve podatke iz baze podataka i važne podatke o korisniku nakon prijave sprema na lokalnu bazu podataka. Za manipulaciju lokalnom bazom podataka koristi se *Redux Toolkit.* Stvaranje zaslona se piše preko *JSX* (*funny tag syntax*). Također, aplikacija presreće sve zahtjeve prema serveru gdje provjerava korisnikove tokene i dopuštenja za pristup podacima na bazi. Pregled aplikacije omogućen je preko *Expo GO* mobilne aplikacije.

3.6. VSC – Visual Studio Code

Visual Studio Code je lagan, ali moćan uređivač izvornog koda koji radi na radnoj površini i dostupan je za Windows, macOS i Linux. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js i ima bogat ekosustav proširenja za druge jezike i okruženja (kao što su C++, C#, Java, Python, PHP, Go, .NET).[9]

Visual Studio Code sadrži IntelliSense dovršavanje koda za varijable, metode i uvezene module, grafičko otklanjanje pogrešaka, uređivanje s više kursora, savjeti za parametre i druge moćne značajke za uređivanje.Također, ima zgodnu navigacija koda i refaktoriranje te ugrađenu kontrolu izvornog koda uključujući Git podršku. Velik dio toga prilagođen je tehnologiji Visual Studio. Visual Studio Code izgrađen je pomoću Electron shell-a, Node.js, TypeScript i Language Server Protocol te se ažurira na mjesečnoj bazi. Brojna proširenja ažuriraju se onoliko često koliko je potrebno. Bogatstvo podrške razlikuje se od različitih programskih jezika i njihovih proširenja, u rasponu od jednostavnog isticanja sintakse i podudaranja zagrada do otklanjanja pogrešaka i refaktoriranja. Korisnik može dodati osnovnu podršku za svoj omiljeni jezik putem TextMate uređivača ako nijedan jezični poslužitelj nije dostupan.[10]

VSC je korišten u radu pri programiranju aplikacije za korisničko sučelje. S lijeve strane je prikazana struktura datoteka aplikacije a s desne prozor gdje se programira. Na dnu se može vidjeti prozor za terminal gdje se upisuju direktive za pokretanje aplikacije ili dodavanje nekih biblioteka (***Slika 3.2.***).



Slika 3.2. Izgled VSC korisničkog sučelja za prikaz koda korištenog u radu

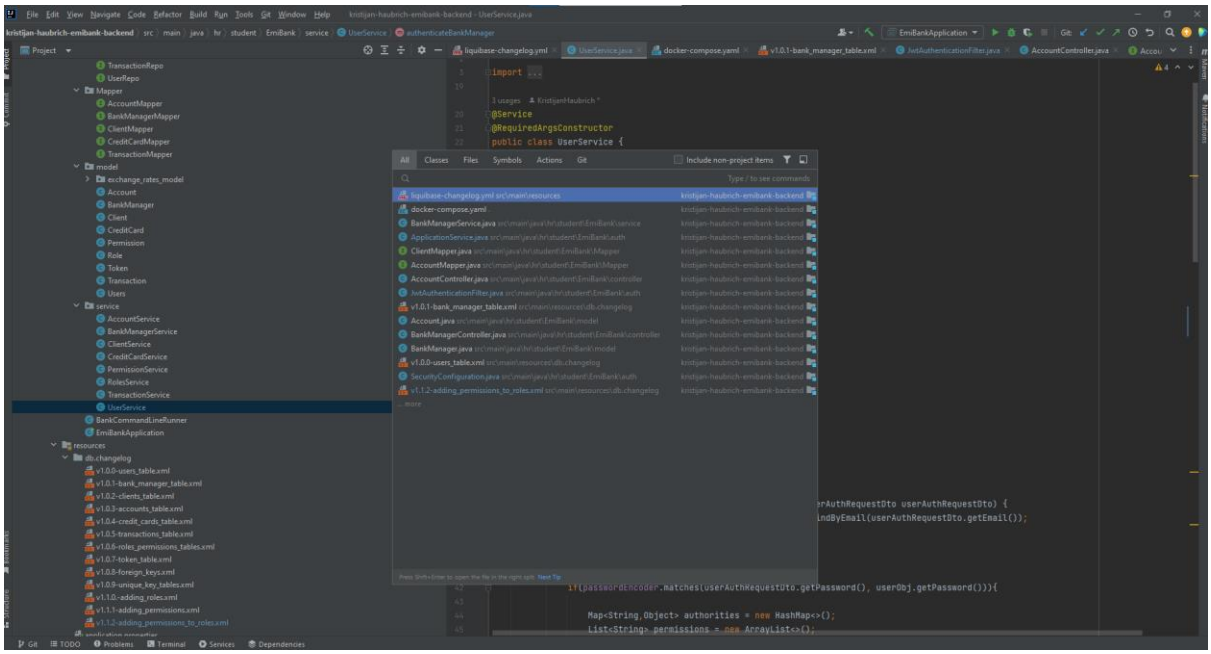
3.7. Intelij

IntelliJ IDEA je integrirano razvojno okruženje za JVM (eng. *Java Virtual Machine*) jezike osmišljeno za povećanje produktivnosti programera. Obavlja rutinske i ponavljajuće zadatke za razvojnog programera pružajući pametno dovršavanje koda, statičku analizu koda i refaktoriranje te omogućuje razvojnom programeru da se usredotoči na svijetlu stranu razvoja softvera, čineći ga ne samo produktivnim nego i ugodnim iskustvom.[11]

IntelliJ IDEA ima neke vrhunske produktivne značajke dovršavanja Java koda. Njegov algoritam može točno pretpostaviti što programer pokušava upisati i dovršava to umjesto njega, čak i ako on ne zna točan naziv određene klase, člana ili bilo kojeg drugog resursa. IntelliJ IDEA stvarno razumije i ima dubok uvid u kod, kao i kontekst programera, što ga čini toliko jedinstvenim među ostalim Java IDE-ima. Podržava dovršavanje koda temeljeno na kontekstu. Daje popis najrelevantnijih simbola primjenjivih u trenutnom kontekstu. Lančano dovršavanje koda − to je napredna značajka dovršavanja koda koja navodi primjenjive simbole kojima se

može pristupiti putem metoda u trenutnom kontekstu. Dovršavanje statičkog člana − omogućuje korištenje statičkih metoda ili konstanti i automatski dodaje potrebne izjave za uvoz kako bi se izbjegla pogreška kompilacije. Otkrivanje duplikata − u hodu pronalazi duple fragmente *kôda* i daje obavijest/prijedlog korisniku o tome. Kad god IntelliJ otkrije da se programer sprema pogriješiti, mala obavijest u obliku žaruljice pojavljuje se u istom retku. Klikom na žaruljicu prikazuje se popis prijedloga.[12]

U ovom radu InteliJ se koristi za programiranje API-a aplikacije. Na ***Slici 3.3.*** može se vidjeti izgled korisničkog sučelja InteliJ-a korištenog za izradu aplikacije. Lijevo se može vidjeti struktura aplikacije, a desno prozor za pisanje koda. Također u sredini se može vidjeti i dijalog za pretraživanje direktorija aplikacije.

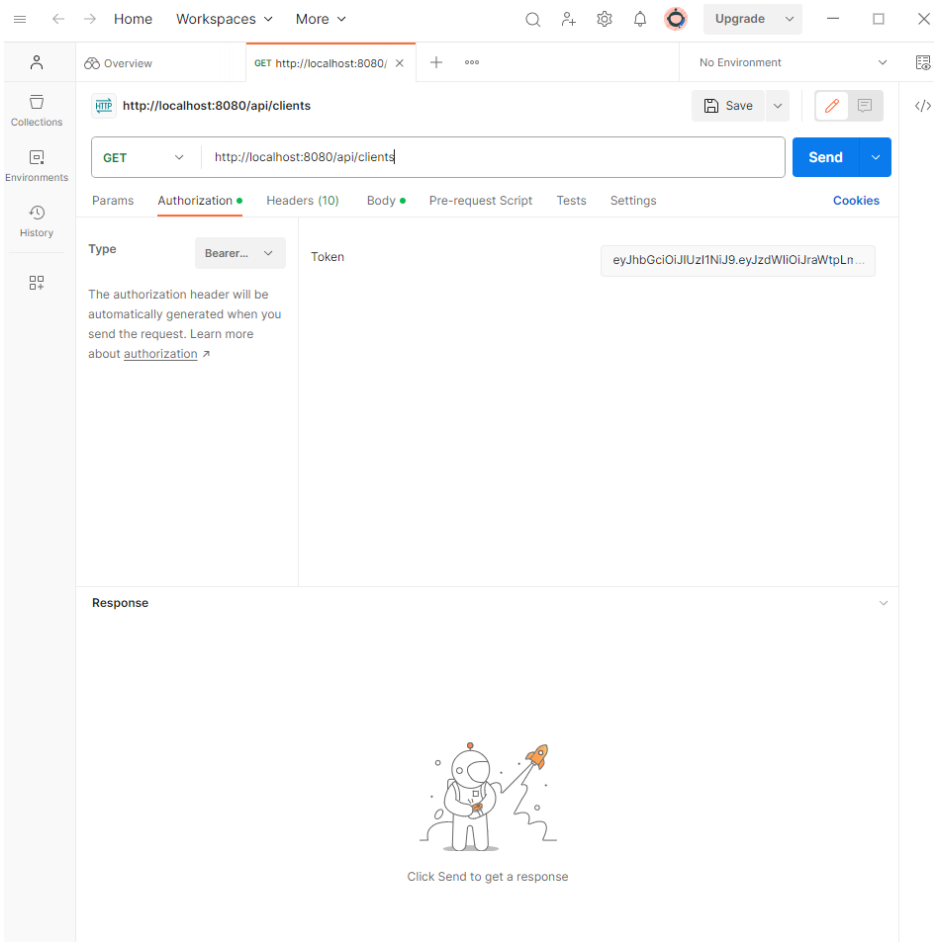


Slika 3.3. Izgled Intelij korisničkog sučelja za prikaz koda korištenog u radu

3.8. Ostalo

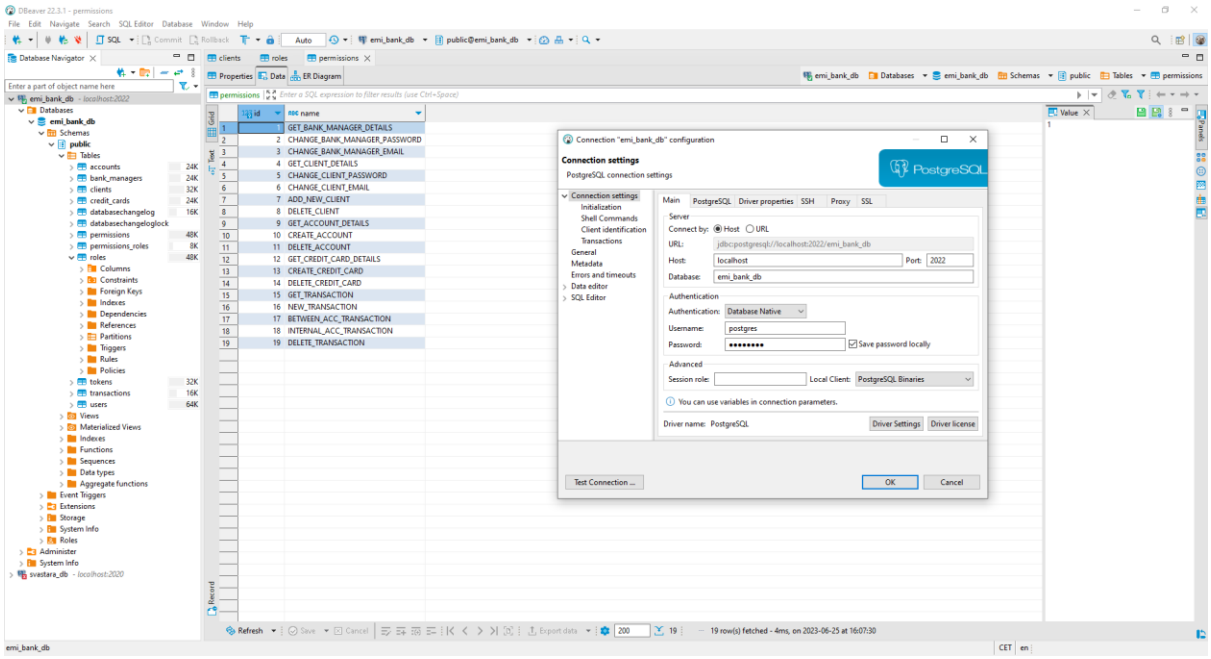
Postman - API platforma za izradu i korištenje API-ja. Postman pojednostavljuje svaki korak životnog ciklusa API-ja i usmjerava suradnju tako da možete stvarati bolje API-je. Jednostavno pohranjuje, katalogizira i surađuje oko svih API artefakata na jednoj središnjoj platformi. Postman može pohraniti i upravljati specifikacijama API-ja, dokumentacijom, receptima za tijek rada, testnim slučajevima i rezultatima, metrikama i svim ostalim što je povezano s API-ima.[13]

Postman se koristi u ovom radu za testiranje API poziva u aplikaciji. Preko Postman-a testiran je svaki zahtjev i provjeren svaki odziv API-a. Jednostavan i prigodan alat za korištenje pri izradi ove aplikacije. Na ***Slici 3.4.*** je prikazano korisničko sučelje Postman-a.



Slika 3.4. Izgled Postman korisničkog sučelja

DBeaver - je besplatni alat za baze podataka na više platformi za programere, administratore baza podataka, analitičare i sve koji rade s podacima. Podržava sve popularne SQL baze podataka. *DBeaver* je korišten u ovom radu za prikaz podataka koji se nalaze u bazi te njihovo uređivanje ako je potrebno. Pomoću ovog alata tokom izrade projekta bilo je lagano pratiti sve podatke spremljene u bazi i uočiti sve nepravilnosti tijekom stvaranja baze podataka. Na ***Slici 3.5.*** može se vidjeti prikaz korisničkog sučelja DBeavera gdje se prikazuje dijalog sa vezom i podacima o bazi koja se koristi. U pozadini se mogu vidjeti i struktura baze sa svim tablicama kao i podaci iz tablice *permissions*.



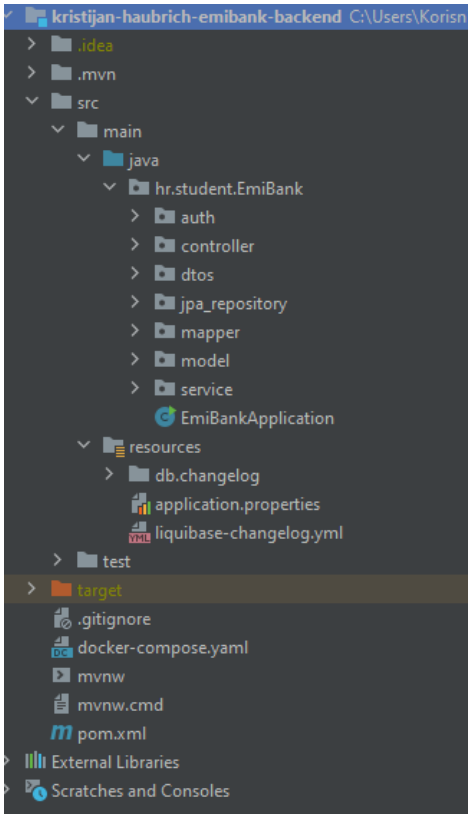
Slika 3.5. Izgled DBeaver korisničkog sučelja

Za manipulaciju valutama koristi se *exchangeratesapi* [14] API-ja. Pomoću ovog sučelja na serveru se obrađuju transakcije koje zahtijevaju pretvorbu jedne valute u drugu te se u korisničkom sučelju dohvaćaju sve trenutne valute da bi korisnik mogao odabrati valutu u slučaju pravljenja novog bankovnog računa ili stvaranja novih transakcija.

4. POSTUPAK IZRADE APLIKACIJE ZA NADGLEDANJE I UPRAVLJANJE RAČUNA U BANCI

U postupku izrade prvo se određuju alati koji će se koristiti za izradu aplikacije. Za spremanje baze podataka koristi se Docker. Za izradu API-a korišteno je SpringBoot Java programsko sučelje. Za izradu korisničkog sučelja koristi se React Native. Za migraciju podataka u bazu koji odgovaraju kodu aplikacije koristi se Liquibase. Za sigurnost prilikom prijave i registracije te određivanje koji korisnik može pristupiti kojim funkcionalnostima API-ja koriste se uloge i dopuštenja. Informacije o ulogama i dopuštenjima sadržane su u tokenima koji se koriste kad se korisnik prijavljuje. Tokeni služe i za autentifikaciju korisnika jednom kada se korisnik prijavio u aplikaciju da ne mora sljedećih 20 dana (trajanje osvježavajućeg tokena). Za lokalnu pohranu podataka koristila se Redux [15] biblioteka pomoću koje su se spremali podaci o trenutno prijavljenom korisniku. Cijeli programski *kôd* je javno dostupan na GitHub repozitoriju [16]. U ovom poglavlju objasnit će se kako su napravljeni svi dijelovi aplikacije kroz šest smislenih cjelina: struktura datoteka, sigurnost, registracija i prijava, bankovni menadžer, klijent i transakcije.

4.1. Struktura datoteka



Slika 4.1. Izgled strukture datoteka API-ja aplikacije

Unutar strukture datoteka u API-ja (***Slika 4.1.***) bitno je navesti najvažnije datoteke i direktorije. Unutar *src* direktorija nalazi se izvorni *kôd* aplikacije koji je podijeljen na *auth*, *controller*, *dtos*, *jpa_repository*, *mapper*, *model i service* direktorije te *EmiBankApplication* datoteku pomoću koje pokrećemo cijeli projekt. Unutar *auth* direktorija se nalaze sigurnosne značajke za prijavu i registraciju korisnika. Sve pristupne točke ispisane su u kontrolerima koji se nalaze unutar *controller* direktorija. Sve klase u kojima su sadržani podaci za prijenos podataka nalaze se u *dtos* direktoriju. Za entitete unutar API-ja koriste se klase kojom pristupamo i manipuliramo podacima u bazi. Navedene klase nalaze se unutar *jpa_repository* direktorija. Postoje klase koje sadrže metode za mapiranje podataka iz jedne klase u drugu što je korisno kod prijenosa podataka s korisničkog sučelja na bazu. Takve klase nalaze se unutar *mapper* direktorija. Modeli svih entiteta nalaze se u *model* direktoriju. Servisi pomoću kojih se obavljaju funkcionalnosti određenih pristupnih točaka nalaze se u *service* direktoriju. Unutar *resource* direktorija važno je spomenuti *db.changelog* direktorij u kojem se nalaze opisi svih tablica koje se kreiraju u bazi te *liquibase-changelog.yml* (***Slika 4.2.***) datoteka u kojoj se nalazi direktiva za stvaranje svih tablica opisanih u *db.changelog* direktoriju.

```yaml
databaseChangeLog:
  - includeAll:
      path:  /db.changelog/
```

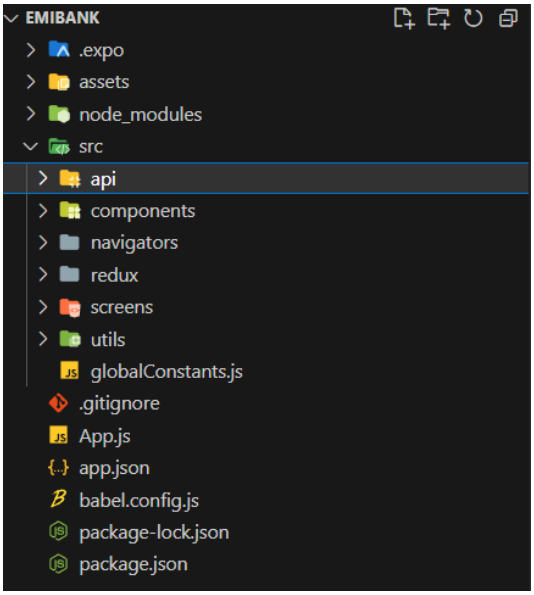
Slika 4.2. Isječak koda liquibase-changelog.yml datoteke

Za API bitno je još spomenuti *docker.compose.yaml* (***Slika 4.3.***) datoteku kojom se pokreće i/ili stvara (ako još nije stvoren) kontejner za bazu u Docker-u. Pokretanje kontejnera baze radi se pomoću direktive *docker compose up* unutar terminala.

```yaml
version: '3'
services:
  postgres:
    image: postgres
    environment:
      POSTGRES_DB: emi_bank_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "2022:5432"
```

Slika 4.3. Isječak koda docker.compose.yaml datoteke

Struktura datoteka korisničkog sučelja (***Slika 4.4.***) je vrlo jednostavna.



Slika 4.4. Izgled strukture datoteka korisničkog sučelja

Unutar *src* direktorija nalaze se svi važni direktorij korisničke aplikacije. Direktorij *api* sadrži sve potrebno za slanje zahtjeva API-ju aplikacije i *exchangeratesapi* API-ju za dohvaćanje valuta. Unutar *components* direktorija nalaze se sve komponente koje se koriste u prikazu zaslona korisničke aplikacije. Navigatori pomoću kojih se ostvaruje navigacija kroz zaslone nalaze se u *navigators* direktoriju. Sve potrebno za lokalno spremanje podataka nalazi unutar *redux* direktorija. Kodovi svih zaslona aplikacije sadržani su u *screens* direktoriju. Unutar *utils* direktorija nalaze se sve funkcije koje se globalno koriste u više zaslona. Datoteka *globalConstants.js* sadrži sve globalne konstante aplikacije. Još je važno napomenuti *App.js* datoteku u kojoj se nalazi logika prikaza cijele aplikacije.

4.2. Sigurnost

Sigurnost aplikacije odrađena je uz pomoć funkcionalnosti i klasa *SpringBoot Security*-a. Tu se nudi niz klasa kojima konfiguriramo sigurnost API-ja. Za razumijevanje sigurnosti API-ja aplikacije potrebno je razumjeti tri važna pojma: uloge, dopuštenja i tokeni. U ovom poglavlju bit će detaljno objašnjena njihova uloga.

4.2.1. Uloge i dopuštenja

Uloge i dopuštenja su entiteti pomoću kojih se provjerava može li korisnik koristiti određenu pristupnu točku. Svaki od ovih entiteta ima svoju tablicu u bazi. Tablice uloga i dopuštenja povezane su vezom više-na-više jer jedna uloga može imati više dopuštenja i obratno. Uloge

su povezane i s tablicom korisnika vezom jedan-na-više jer jednu ulogu može imati više korisnika. Tablica uloga naziva je *roles* i sadrži atribute *id* (primarni ključ) i *name* (ime uloge). Tablica dopuštenja naziva je *permissions* i sadrži atribute *id* (primarni ključ) i *name* (ime dopuštenja). Pomoćna tablica koja veže uloge i dopuštenja vezom više na više je naziva *permissions_roles* i sadrži atribute *roles_id* (primarni ključ uloge) i *permission_id* (primarni ključ dopuštenja). Uloge i dopuštenja svakog korisnika se spremaju u njegov token. Kada se prilikom dolaska zahtjeva za određenu pristupnu točku, dekodira token (ako je potreban token za pristupnu točku), provjeravaju se podaci o ulogama i dopuštenjima. Pristupne točke se lagano zaštite s ulogom ili dopuštenjem uz korištenje *@PreAuthorize* SpringBoot anotacije. Primjer jedne takve pristupne točke možemo vidjeti na ***Slici 4.5.***

```
@DeleteMapping("{clientEmail}")
@PreAuthorize("hasAuthority('DELETE_CLIENT')")
public IfResponseDto deleteClient(@PathVariable String clientEmail){
    return clientService.deleteClient(clientEmail);
}
```

Slika 4.5. Isječak koda pristupne točke za brisanje klijenta unutar ClientController.java datoteke

Uloge i dopuštenja su definirane tijekom kreiranja baze te se više ne mijenjaju. Postoje tri uloge: ROLE_ADMIN (programer koji održava aplikaciju), ROLE_BANK_MANAGER (bankovni menadžer) i ROLE_CLIENT (klijent). Svaka od tih uloga ima svoja dopuštenja za određene pristupne točke. ROLE_ADMIN ima sva dopuštenja dok ROLE_BANK_MANAGER i ROLE_CLIENT imaju samo ona koja su prikladna za njihovu ulogu. Na primjer, dopuštenje za stvaranje kreditne kartice (CREATE_CREDIT_CARD) ima samo ROLE_ADMIN i ROLE_BANK_MANAGER uloga dok ROLE_CLIENT nema to dopuštenje. Unatoč tome što aplikacija trenutno nema potrebu za dodavanjem ili izmjenom uloga i dopuštenja, napravljene su pristupne točke (unutar *RoleController.java* i *PermissionController.java* datoteka koje predstavljaju kontrolere za uloge i dopuštenja) koje služe za izmjenu, dodavanje ili brisanje uloga i dopuštenja ako to zatraži daljnji razvoj aplikacije.

4.2.2. Tokeni

Tokenima je glavna svrha da korisnika zadrže prijavljenim u aplikaciju na određeni period. Osim toga, služe i za neke osnovne informacije o korisniku. Vrsta tokena koji se koristi je JWT

Token koji je opisan u poglavlju 3.4. Tokeni se koriste u *JwtAuthenticationFilter.java* datoteci gdje se nalazi filter zahtjeva. Taj filter koristi token (koji se nalazi u *Authorization* zaglavlju zahtjeva) za provjeru korisnika te njegovih uloga i dopuštenja. Filter će zabraniti obradu zahtjeva ako korisnik nema dopuštenja za obradu zahtjeva ili je token istekao. Postoje dvije vrste tokena u aplikaciji: osvježavajući i pristupni. Oba tokena se generiraju pri svakoj uspješnoj prijavi korisnika u aplikaciju. Svaki korisnik kada se prijavi ima svoj osvježavajući i pristupni token spremljen lokalno. Trajanje pristupnog tokena je 3 sata dok je trajanje osvježavajućeg tokena 20 dana. Razlog ovakvoj raspodijeli trajanja tokena je jednostavan. Pristupni token služi samo tri sata nakon generiranja jer predstavlja token koji će se koristiti za sve zahtjeve prema API-ju. Tri sata predstavljaju okvirno trajanje jedne sesije korisnika. Kada pristupni token istekne, dobavlja se novi token od API-ja preko osvježavajućeg tokena. Novi pristupni token zamjenjuje lokalno s isteklim. Nakon 20 dana istekne i osvježavajući token te se korisnika preusmjeruje na novu prijavu. 20 dana u ovom slučaju predstavljaju maksimalno vrijeme korištenja aplikacije bez prijave. Tako se korisnik ne mora prijavljivati svaki put kada ponovno uđe u aplikaciju. Provjeru trajanja tokena na klijentskoj strani (korisničko sučelje) provjerava se pomoću *axios* biblioteke koja služi za formiranje i slanje zahtjeva API-ju aplikacije. Sve zahtjeve prije slanja se zaustavi (eng. *interception*) i provjerava se imaju li pristupni token (***Slika 4.6.***).

```
if(accessToken && refreshToken){
    const accessExpired = checkTokenExpiration(accessToken)
    const refreshExpired = checkTokenExpiration(refreshToken)
    if(refreshExpired){
        logout({dispatch})
    }else if(accessExpired && !refreshExpired){
        let accessTokenAxiosInstance
        if(isBankManager){
            accessTokenAxiosInstance =  axios.create({
                baseURL: `${baseUrl}/api/bank_managers/getAccessToken/${email}`,
            });
        }else{
            accessTokenAxiosInstance =  axios.create({
                baseURL: `${baseUrl}/api/clients/getAccessToken/${email}`,
            });
        }
        accessTokenAxiosInstance.defaults.headers.common['Authorization'] = `Bearer ${refreshToken}`
        const accessResponse = await accessTokenAxiosInstance.get()
        if(accessResponse?.data){
            dispatch(setAccessToken({token: accessResponse.data.token}))
            if(config.headers.Authorization){
                config.headers.Authorization = `Bearer ${accessResponse.data.token}`
            }
        }
    }
}
```

Slika 4.6. Isječak koda za provjeru tokena unutar apiRequest.js datoteke

Prvo se provjerava rok trajanja osvježavajućeg tokena ako zahtjev sadrži token. Korisnika se vraća na prijavu ako je istekao osvježavajući token. Ako je istekao samo pristupni token, pomoću osvježavajućeg tokena se šalje zahtjev za novi pristupni token. Novi pristupni token se sprema lokalno i stavlja se u zaglavlje zahtjeva koji se zaustavio. Potom se zahtjev šalje dalje na API. U slučaju da ni jedan token nije istekao zahtjev se šalje bez ikakvih promjena.

4.2.3. Konfiguracija sigurnosti

Kod konfiguracije sigurnosti posebno se misli na filtere zahtjeva te javne i zaštićene pristupne točke. Cijela konfiguracija se programira na serverskoj strani (API). Prvo se napravi filter zahtjeva koji se nalazi u *JwtAuthenticationFilter* klasi. Tamo je konfiguriran filter koji provjerava podatke korisnika i korišteni token. Radi se sigurnosna konfiguracija za cijeli API unutar *SecurityConfiguration* klase nakon što je napravljen filter zahtjeva koji osigurava pristup pristupnim točkama. Tu su postavljeni svi filteri koji će se koristiti za provjeru zahtjeva i javno dostupne pristupne točke. Javno dostupne pristupne točke su točke kojima bilo koji zahtjev može pristupiti bez potrebe tokena. To su zapravo svi zahtjevi koji se odvijaju prije prijave korisnika (zahtjevi na API prilikom prijave i registracije). Javno dostupne pristupne točke nalaze se unutar *PublicController* klase. Sve navedeno je postavljeno unutar *securityFilterChain* metode (***Slika 4.7.***) *SecurityConfiguration* klase.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http)throws Exception{
    http
            .csrf().disable() HttpSecurity
            .authorizeHttpRequests() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
            .antMatchers( ...antPatterns: "/api/public/**","/actuator/**","/error").permitAll()
            .anyRequest().authenticated()
            .and() HttpSecurity
            .sessionManagement() SessionManagementConfigurer<HttpSecurity>
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and() HttpSecurity
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

Slika 4.7. Isječak koda securityFilterChain metode unutar SecurityConfiguration klase

4.3. Registracija i prijava

Registracija i prijava prve su funkcionalnosti koje korisnik koristi kako bi stvorio svoj profil i prijavio se u isti. U ovom poglavlju opisano je sve vezano uz prijavu i registraciju. Od entiteta

spomenuti su *Users*, *Client*, *BankManager* te njihove tablice i veze. Postupci registracije i prijave odvijaju se u dva zaslona na korisničkom sučelju. Detaljan opis oba postupka nalazi se u nastavku ovog poglavlja.

4.3.1. *Users*

Korisnikov entitet u API-ju predstavlja klasa *Users*. *Users* je klasa koju nasljeđuju *BankManager* i *Client* o kojima će biti riječi u idućim poglavljima. *Users* također implementira SpringBoot sučelje *UserDetails* koje se koristi pri provjeri uloga i dopuštenja korisnika unutar sigurnosti spomenutim u 4.2. poglavlju. Klasa sadrži atribute *id,name,email,role* i *password.* Atribut *id* predstavlja jedinstveni identifikator koji se u tablici koristi kao primarni ključ. Atributi *name,email,password i role* predstavljaju puno ime, e-mail, lozinku i ulogu korisnika. Tri metode koje *Users* klasa sadrži su ekstremno bitne za prijavu i registraciju. Prva i najvažnija je *getAuthorities()* (***Slika 4.8.***) metoda koja vraća sva dopuštenja i uloge korisnika s obzirom na atribut *role*.

```
@Override
@Transactional
public Collection<? extends GrantedAuthority> getAuthorities() {
    Collection<SimpleGrantedAuthority> simpleGrantedAuthorities = new ArrayList<>();

    simpleGrantedAuthorities.add(new SimpleGrantedAuthority(role.getName()));

    this.getRole().getPermissions().forEach(
            permission -> simpleGrantedAuthorities.add(new SimpleGrantedAuthority(permission.getName()))
    );

    return simpleGrantedAuthorities;
}
```

Slika 4.8. Isječak metode getAuthorities() od Users klase

Metoda se koristi pri provjeri dopuštenja korisnika pristupnoj točci. Druge dvije metode su *getPassword()* i *getUsername()* koje vraćaju lozinku i e-mail korisnika. Ove metode se koriste pri potvrdi e-maila i lozinke korisnika koji se pokušava prijaviti. Tablica *users* sadrži ranije objašnjenje atribute *id,name,email,password* te *role_id* koji predstavlja primarni ključ tablice koja predstavlja ulogu (tablica *roles*) jer su *roles* i *users* tablice povezane vezom jedan-na-više. Prikaz stvaranja tablice *users* koristeći Liquibase prikazan je na ***Slici 4.9.***

```xml
<changeSet id="users_table" author="superuser" context="emibank">
    <createTable tableName="users">

        <column name="id" type="bigint" autoIncrement="true">
            <constraints nullable="false" primaryKey="true"/>
        </column>

        <column name="name" type="varchar">
            <constraints nullable="false" />
        </column>

        <column name="email" type="varchar">
            <constraints nullable="false" />
        </column>

        <column name="role_id" type="bigint">
            <constraints nullable="false" />
        </column>

        <column name="password" type="varchar">
            <constraints nullable="false" />
        </column>

    </createTable>

</changeSet>
```

Slika 4.9. Isječak koda v1.0.0-users_table.xml datoteke

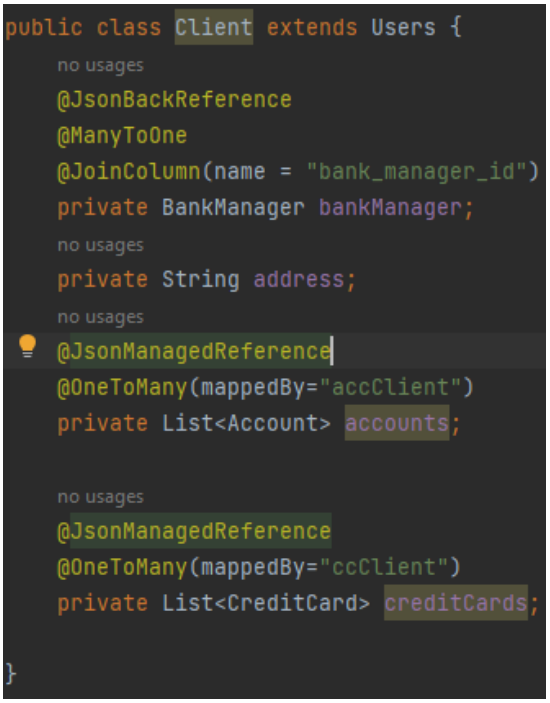
4.3.2. *BankManager*

Ovo je klasa koja predstavlja bankovnog menadžera u API-ju. Klasa nasljeđuje klasu *Users* opisanu u prethodnom poglavlju te jedini novi atribut koji sadrži jest lista klijenata koja je rezultat veze jedan-na-više između tablice koja predstavlja bankovnog menadžera (tablica *bank_managers*) i tablice koja predstavlja klijenta (tablica *clients*). Tablica *bank_managers* sadrži samo *id* kao primarni ključ dok sve ostalo sadrži tablica *users.*

4.3.3. *Client*

Klasa koja predstavlja klijenta u API-ju jest klasa *Client*. Nasljeđuje klasu *Users* opisanu u prethodnom poglavlju. Atribut *bankManager* je instanca *BankManager* kao rezultat veze jedan-na-više između bankovnog menadžera i klijenta. Klijent je povezan u bazi vezama jedan-na-više s tablicom koja predstavlja bankovne račune (tablica *accounts*) i tablicom koja predstavlja kreditne kartice (tablica *credit_cards*). Tako sadrži atribute koji imaju liste

bankovnih računa (atribut *accounts*) i liste kreditnih kartica (atribut *creditCards*). Klasa također sadrži atribut *address* koji predstavlja kućnu adresu klijenta. Klasa *Client* prikazana je na ***Slici 4.10.***



Slika 4.10. Isječak koda Client klase

4.3.4. Postupak registracije

Postupak registracije korisnika počinje onog trenutka kada korisnik stisne na ekranu tipku *Register* unutar zaslona za prijavu. Korisnika se odvodi na zaslon za registraciju gdje se prvo prikaže forma za registraciju korisnika kao klijenta. Registracija klijenta se izvršava tako da se prvo popune polja za unos imena klijenta, e-mail-a klijenta, lozinke klijenta, kućne adrese klijenta te e-mail-a bankovnog menadžera koji će biti zadužen za profil klijenta. Tako se osigurava da se klijent ne može prijaviti bez prisutnosti bankovnog menadžera koji je zadužen za njegov profil. Korisnik može kliknuti na gumb *Register* ako unese sve podatke ispravno. Kada korisnik stisne gumb *Register* sastavlja se zahtjev za registriranje korisnika uz pomoć prikupljenih podataka iz forme. Za zahtjev nije potreban pristupni token jer se radi o funkcionalnosti koja se odrađuje prije prijave korisnika. Putanja na koju se zahtjev šalje je */public/register/client.* Ova putanja je nastavak na osnovnu putanju za sve zahtjeve koja glasi *http://localhost:8080/api.* Za sve putanje u ostatku ovog rada navodit će se samo nastavci koji se dodaju na osnovnu putanju. API sprema sve podatke o novom korisniku pri čemu treba navesti da se lozinka korisnika kodira i sprema kao kodirani niz u bazu. API potom vraća

odgovor korisničkoj aplikaciji u kojem se nalazi samo informacija o uspješnoj registraciji klijenta. Za kraj korisnika se odvede na zaslon za prijavu. Prikaz zahtjeva prikazan je na ***Slici 4.11.***

```
const body = {
    name: nameInput,
    email: emailInput,
    password: passwordInput,
    address: addressInput,
    bankManagerEmail: bankManagerEmailInput};
const response = await apiRequest("").post("/public/register/client",body);
```

Slika 4.11. Isječak koda za zahtjev registriranja klijenta unutar zaslona za registraciju

Ako želimo korisnika registrirati kao bankovni menadžer označit ćemo kućicu s tekstom *Are you Bank Manager?*. Tada nam se na zaslonu prikaže forma za registraciju bankovnog menadžera koja se sastoji od polja za unos imena, e-mail-a i lozinke bankovnog menadžera. Ako korisnik ispuni ispravno sva navedena polja i stisne tipku *Register*, započet će se postupak registracije bankovnog menadžera. Putanja zahtjeva je */public/register/bankManager*. Prikaz slanja zahtjeva registracije bankovnog menadžera prikazan je na ***Slici 4.12.***

```
const body = {
    name: nameInput,
    email: emailInput,
    password: passwordInput,
    bankManagerRole: "ROLE_BANK_MANAGER"};
const response = await apiRequest("").post("/public/register/bankManager",body);
```

Slika 4.12. Isječak koda za zahtjev registriranja bankovnog menadžera unutar zaslona za registraciju

4.3.5. Postupak prijave

Postupak prijave korisnika u aplikaciju je jednostavan. Zaslon za prijavu je prvi zaslon koji se prikaže korisniku ako nije prijavljen u aplikaciju. Ukoliko se prijavljuje klijent, u aplikaciju mora popuniti polja za e-mail i lozinku i stisnuti gumb *Login*. Nakon toga započinje proces prijave. Prvo se pregledaju jesu li svi podaci ispravni i postoji li klijent u bazi podataka. Ako postoji, šalje se zahtjev na API za prijavu klijenta s prikupljenim podacima. Zahtjev nema token i ima putanju */public/authenticateClient.* Zahtjev se primi na API-ju te se izvrši provjera je li korisnik unio pravilne podatke. Korisničkoj aplikaciji vraćaju se podaci o korisniku. Podaci o

korisniku sadrže pristupni i osvježavajući token te podatke o klijentu. Tokeni korisnika prvo se kreiraju pomoću *generateToken()* metode od *JwtService* klase. Metoda prima listu svih uloga i dopuštenja koje ima korisnik, njegov pripadajući *Users* objekt te vrijeme trajanja tokena. Stoga se generira pristupni token u trajanju od 3 sata i osvježavajući token u trajanju od 20 dana. Postupak generiranja pristupnog tokena prikazan je na ***Slici 4.13.***

```
Map<String,Object> authorities = new HashMap<>();
List<String> permissions = new ArrayList<>();

userObj.getAuthorities().forEach(
        grantedAuthority -> permissions.add(grantedAuthority.getAuthority())
);

authorities.put("authorities",permissions);

String accessToken = jwtService.generateToken(authorities,userObj,accessTokenDuration);
```

Slika 4.13. Isječak koda za generiranje pristupnog tokena prilikom prijave unutar metode authenticateClient() od klase UserService

Server šalje odgovor s podacima klijenta i tokenima. Na klijentskoj strani lokalno se spremaju svi potrebni podaci za korištenje aplikacije nakon uspješne prijave klijenta. Lokalno spremanje podataka izvršava se pomoću *dispatch()* funkcije koja je dio *Redux* biblioteke. Unutar *dispatch()* funkcije stavljamo kao argument funkciju koja će promijeniti neko stanje u lokalnom spremniku. Primjerice *setClientData()* prima podatke o klijentu i sprema ih lokalno. Stoga se spremaju podaci o tome je li prijavljen klijent ili bankovni menadžer (*bankManagerChange()* ili *clientChange()*), podatak da je korisnik odsada prijavljen (*loginChange()*), podaci o korisniku (*setClientData()* ili *setBankManagerData()*) i tokeni (*setAccessToken()* i *setRefreshToken()*). Nakon što su svi podaci spremni, klijenta se odvodi na njegov profil. Prilikom prijave bankovnog menadžera jedino se spremi da je prijavljen bankovni menadžer te umjesto podataka o klijentu spremaju se podaci od bankovnog menadžera. Prikaz spremanja podataka te navigacije korisnika na njegov profil može se vidjeti na ***Slici 4.14***.

```
if(response?.data){
    const accessToken = response.data.accessToken
    const refreshToken = response.data.refreshToken

    dispatch(setAccessToken({token:accessToken}))
    dispatch(setRefreshToken({token:refreshToken}))

    setEmailInput("")
    setPasswordInput("")
    dispatch(loginChange())

    if(isBankManager){
        dispatch(bankManagerChange())
        dispatch(setBankManagerData({data:response.data.bankManagerDetailsResponseDto}))
        hardNavigate("BankManagerTabNavigation")
    }
    else{
        dispatch(clientChange())
        dispatch(setClientData({data:response.data.clientDetailsResponseDto}))
        hardNavigate("ClientTabNavigation")
    }
}else{
    createTwoButtonAlert("ERROR","Fail to login")
}
```

Slika 4.14. Isječak koda za spremanje podataka o prijavljenom korisniku unutar zaslona za prijavu

4.4. Bankovni menadžer

U ovom poglavlju opisat će se sve funkcionalnosti koje ima bankovni menadžer kada se prijavi u aplikaciju. Prilikom opisa obuhvatit će se svi zasloni s kojima se bankovni menadžer može susresti osim zaslona vezanih za izvršavanje novih transakcija. Uz to opisat će se entiteti za bankovne račune i kreditne kartice.

4.4.1. *Account*

Entitet koji predstavlja bankovni račun klijenta predstavljen je klasom *Account*. Od osnovnih atributa klasa ima *id, accNum, accType, balance, currency* koji redom predstavljaju jedinstveni identifikator (primarni ključ u bazi podataka), broj računa, tip računa, stanje računa i valutu računa. Broj računa u aplikaciji se sastoji od početnih slova *HR* i 13 brojeva te također predstavlja jedinstveni atribut za svaki račun. Primjer jednog takvog broja računa bi bio *HR1234567890123*. Tip računa može biti bilo koji tip koji bankovni menadžer u dogovoru s klijentom odluči, a jedan takav tip je *žiro* račun. Stanje računa je količina financijskih sredstava koju račun posjeduje. Valuta računa predstavlja oznaku valute na tržištu prema *ISO 4217*

standardu i sastoji se od tri slova koja je jedinstveno predstavljaju. Primjerice, oznaka za valutu euro je *EUR*. Osim osnovnih atributa, tu su još tri atributa koji su rezultat veza ostalih tablica s tablicom koja predstavlja ovaj entitet (tablica *accounts*). Prvi je atribut *creditCard* koji je instanca *CreditCard* klase koja predstavlja entitet kreditne kartice. Veza između kreditne kartice i bankovnog računa je jedan-na-jedan. Potom postoji atribut *transactions* koji je lista objekata klase *Transactions* koja predstavlja entitet transakcija u aplikaciji. Veza između bankovnog računa i transakcija je jedan-na-više. Na kraju postoji atribut *accClient* koji je instanca već opisane klase *Client.* Veza između klijenta i bankovnog računa je jedan-na-više. Klasa *Account* prikazana je na ***Slici 4.15***.

```
public class Account{

    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private String accNum;
    no usages
    private String accType;
    no usages
    private Double balance;
    no usages
    private String currency;

    no usages
    @JsonBackReference
    @OneToOne(mappedBy = "account",cascade = CascadeType.ALL,orphanRemoval = true)
    private CreditCard creditCard;

    no usages
    @JsonManagedReference
    @OneToMany(mappedBy="account")
    private List<Transaction> transactions;

    no usages
    @JsonBackReference
    @ManyToOne
    @JoinColumn(name = "client_id",nullable = false)
    private Client accClient;
}
```

Slika 4.15. Isječak koda Account klase

4.4.2. *CreditCard*

Klasa *CreditCard* predstavlja entitet kreditne kartice. Od osnovnih atributa sadrži *id* (jedinstveni identifikator i primarni ključ), *cardNum* (broj kartice), *cardType* (tip kartice), *cardLimit* (limit kartice) i *balance* (stanje kartice). Broj kartice se sastoji od 16 brojeva koji jedinstveno opisuju karticu. Tip kartice odlučuje bankovni menadžer u dogovoru s klijentom, a jedan takav tip je *debitna* kartica. Limit kartice predstavlja maksimalna financijska sredstva koja klijent može povući s kartice u jednoj transakciji. Klasa *CreditCard* prikazana je na ***Slici 4.16***.

```
public class CreditCard {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    @JsonManagedReference
    @OneToOne
    @JoinColumn(name = "account_id")
    private Account account;
    no usages
    @JsonBackReference
    @ManyToOne
    @JoinColumn(name = "client_id")
    private Client ccClient;
    no usages
    private String cardNum;
    no usages
    private String cardType;
    no usages
    private Double cardLimit;
    no usages
    private Double balance;

}
```

Slika 4.16. Isječak koda CreditCard klase

4.4.3. Zaslon profila

Zaslon profila je prvi zaslon koji se prikaže bankovnom menadžeru kada se prijavi u aplikaciju. Na zaslonu bankovni menadžer vidi svoje ime, popis svojih klijenata koje može pretraživati, gumb za osvježavanje podataka te gumb za odjavu. Ako se stisne gumb za osvježavanje

podataka (*Refresh*), korisniku se otvara mali prozor za potvrdu osvježavanja podataka. Ako korisnik pristane, šalje se zahtjev za dohvaćanje najnovijih podataka o bankovnom menadžeru s baze. Zahtjev sadrži pristupni token te ima putanju */bank_managers/{email}* gdje *{email}* predstavlja e-mail bankovnog menadžera. Kada se dobiju podaci iz baze pohranjuju se na lokalnu bazu. Gumb za odjavu naziva *Logout* služi za odjavu korisnika te ga vraća na prijavu. U slučaju da korisnik pritisne na nekog od svojih klijenata, odvest će ga se na profil odabranog klijenta.

4.4.4. Zaslon postavki

Odabere li korisnik zaslon za postavke u navigaciji, prikazat će mu se forma za promjenu lozinke i gumb za brisanje vlastitog profila (*Delete account*). Unutar forme za promjenu lozinke korisnik mora popuniti polja za unos stare i nove lozinke. Prije promjene lozinke provjerava se da li se stara lozinka podudara s onom u bazi. Kada korisnik pritisne gumb *ChangePassword* lozinka će se promijeniti ako je sve pravilno uneseno.

Ako korisnik pritisne *Delete account*, njegov profil će se obrisati u slučaju da na njegovom profilu više nema klijenata koje nadgleda.

4.4.5. Zaslon profila klijenta

Popis klijenata bankovnog menadžera nalazi se na zaslonu profila. Ako korisnik odabere jednog klijenta, prelazi se na zaslon profila klijenta. Zaslon sadrži podatke o imenu, e-mailu, kućnoj adresi, listi bankovnih računa klijenta te gumbove za dodavanje novog bankovnog računa (*Create Bank Account*) i gumb za brisanje profila klijenta (*Delete account*). Ako korisnik pritisne na *Create Bank Account*, odvodi ga se na zaslon sa formom za stvaranje novog računa. Ako korisnik pritisne na neki od računa klijenta, odvodi ga se na prikaz detalja odabranog bankovnog računa.

U slučaju da korisnik želi obrisati profil klijenta i stisne na *Delete account*, obrisat će se profil u bazi. Ako korisnik želi vidjeti trenutne promjene mora osvježiti podatke na zaslonu profila.

4.4.6. Zaslon za pravljenje novog bankovnog računa

Na zaslonu za pravljenje novog bankovnog računa korisnik popunjava formu za pravljenje novog bankovnog računa klijentu. Prije prikazivanja zaslona moraju se dohvatiti sve dostupne valute preko *exchangeratesapi*-a. Ovaj API nudi niz pristupnih točaka za valute, a na ovom

zaslonu je potrebna lista svih valuta. Stoga se šalje zahtjev API-ju za dohvaćanje svih valuta s putanjom *https://api.apilayer.com/exchangerates_data/symbols*. U svrhu ovog diplomskog rada koristio se besplatan pristup kojim aplikacija ima pravo na maksimalno 1000 zahtjeva mjesečno. Prikaz cijelog zahtjeva prikazan je na ***Slici 4.17.*** gdje je zamagljen *apikey* atribut jer je povjerljiv za svakog korisnika API-ja.

```
import axios from "axios"
const currencyRequest = async () => {
    return axios.get("https://api.apilayer.com/exchangerates_data/symbols",{
                headers: {
                'Content-Type': 'application/json',
                'apikey':
                }})
}

export default currencyRequest
```

Slika 4.17. Isječak koda currencyRequest.js datoteke

Odgovor na zahtjev treba čekati i po nekoliko sekundi. Dok se čeka odgovor na zahtjev, korisniku se prikazuje zaslon za učitavanje (eng. *loading screen*) koji se koristi na puno mjesta gdje se prije prikaza zaslona moraju čekati neki podaci. Ovaj zaslon prima kao argument samo naslov koji će se prikazivati korisniku dok čeka. Kada podaci o valutama stignu, nastavlja se s prikazom originalnog zaslona. Korisnik popunjava polja za unos broja računa, tip računa, stanje računa te valutu. Valuta se upisuje na način da se upisuje njena oznaka opisana u poglavlju 4.4.1. Korisnik može sam unijeti oznaku ili pritisnuti na padajući izbornik *AVAILABLE CURRENCIES*. U padajućem izborniku nalaze se sve trenutno dostupne valute. Korisnik klikom na jednu valutu automatski odabire istu. Prije slanja zahtjeva na server, provjerava se je li dobro unesena valuta uz ostale provjere. Ako je sve ispravno uneseno, račun se dodaje u bazu i korisnika se vraća nazad na zaslon profila.

4.4.7. Zaslon bankovnog računa

Ovaj zaslon prikazuje sve podatke o nekom bankovnom računu. Podaci koje sadrži su broj računa, stanje računa, podaci o kreditnoj kartici (broj kartice, limit kartice i tip kartice) ako ih račun ima te podaci o provedenim transakcijama ako postoje. Pri dnu postoji gumb *Make Transaction* za stvaranje nove transakcije i gumb za brisanje bankovnog računa *Delete Bank Account*. Ukoliko račun nema kreditne kartice, na mjesto podataka o kreditnoj kartici nalazi se gumb za dodavanje kreditne kartice *Add Credit Card*. Ako korisnik pritisne *Add Credit Card* odvodi ga se na zaslon s formom za dodavanje nove kreditne kartice. Ako korisnik pritisne

Delete Bank Account cijeli se račun briše iz baze te se korisnika odvodi na zaslon profila. Pritisne li korisnik *Make Transaction*, odvodi ga se na zaslon za odabir tipa nove transakcije. Korisnik može provedene transakcije vidjeti i pretraživati po vremenu izvršenja transakcije. Odabere li jednu od transakcija, prikazat će mu se detalji transakcije (datum izvršenja, tip transakcije i količina sredstava) u zaslonu za prikaz detalja transakcije.

4.4.8. Zaslon za dodavanje kreditne kartice

Unutar ovog zaslona nalazi se forma za stvaranje nove kreditne kartice računa. Iznad forme pišu podaci o broju bankovnog računa i e-mailu klijenta koji je vlasnik računa. Forma sadrži polja za unos broja kreditne kartice, tipa kreditne kartice i limita kartice. Prije nego li se doda kartica u bazu, postoji provjera pravilnog unosa limita kartice. Provjera prvo provjerava je li uneseni limit pozitivni broj a potom uz pomoć metode *validNumberOfDecimals()* provjeri ima li broj više od dvije znamenke. Isječak koda metode *validNumberOfDecimals()* može se vidjeti na ***Slici 4.18.*** Ako se sve ispravno unese, kreditna kartica se dodaje u bazu.

```
const validNumberOfDecimals = ()=> {
    if(creditCardLimit.toString().includes(".")){
        return creditCardLimit.toString().split('.')[1].length <= 2
    }
    return true
}
```

Slika 4.18. Isječak koda validNumberOfDecimals() metode unutar zaslona za dodavanje kreditne kartice

4.5. Klijent

U ovom poglavlju objasnit će se sve mogućnosti prijavljenog klijenta. Dosta zaslona koji se prikazuju bankovnom menadžeru prikazuju se i klijentu samo s manjim brojem funkcionalnosti pa će ovo poglavlje biti skraćeno objašnjavanja zaslona koji su već objašnjeni.

Klijent se nakon prijave nalazi u zaslonu profila kada je prijavljen klijent. Tu su prikazani podaci o imenu klijenta, e-mailu klijenta i bankovnim računima klijenta. Pri dnu se nalazi gumb za osvježavanje podataka *Refresh Data* te gumb za odjavu *Logout*. Ako korisnik pritisne neki od njegovih bankovnih računa, odvede ga se na detaljan prikaz odabranog bankovnog računa. Korisnik osvježava svoje podatke pritiskom na gumb *Refresh Data*. Pritisne li *Logout*, klijent će se odjaviti i preusmjeriti na zaslon za prijavu. Odabere li korisnik u navigaciji zaslon postavki, prikazat će mu se samo forma za promjenu lozinke koja funkcionira na isti način kao

što je opisano i u poglavlju 4.4.4. Ako korisnik odabere jedan od svojih bankovnih računa, odvest će ga na *AccountScreen* zaslon. Ovdje su korisniku prikazani svi detalji o računu kao što je objašnjeno i u poglavlju 4.4.7. Razlika je što klijent nema opciju dodavanja kreditne kartice kao ni brisanja bankovnog računa. Klijent ovdje može pregledati transakcije bankovnog računa kao i bankovni menadžer. Klijent također može izvršiti transakcije kao i bankovni menadžer ali ne smije dodavati nikakva sredstva na račun jer je to jedino dopušteno bankovnom menadžeru.

4.6. Transakcije

Transakcije su jedan od najbitnijih dijelova aplikacije i jedan od glavnih razloga postojanja iste. Postoje tri vrste transakcije u aplikaciji: uplata sredstava na bankovni račun, isplata sredstava s računa i isplata sredstava na drugi bankovni račun. U ovom poglavlju opisat će se entitet transakcije, tri vrste transakcija i najbitnije funkcije u izvršavanju transakcije.

4.6.1. *Transaction*

Klasa *Transaction* (***Slika 4.19.***) predstavlja entitet transakcije.

```
public class Transaction {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    @JsonBackReference
    @ManyToOne
    @JoinColumn(name = "account_id",nullable = false)
    private Account account;
    no usages
    private String fromAccNum;
    no usages
    private String date;
    no usages
    private String toAccNum;
    no usages
    private String transactionType;
    no usages
    private Double amount;
    no usages
    private String currency;
}
```

Slika 4.19. Isječak koda Transaction klase

Od osnovnih atributa sadrži *id* (primarni ključ i jedinstveni identifikator), *fromAccNum* (broj računa s kojeg se uplaćuju ili podižu sredstva), *toAccNum* (broj računa na koji se uplaćuju sredstva), *date* (datum i vrijeme u kojem je izvršena transakcija), *transactionType* (tip transakcije), *amount* (količina sredstava) i *currency* (valuta transakcije). Uz osnovne atribute sadrži i atribut *account* koji je instanca *Account* klase. Ovaj atribut je nastao zbog veze jedan-na-više između bankovnog računa i transakcije.

4.6.2. Uplata

Uplatu financijskih sredstava na isti bankovni račun može izvršiti samo bankovni menadžer. Ako korisnik pritisne gumb *Top up balance,* unutar zaslona za odabir vrste transakcije odvodi ga se na zaslon za stvaranje nove transakcije koji prikazuje podatke o tipu transakcije, računu na koji se uplaćuju sredstva, valutu transakcije te polje za unos količine sredstava koji se uplaćuju. Ako korisnik unese ispravnu količinu sredstava i pritisne gumb *Execute Transaction,* šalje se zahtjev za provedbom transakcije na server. Transakcija se obrađuje unutar *executeInternalAccountTransaction()* metode od *TransactionService* klase. Prvo se kreira vrijeme izvršavanja transakcije i zapisuje ga se u obliku teksta. Potom se količina sredstava pretvara u decimalni zapis s najviše dvije decimale. Količina koja je zapisana kao tekst preda se kao argument konstruktoru za stvaranje objekta *BigDecimal* tipa. Prilikom stvaranja *BigDecimal* objekta (***Slika 4.20.***) postavi se na koliko će se decimala zaokružit broj i kojom metodom (u aplikaciji se koristi *HALF-DOWN* metoda koja zaokruži drugu decimalu na manji broj ako je treća decimala manja od 6).

```
String amount = transaction.getAmount();
BigDecimal realAmount = new BigDecimal(amount).setScale( newScale: 2, RoundingMode.HALF_DOWN);
```

Slika 4.20. Isječak koda stvaranja BigDecimal objekta unutar metode executeInternalAccountTransaction() klase Transaction Service

Bitno je da argument bude tekst, a ne decimalna vrijednost jer se lagano može dogoditi greška pri direktnom zaokruživanju decimalnih vrijednosti (decimalna vrijednost sklona je greškama u zaokruživanju). Također, isto se radi kada se zapisuje nova vrijednost stanja računa. Potom se provjeri tip transakcije (uplata ili isplata) i izvrši ista. Nakon što se transakcija izvrši korisničkoj aplikaciji se šalje podatak o uspješnosti izvršenja transakcije.

4.6.3. Isplata

Isplatu financijskih sredstava mogu izvršiti i klijent i bankovni menadžer. Simulira isplatu sredstava s kartice računa ili bilo kojim drugim načinom. Kako bi korisnik ostvario isplatu, mora pritisnuti gumb *Withdrawal* unutar zaslona za odabir tipa transakcije. To ga odvodi na zaslon za novu transakciju koji sadrži podatke o tipu transakcije, bankovnom računu s kojeg se isplaćuju sredstva te valuti transakcije. Korisnik mora unijeti pravilnu količinu sredstava i pritisnuti gumb *Execute Transaction* da bi poslao zahtjev serveru za provedbu transakcije. Transakcija se obrađuje unutar *executeInternalAccountTransaction()* metode od *TransactionService* klase. Ovdje se prvo sprema datum i vrijeme izvršenja transakcije. Potom se pretvori količina u decimalni broj s maksimalno dva decimalna mjesta. Odredi se tip transakcije i provjeri se je li moguće izvršiti transakciju. Provjera se izvršava metodom *isTransactionPossible()* (***Slika 4.21.***).

```
@Transactional
public Boolean isTransactionPossible(Account account, Double amount)throws Exception{
    ExchangeRate converter = new ExchangeRate();
    BigDecimal realAmount = new BigDecimal(String.valueOf(amount)).setScale( newScale: 2,RoundingMode.HALF_DOWN);

    BigDecimal balanceAfterTransaction = new BigDecimal(
            String.valueOf( d: account.getBalance()-realAmount.doubleValue()))
            .setScale( newScale: 2,RoundingMode.HALF_DOWN);

    if(!account.getCurrency().equals("USD")){
        balanceAfterTransaction = new BigDecimal(
                String.valueOf(converter.exchangeRate(
                        account.getCurrency(),
                        toCurrency: "USD" ,
                        balanceAfterTransaction.doubleValue())))
                .setScale( newScale: 2,RoundingMode.HALF_DOWN);
    }

    if(account.getCreditCard() != null)
        return  balanceAfterTransaction.doubleValue() > -1000D &&
                account.getCreditCard().getCardLimit() > realAmount.doubleValue();
    return balanceAfterTransaction.doubleValue() > -1000D;
}
```

Slika 4.21. Isječak koda isTransactionPossible() metode od TransactionService klase

Ova metoda provjerava dva važna uvjeta izvršavanja transakcije. Prvi uvjet je da stanje računa nakon transakcije ne smije biti ispod 1000$ neovisno o valuti računa. Drugi uvjet je ako račun ima karticu, iznos transakcije ne smije prelaziti limit kartice. Nakon što se transakcija izvršila, korisničko sučelje se obavještava o uspješnosti izvršenja transakcije.

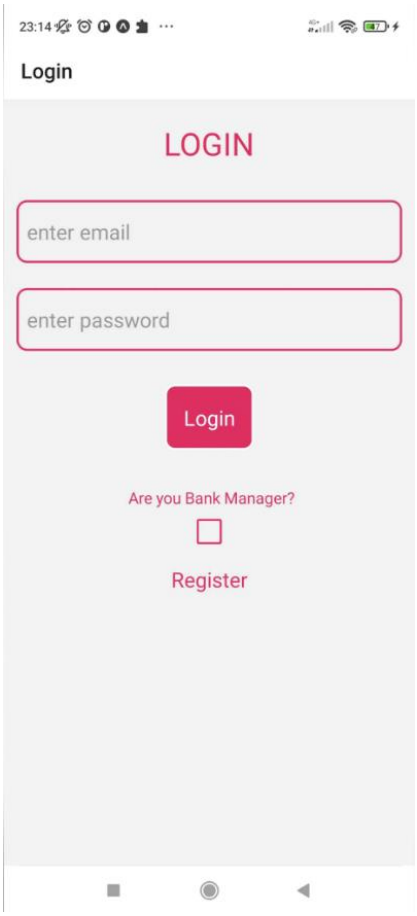
4.6.4. Isplata na drugi račun

Isplata na drugi račun je transakcija u kojoj se sredstva prenose s jednog bankovnog računa na drugi. Mogu je izvršiti i klijenti i bankovni menadžeri. Korisnik prvo pritisne gumb *Payment to another account* unutar zaslona za odabir tipa transakcije. Korisnika to odvodi na zaslon za novu transakciju koji sadrži podatke o tipu transakcije i broju računa s kojeg se šalju sredstva. Ovaj puta zaslon ima i padajući izbornik sa svim trenutno dostupnim valutama. Korisnik mora unijeti broj računa na koji uplaćuje sredstva te količinu sredstava i valutu u kojoj će biti odrađena transakcija. Valuta u kojoj će se odraditi transakcija ne mora biti jednaka valuti računa s kojeg se sredstva šalju kao ni valuti računa na kojeg se sredstva šalju. Ako korisnik unese sve potrebne podatke ispravno i pritisne gumb *Execute Transaction*, poslat će se zahtjev za izvršavanje transakcije na server. Na serveru će se prvo spremiti datum u kojem se izvršila transakcija. Zatim će se ,ako je potrebno, pretvoriti količina sredstava transakcije (koja je izražena u valuti transakcije) u količinu sredstava transakcije izraženu u valutama za svaki račun posebno. Potom će se provjeriti mogućnost prenošenja sredstva s odabranog računa metodom *isTransactionPossible()* koja je opisana detaljno u poglavlju 4.6.3. Ako je sve korektno i ispravno, transakcija će se izvršiti tako da se oduzme količina sredstava od računa s kojeg se šalju sredstva i dodaju se na račun kojem se šalju sredstva. Računima se dodaje ili oduzima količina sredstava u vlastitoj valuti koja je jednaka vrijednosti količine sredstava u valuti transakcije. Nakon što se transakcija izvršila korisničko sučelje se obavještava o uspješnosti izvršenja transakcije.

5. PRIKAZ APLIKACIJE ZA NADGLEDANJE I UPRAVLJANJE RAČUNA U BANCI

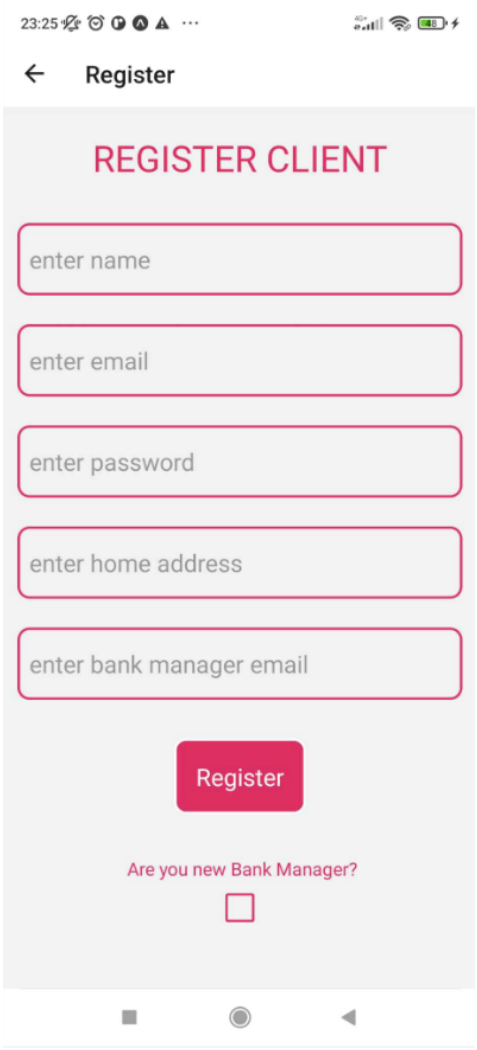
U ovom poglavlju opisat će se i prikazati izgled aplikacije i korisničkog sučelja. Pratit će se prirodni i očekivani prolasci klijenta i bankovnog menadžera kroz aplikaciju. Uz prikaz korisničkog sučelja, ukratko će se opisati funkcionalnosti koje svaki element korisničkog sučelja odrađuje.

Kada korisnik pokrene aplikaciju prikazat će mu se zaslon za prijavu (***Slika 5.1.***).



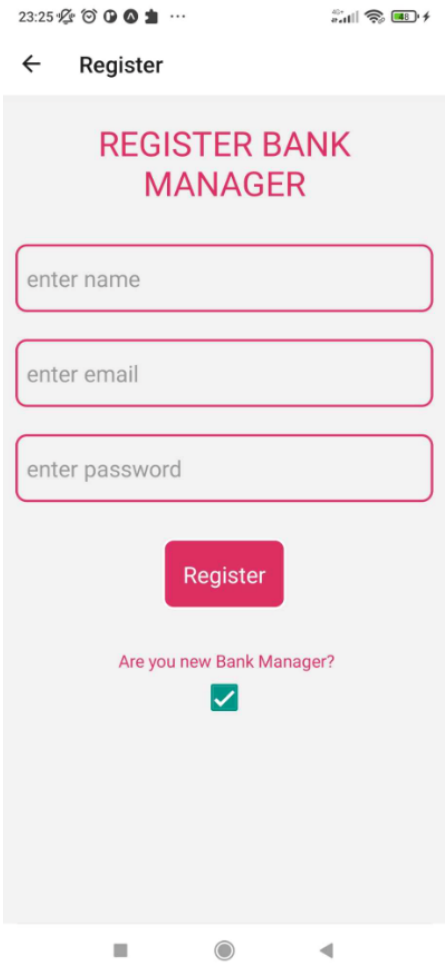
Slika 5.1. Prikaz zaslona za prijavu

Korisnik se može prijaviti u aplikaciju preko popunjavanja polja za unos e-mail-a i lozinke i pritiska na gumb *Login*. U slučaju da je korisnik bankovni menadžer prije prijave označit će *Are you Bank Manager?*. Ako korisnik nema svoj profil, pritisnuti će tipku *Register* kako bi se registrirao. Korisnika se tada odvodi na zaslon za registraciju (***Slika 5.2.***)



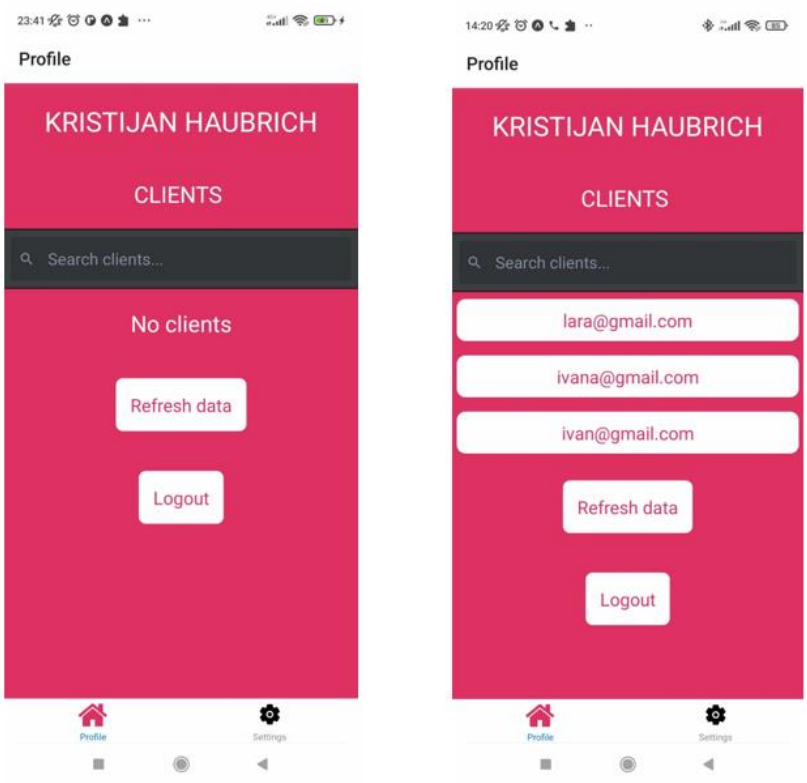
Slika 5.2. Prikaz zaslona za registraciju pri registraciji klijenta

Zaslon prvo prikazuje formu za registraciju klijenta koju korisnik može popunit i registrirati se kao klijent. U slučaju da se želi registrirati kao bankovni menadžer, korisnik označi *Are you new Bank Manager?* pri čemu se otvori forma za registraciju bankovnog menadžera (***Slika 5.3.***). Ako korisnik pravilno unese sva polja, može se registrirati kao bankovni menadžer.



Slika 5.3. Prikaz zaslona za registraciju pri registraciji bankovnog menadžera

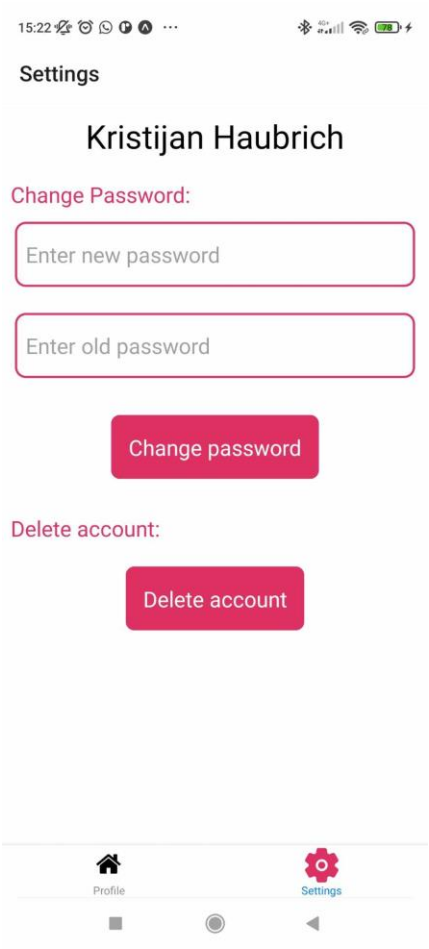
Korisnik se može prijaviti u aplikaciju nakon što se registrirao. Prilikom prijave, korisnik unosi svoj e-mail i lozinku. Kada se prvi put prijavi, bankovni menadžer u aplikaciju dospije na zaslon profila (***Slika 5.4.***).



z

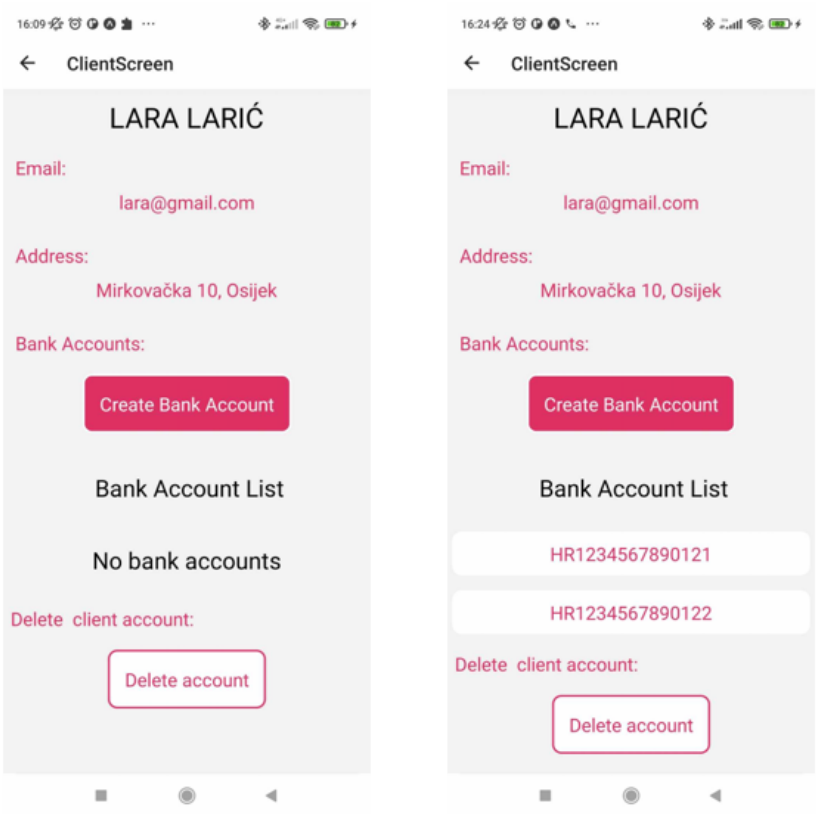
Slika 5.4. Prikazi zaslona profila bankovnog menadžera

Na ***Slici 5.4.*** se vidi zaslon kada korisnik nema ni jedan klijentski račun i kad ima tri klijentska računa. Korisnik može otići na zaslon postavki za mijenjanje lozinke ili brisanja vlastitog računa uz pomoć navigacije na dnu zaslona. Svoje podatke može osvježiti pritiskom na gumb *Refresh* ili se odjaviti pritiskom na gumb *Logout*. Korisnik ima i opciju pretraživanja svojih klijenata. Ako korisnik odabere jednog od klijenata na njegovoj listi, odvest će ga se na njegov profil. Izgled zaslona postavki može se vidjeti na ***Slici 5.5.***



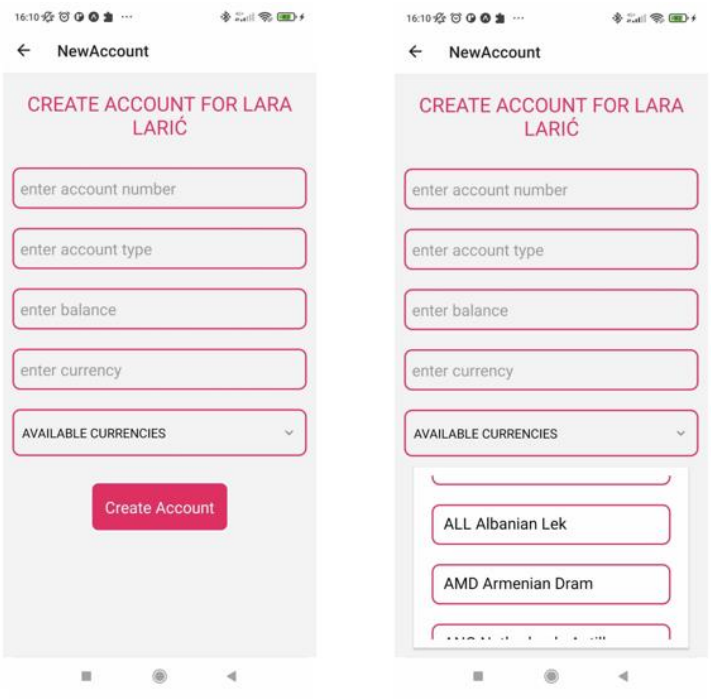
Slika 5.5. Prikaz zaslona postavki

Zaslon postavki ima opciju mijenjanja lozinke. Korisnik ima i opciju brisanja računa ako su pritom svi klijenti korisnika obrisani ili premješteni na brigu drugom bankovnom menadžeru. Ako korisnik odabere nekog od svojih klijenata unutar zaslona profila, otvara se profil odabranog klijenta na zaslonu profila klijenta(***Slika 5.6.***).



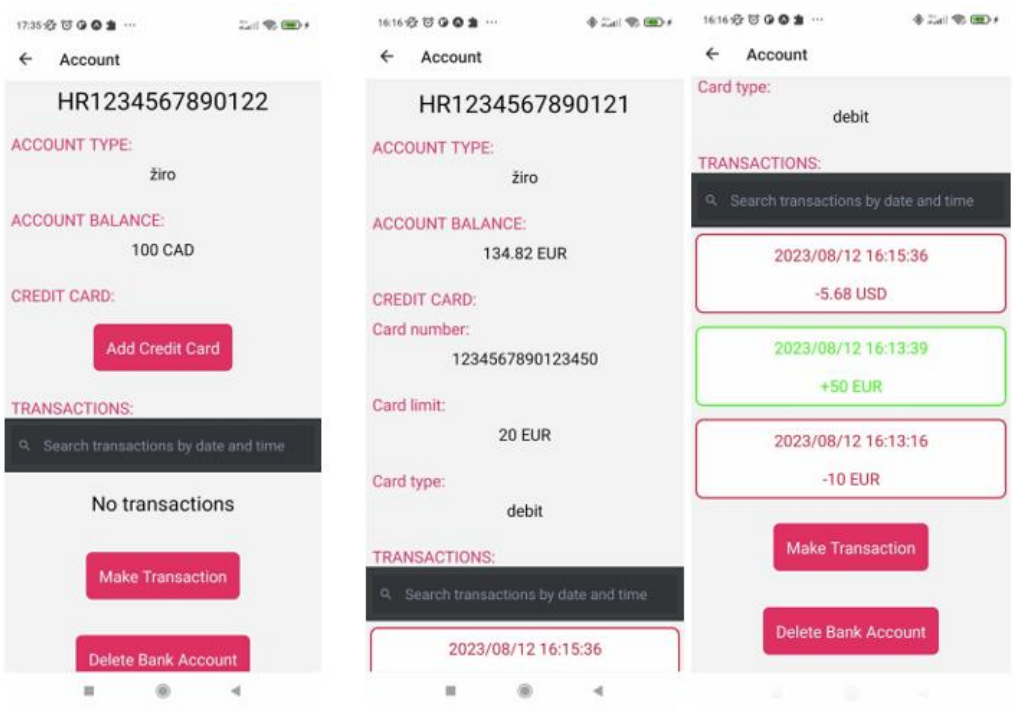
Slika 5.6. Prikazi zaslona profila klijenta

Na ***Slici 5.6.*** se može vidjeti zaslon kada je klijent bez bankovnih računa i kada ima dva bankovna računa. Bankovni menadžer može dodati novi bankovni račun na gumb *Create Bank Account* ili obrisati cijeli profil od klijenta klikom na gumb *Delete account*. Ako korisnik odluči stvoriti novi bankovni račun klijentu, odvodi ga se na zaslon za dodavanje novog računa (***Slika 5.7.***).



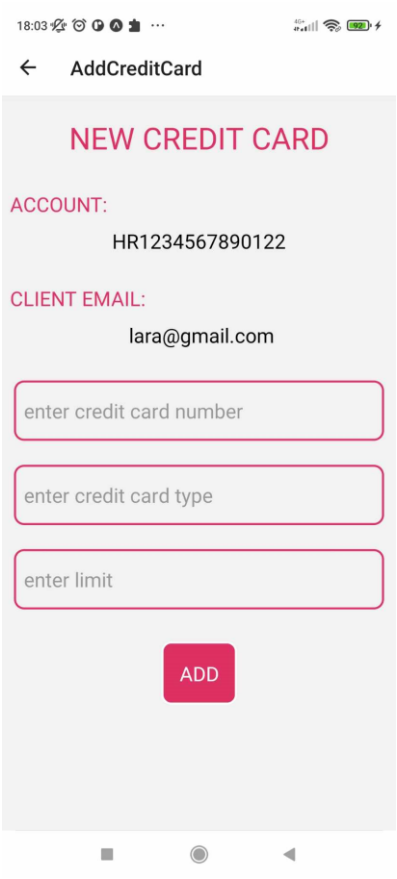
Slika 5.7. Prikazi zaslona za dodavanje novog računa

Slika 5.7. prikazuje zaslon kada nije otvoren padajući izbornik za valute i kada je otvoren isti. Ako korisnik pravilno unese sva polja i pravilno odabere valutu, može stvoriti novi bankovni račun pritiskom na gumb *Create Account.* Odabere li korisnik neki od bankovnih računa unutar zaslona profila klijenta, otvorit će mu se zaslon bankovnog računa (***Slika 5.8.***) koji prikazuje detalje odabranog bankovnog računa.



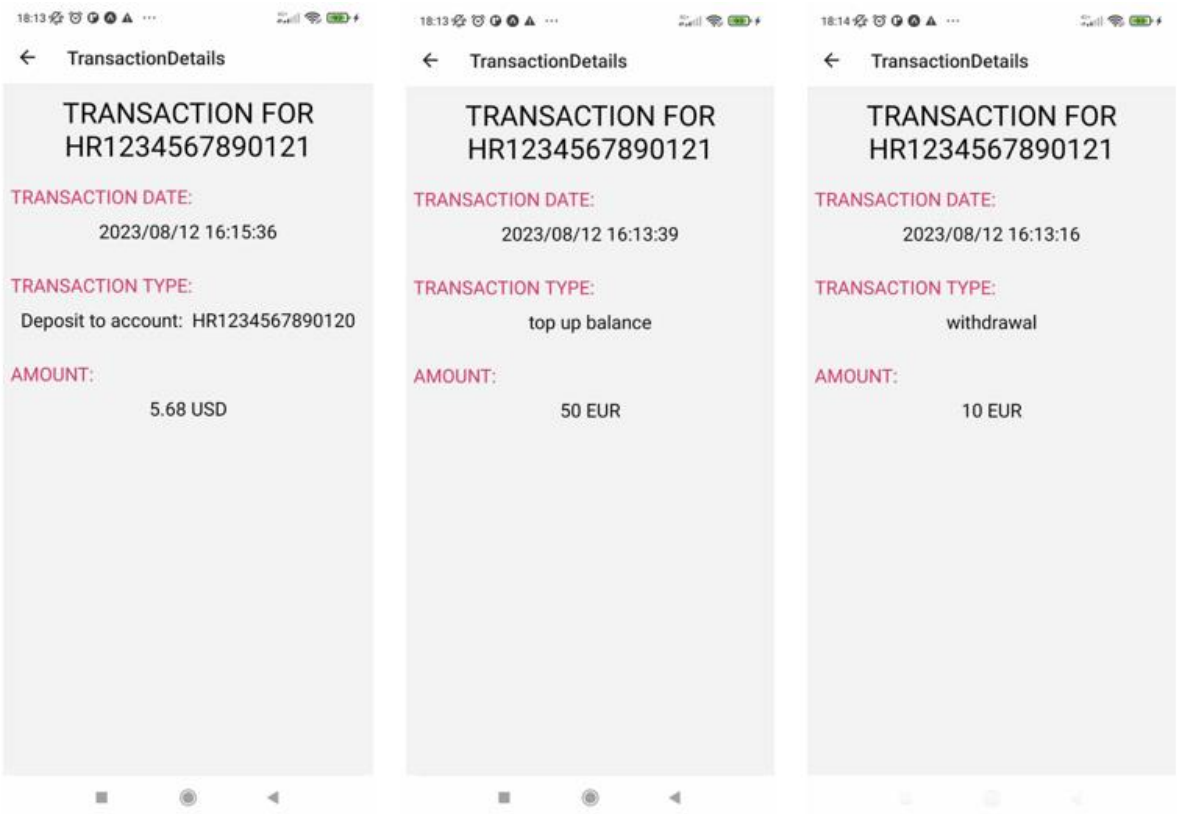
Slika 5.8. Prikazi zaslona bankovnog računa

Slika 5.8. prikazuje dva primjera zaslona bankovnog računa uz pomoć tri odvojena prikaza. Lijevi prikaz pokazuje zaslon kada bankovni račun nema kreditnu karticu ni transakcije. Prikazi u sredini i desno prikazuju bankovni račun koji ima i kreditnu karticu i transakcije. Ako račun nema kreditnu karticu, bankovni menadžer može dodati istu pritiskom na gumb *Add Credit Card*. Korisnik može izvršiti novu transakciju za račun pritiskom na gumb *Make Transaction*. Tipka *Delete Bank Account* služi za mogućnost brisanja bankovnog računa. Korisnik može pretražiti transakcije po vremenu izvršenja te odabrati jednu. Prilikom odabira transakcije, korisnika se vodi na zaslon koji prikazuje detalje o transakciji. Ako se korisnik odluči za stvaranje kreditne kartice za račun, odvodi ga se na zaslon za dodavanje kreditne kartice(***Slika 5.9.***).



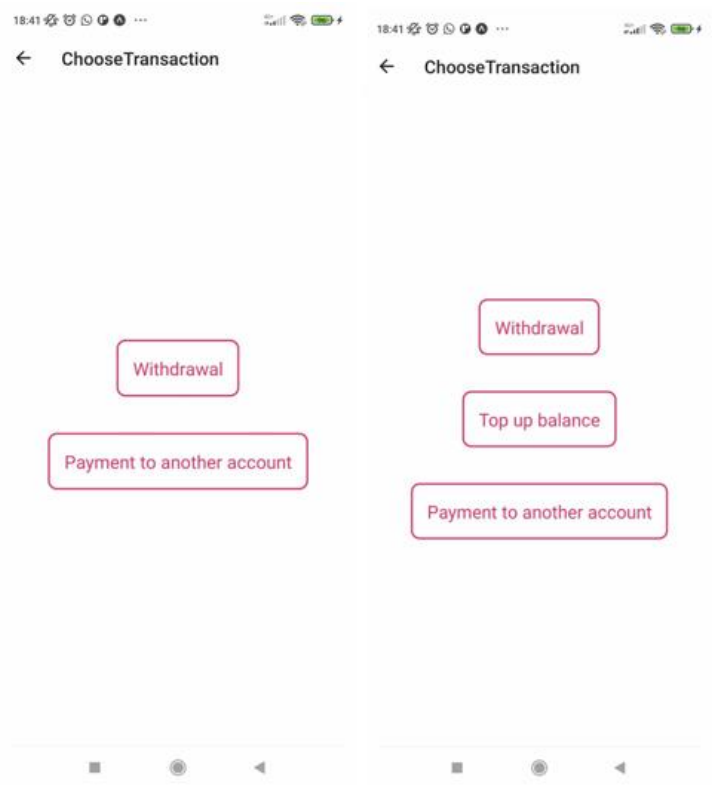
Slika 5.9. Prikaz zaslona za dodavanje kreditne kartice

Slika 5.9. prikazuje zaslon gdje se mogu vidjeti podaci o broju računa, e-mail-u klijenta te forma za stvaranje nove kreditne kartice. Kreditna kartica se dodaje na popunjavanjem polja za unos broja kartice, tipa kartice i limita kartice. U slučaju da korisnik pritisne neku od transakcija unutar zaslona bankovnog računa prikazuje se zaslon detalja transakcije (***Slika 5.10.***).



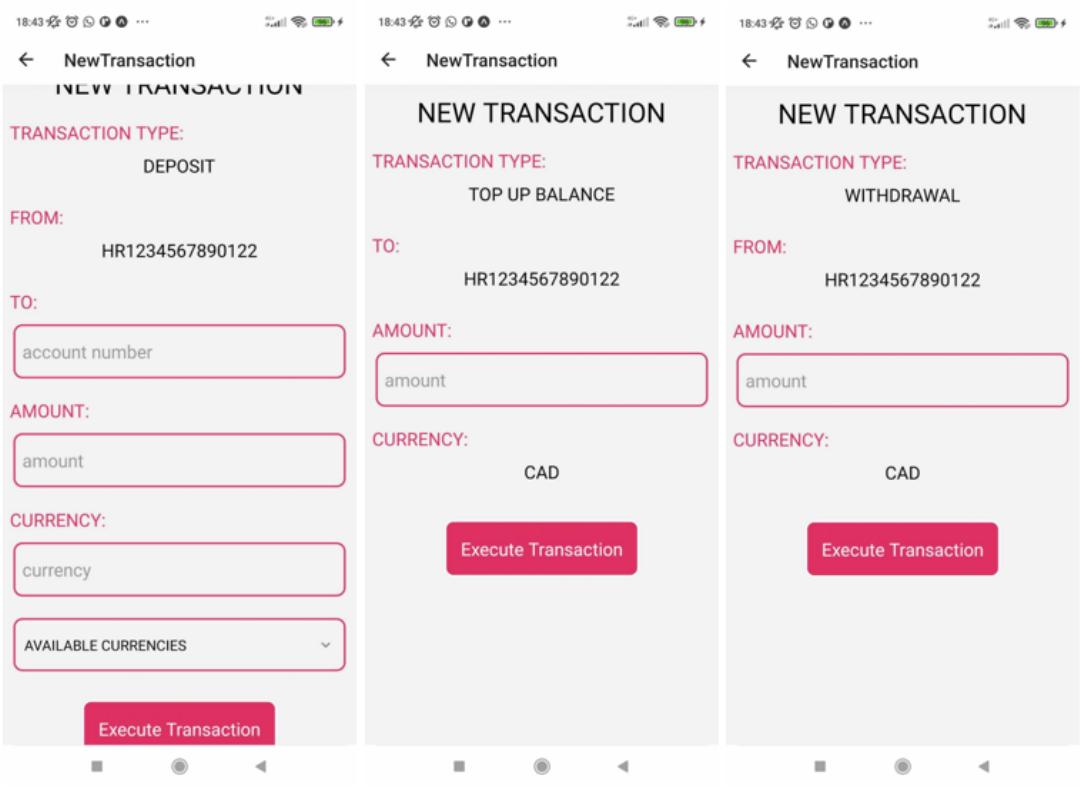
Slika 5.10. Prikazi zaslona detalja transakcije

Slika 5.10. prikazuje tri prikaza zaslona. Lijevi prikaz prikazuje uplatu na drugi račun s podacima o vremenu izvršenja transakcije, tipu transakcije i količini prenesenih sredstava zajedno sa valutom transakcije. U slučaju da je netko s drugog računa uplatio novac na trenutno prikazani, tip transakcije pisao bi se kao *Deposit from account: {account number}* gdje *{acccount number}* predstavlja broj računa s kojeg su se uplatila sredstva. Prikaz u sredini prikazuje uplatu na račun od strane bankovnog menadžera, a desni prikaz prikazuje isplatu sredstava s računa. Ako korisnik unutar zaslona bankovnog računa pritisne gumb *Make Transaction*, odvodi ga se u zaslon za odabir tipa transakcije (***Slika 5.11.***)



Slika 5.11. Prikazi zaslona za odabir tipa transakcije

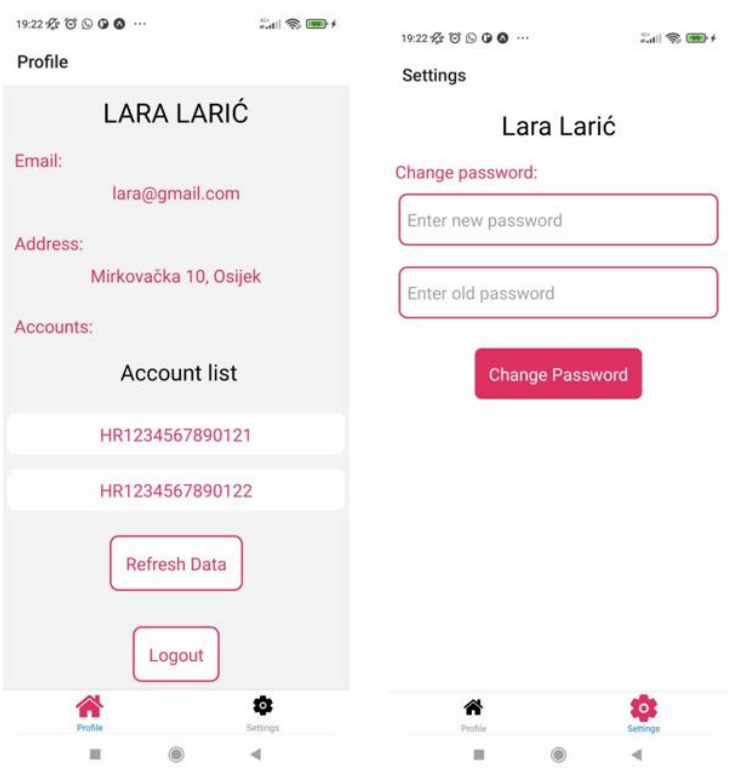
Slika 5.11. prikazuje dva prikaza zaslona. Lijevi prikaz prikazuje zaslon s mogućnostima klijenta, a desni prikazuje zaslon s mogućnostima bankovnog menadžera. Važno je uočiti da dodavanje sredstava na račun (eng. *top up balance*) može izvršiti samo bankovni menadžer u banci. Tu korisnik odabire koji tip transakcije može izvršiti. Tri tipa transakcije su isplata sredstava s računa (eng. *withdrawal*), uplata sredstava na račun i uplata sredstava na drugi račun (*eng. payment* to another account). Ukoliko je korisnik odabrao neki od tipova transakcije, odvodi ga se na zaslon nove transakcije(***Slika 5.12.***).



Slika 5.12. Prikazi zaslona nove transakcije

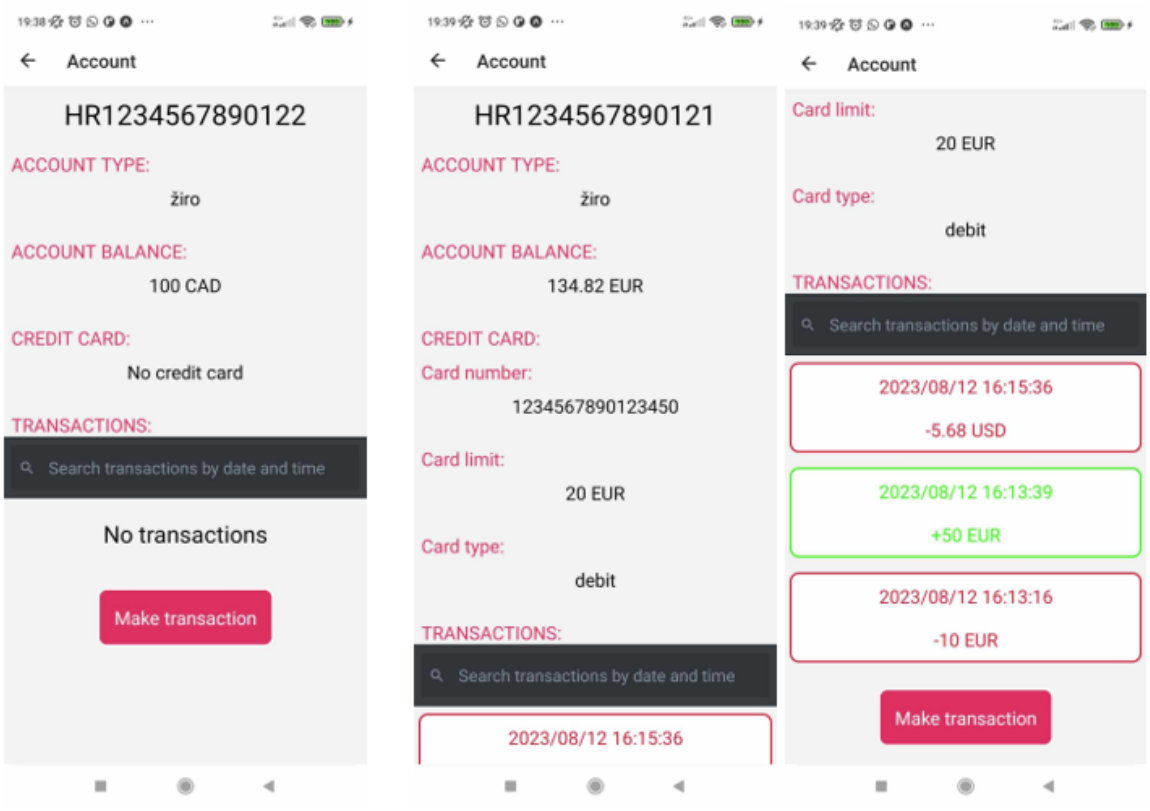
Slika 5.12. prikazuje tri prikaza zaslona. Lijevi prikaz je kada korisnik odabere uplatu na drugi račun. Tada popuni podatke među kojima je i valuta transakcije u kojoj će se transakcija izvršiti. Valutu korisnik može odabrati iz padajućeg izbornika *AVAILABLE CURRENCIES* ili može upisati samu oznaku valute ako zna. Pritiskom na gumb *Execute Transaction* izvršava transakciju. U sredini je prikaz dodavanja transakcije na račun te desno je prikaz povlačenja sredstava s računa.

U nastavku će se još kratko objasniti put klijenta kroz aplikaciju. Ako se klijent prijavi u aplikaciju, odvede ga se na zaslon profila kada je klijent prijavljen (***Slika 5.12.***).



Slika 5.13. Prikaz zaslona profila kada je prijavljen klijent (lijevo) i prikaz zaslona postavki kad je prijavljen klijent (desno)

Slika 5.13. prikazuje dva zaslona. Zaslon lijevo je zaslon profila kada je prijavljen klijent. Ovaj zaslon je prvi zaslon koji se prikaže klijentu ad se prijavi. Zaslon sadrži podatke o klijentu te gumb za osvježavanje podataka (*Refresh Data*) i odjavu klijenta (*Logout*). Zaslon desno je zaslon postavki kada je klijent prijavljen i sadrži samo formu za promjenu lozinke. Važno je primijetiti kako klijent ne može obrisati svoj profil nego samo bankovni menadžer koji je za njega zadužen. Odabere li klijent jedan od njegovih bankovnih računa, prikazuje se zaslon bankovnog računa za klijenta (***Slika 5.14.***).



Slika 5.14. Prikazi zaslona bankovnog računa za klijenta

Slika 5.14. prikazuje dva zaslona bankovnog računa za klijenta u tri prikaza. Lijevi prikaz prikazuje izgled zaslona kada bankovni račun nema kreditnu karticu niti transakcije. Važno je primijetiti da klijent nema mogućnost dodavanja kreditne kartice kao ni bankovnog računa. To su zadaće bankovnog menadžera. Prikazi u sredini i desno prikazuju zaslon kada bankovni račun ima transakcije i kreditnu karticu. Korisnik može pretraživati transakcije i provjeriti detalje transakcije kao i bankovni menadžer. Pritiskom na gumb *MakeTransaction*, odvodi ga se na odabiranje transakcije gdje može izvršiti samo uplatu na drugi račun ili isplatu sredstava. Klijent izvršava transakcije identično kao i bankovni menadžer.

6. ZAKLJUČAK

Cilj ovog diplomskog rada bio je izrada aplikacije koja bi imala svrhu nadgledavanja i upravljanja računima u banci. Aplikacija omogućava bankovnim menadžerima upravljanje svim komponentama profila klijenta kao što je brisanje i dodavanje bankovnog računa, dodavanje transakcija, dodavanje kreditnih kartica i brisanje cijelog profila klijenta. Klijentu se omogućava jednostavan pregled bankovnih računa i izvršavanje transakcija. Za realizaciju aplikacije bilo je dobro proučiti Java i React Native programske jezike.

Transakcije i bankovni računi čine srž ove aplikacije jer se nad njima odvijaju glavne funkcionalnosti aplikacije. Transakcije su bile nedvojbeno najteža logika koja se morala implementirati te je u nju uloženo i najviše vremena. Korištenje Dockera i SpringBoot Java programskog okvira je vrijedno iskustvo za daljnju karijeru kao i razvoj korisničkog sučelja uz React Native. Komunikacija između servera i klijenta aplikacije na klijentskoj strani je odrađena lagano uz pomoć *axios* biblioteke React Native-a.

Aplikacija sadrži puno DTO (eng. *Dana Transfer Object*) klasa koje su uvelike olakšale komunikaciju između klijenta i servera. SpringBoot anotacije su odradile veliku ulogu u smanjenju koda i lakšem programiranju inače složenih koncepata na serverskoj strani aplikacije. Uz pomoć Liquibase-a tablice u bazi su se stvarale jako jednostavno.

Za kraj treba reći da je ova aplikacija namijenjena upravo bankama koje bi uz pomoć nje mogle testirati svoje sustave. Aplikacija je otvorena za proširenja i daljnji razvoj te se može izgraditi iz nje aplikacija koja bi se mogla jednom koristi u banci. Ova aplikacija nudi mogućnost brzog komuniciranja preko baze podataka u stvarnom vremenu. Postoji mjesta za napredak u smislu ubrzanja stvaranja bankovnih računa i klijentskih profila. Upravo zato, ova aplikacija bi imala mjesto na tržištu ako se dorade određeni segmenti i implementira logika rada banke koja je tajna za sve banke.

LITERATURA

[1] Docker, dostupno na: https://aws.amazon.com/docker/, [pristupljeno 24. lipnja 2023.]

[2] Docker dokumentacija, dostupno na: https://docs.docker.com/get-started/overview/ , [pristupljeno 24. lipnja 2023.]

[3] Liquibase dokumentacija, dostupno na: https://docs.liquibase.com/home.html?_ga=2.169632993.1766633494.1687621950-39059226.1671438086, [pristupljeno 24. lipnja 2023.]

[4] Liquibase dokumentacija, dostupno na: https://www.liquibase.com/?_ga=2.231556447.1766633494.1687621950-39059226.1671438086, [pristupljeno 24. lipnja 2023.]

[5]: IBM, Spring Boot, dostupno na: https://www.ibm.com/topics/java-spring-boot , [pristupljeno 24. lipnja 2023.]

[6] JWT Token, dostupno na: https://jwt.io/introduction, [pristupljeno 25. lipnja 2023.]

[7] What is React Native?, dostupno na: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html, [pristupljeno 25. lipnja 2023.]

[8] React Native, dostupno na: https://reactnative.dev/ , [pristupljeno 25. lipnja 2023.]

[9] Visual Studio Code,dokumentacije, dostupno na: https://code.visualstudio.com/docs, [pristupljeno 25. lipnja 2023.]

[10] Visual Studio Code, Infoworld, dostupno na: https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html , [pristupljeno 25. lipnja 2023.]

[11] Intelij, dokumentacija, dostupno na: https://www.jetbrains.com/help/idea/discover-intellij-idea.html, [pristupljeno 25. lipnja 2023.]

[12] Intelij,TutorialsPoint, dostupno na: https://www.tutorialspoint.com/intellij_idea/intellij_idea_introduction.htm, [pristupljeno 25. lipnja 2023.]

[13] Postman, dokumentacija dostupno na: https://www.postman.com/product/what-is-postman/, [pristupljeno 25. lipnja 2023.]

[14] Exchange rates api, službena stranica dostupna na: https://exchangeratesapi.io/, [pristupljeno 9. kolovoz 2023.]

[15] Redux, službena stranica dostupna na: https://redux.js.org/, [pristupljeno 10. kolovoz 2023.]

[16] GitHub repozitorij diplomskog rada, dostupno na: https://github.com/KristijanHaubrich/bank_app, [pristupljeno 11. kolovoz 2023.]

SAŽETAK

Cilj ovog diplomskog rada bila je izrada demo aplikacije za nadgledanje i upravljanje računima u banci. Za programiranje koristio se Java i JavaScript programski jezik. U Docker spremniku napravljena je baza podataka a API za pristup bazi je napravljen pomoću *SpringBoot* Java programskog okvira. Migracije podataka su napravljene pomoću *Liquibase*-a. Korisničko sučelje napravljeno je u React Native JavaScript programskom okviru. Napravljene su i ostvarene glavne funkcionalnosti aplikacije a to su: funkcionalnost registracije i prijave u aplikaciju, funkcionalnost pregleda bankovnih računa i klijenata kao i njihovih kreditnih kartica i transakcija za bankovnog menadžera te mogućnost pregleda računa i obavljanje transakcija za klijenta. U uvodu su se navele glavne zadaće aplikacije. U poglavlju o teorijskoj podlozi kratko su se opisale sve tehnologije korištene u aplikaciji. Zatim je objašnjen način rada aplikacije te prikaz iste. Na kraju je donesen zaključak u obliku kratkog opisa rada. Aplikacija je testirana i sve funkcionalnosti rade bez ikakvih problema ili poteškoća.

Ključne riječi: banka, React Native, računi, SpringBoot, transakcije

ABSTRACT

APPLICATION FOR MONITORING AND MANAGING BANK ACCOUNTS

The goal of this thesis was to create a demo application for monitoring and managing bank accounts. Java and JavaScript programming languages were used for programming. A database was created in the Docker container, and the API for accessing the database was created using the SpringBoot Java programming framework. Data migrations were done using Liquibase. The user interface is built in the React Native JavaScript framework. The main functionalities of the application were created and implemented, namely: the functionality of registration and logging into the application, the functionality of reviewing bank accounts and clients as well as their credit cards and transactions for the bank manager, and the ability to review accounts and perform transactions for the client. The main tasks of the application were stated in the introduction. In the chapter on the theoretical background, all the technologies used in the application were briefly described. Then the way the application works and how it looks is explained. At the end, a conclusion was made in the form of a short description of the work. The application has been tested and all functionalities work without any problems or difficulties.

Keywords: accounts, bank, React Native, SpringBoot, transactions

ŽIVOTOPIS

Kristijan Haubrich rođen je 19.9.1998. godine u Vinkovcima. 2005. godine počeo je osmogodišnje osnovnoškolsko obrazovanje u školi Josipa Kozarca u Vinkovcima. Nakon osam godina upisuje Tehničku školu Ruđera Boškovića Vinkovci, smjer Tehnička gimnazija. Godine 2016. nastupa u Međužupanijskom natjecanju učenika u obrazovnom sektoru Elektrotehnika i računalstvo u disciplini Osnove elektrotehnike i mjerenja u elektrotehnici. Maturu je pisao 2017. godine te upisao Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek. 2021. godine stekao je titulu Sveučilišni prvostupnik inženjer računarstva. Odradio je jednu neobaveznu praksu u EM2 Solutions firmi te praksu u firmi Enea. Dobro se služi programskim jezicima kao Java, JavaScript, C, C++ te zna koristiti bazu podataka. Posjeduje znanje engleskog jezika za čitanje i pisanje te se bavi sportom od rane dobi. Posjeduje vozačku dozvolu B kategorije.

Kristijan Haubrich