Web aplikacija za upravljanje računima online mjenjačnica

Lazić, Srđan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni prijediplomski studij Računarstvo

WEB APLIKACIJA ZA UPRAVLJANJE RAČUNIMA ONLINE MJENJAČNICA

Završni rad

Srđan Lazić

Osijek,2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za ocjenu završnog rada na stručnom prijediplomskom studiju

Ocjena završnog rada na stručnom prijediplomskom studiju

Ime i prezime pristupnika:	Srđan Lazić
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	AR 4790, 27.07.2020.
JMBAG:	0165085010
Mentor:	doc. dr. sc. Tomislav Galba
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 1:	doc. dr. sc. Tomislav Galba
Član Povjerenstva 2:	prof. dr. sc. Tomislav Keser
Naslov završnog rada:	Web aplikacija za upravljanje računima online mjenjačnica
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Potrebno je razviti web aplikaciju baziranu na Java Spring programskom okviru koja će imati minimalno dvije korisničke role: korisnik --> mogućnost prijave/registracije, pregled stanja pojedinih kriptovaluta na različitim mjenjačnicama, uređivanje pristupnih podataka, eventualno slanje i prebacivanje novca; administrator --> pregled i uređivanje podataka o korisnicima, ograničeni uvid u korisničke račune. Uz web aplikaciju potrebno je razviti bazu podataka kao i API za pristup. Napomena: tema rezervirana za Srđan Lazić

Datum ocjene pismenog dijela završnog rada od strane mentora:	12.09.2024.
Ocjena pismenog dijela završnog rada od strane mentora:	Izvrstan (5)
Datum obrane završnog rada:	30.09.2024.
Ocjena usmenog dijela završnog rada (obrane):	Izvrstan (5)
Ukupna ocjena završnog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio stručni prijediplomski studij:	30.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	Srđan Lazić
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR 4790, 27.07.2020.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za upravljanje računima online mjenjačnica**

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

„Blockchain“ tehnologija i kriptovalute revolucioniraju bankarski sektor u sadašnjoj digitalnoj eri. Sve veća prihvatljivost i upotreba digitalnih valuta učinila je mrežne aplikacije koje olakšavaju rukovanje kriptovalutama sve važnijima. Kako bi klijenti mogli upravljati svojim portfeljem kriptovaluta, a administratorima mogućnost da paze na korisničke račune, u sklopu ovog završnog rada izrađena je mrežna aplikacija koristeći *„React“* za sučelje i *„Java Spring“* za pozadinsku aplikaciju.

Budući da korisnici često imaju račune na više burzi, upravljanje portfeljima i praćenje stanja može biti izazovno. Nadalje, budući da su financijski podaci osjetljivi, sigurnost korisničkih podataka i transakcija je ključna.

Struktura ovog završnog rada je organizirana, tako da prvo predstavi problematiku i važne teme, zatim detaljnije obrazloži zadatak i ciljeve koji se žele postići. Bitno je spomenuti trenutne praktične primjere mrežnih aplikacija u financijskom sektoru i analizi kriptovaluta. U završnom radu su obrađena bitna poglavlja, kao što je arhitektura sustava, koja opisuje klijent-poslužitelj model, glavne komponente sustava i njihov međusobni odnos. Također, spomenut je dizajn baze podataka, koji uključuje strukturu i dijagram. S obzirom na to da zadatak zahtjeva više korisničkih uloga, gdje svaka ima jasno definiran pristup, u završnom radu je objašnjen način na koji su implementirani autentifikacija i autorizacija.

Na kraju, u završnom radu je prikazan dijagram toka i način na koji komponente aplikacija međusobno komuniciraju. Prikazani su detalji razvoja, kao što su slike programskog koda, za sučelje i pozadinsku aplikaciju, u koje se ubraja i način na koji se aplikacija povezuje s *„API“*

Ovaj rad pruža sveobuhvatan prikaz procesa razvoja mrežne aplikacije za upravljanje i pregled kriptovalutama, od inicijalnog dizajna do implementacije i evaluacije postignutih rezultata kroz prikaz skica ekrana, konačan izgled ekrana te metode na koji se aplikacija testirala.

1.1 Zadatak završnog rada

Potrebno je razviti web aplikaciju baziranu na Java Spring programskom okviru koja će imati minimalno dvije korisničke role:

- Korisnik - mogućnost prijave/registracije, pregled stanja pojedinih kriptovaluta na različitim mjenjačnicama, uređivanje pristupnih podataka, eventualno slanje i prebacivanje novca.
- Administrator - pregled i uređivanje podataka o korisnicima, ograničeni uvid u korisničke račune.

Uz web aplikaciju potrebno je razviti bazu podataka kao i API za pristup.

2. PREGLED PODRUČJA TEME

Glavni cilj ovog projekta je koristiti *„Java Spring“* i *„React“* za stvaranje mrežne aplikacije koja upravlja korisničkim računima i analizira cijene kriptovaluta na više burzi. U ovom poglavlju objašnjene su trenutne praktične primjene u prostoru web aplikacija, s fokusom na financijske aplikacije i analizu kriptovaluta.

„Java Spring“ je okvir, prema [1], koji se često koristi u razvoju aplikacija na poduzetnoj razini zbog svojih mogućnosti, skalabilnosti i jednostavnosti integracije s različitim uslugama. *„React“* je popularna *„JavaScript“* biblioteka, prema [2], koja se koristi za izradu korisničkog sučelja. Također je odabran zbog svoje učinkovitosti u prikazivanju dinamičkih podataka i osiguravanja glatkog korisničkog iskustva.

Da bi se održala sigurnost i integritet podataka u financijskim sustavima, definirane su uloge poput administratora i korisnika. Najčešće tehnike kontrole pristupa, prema [1, 3], su: unos korisničkih podataka, s pomoću kojih aplikacija generira *„JWT“* (*„JSON web token“*), ili *„OAuth2“*, servis treće strane koji generira *„token“* za pristup umjesto aplikacije.

Proces praćenja nekoliko kriptovaluta u stvarnom vremenu na brojnim burzama poznat je kao analiza kriptovaluta. S pomoću *„API“* koje pružaju burze, kao što su *„Binance“*, *„Coinbase“* i *„Kraken“*, mogu se dobiti podaci u stvarnom vremenu. Aplikacijom, kao što je ova, korisnici dobijaju potpunu sliku svih sredstava na jednom mjestu integracijom navedenih *„API“* u jedan, što poboljšava korisničko iskustvo i nadzor.

Platforma *„TradingView“*, prema [4], nudi alate za izradu grafikona i širokim značajkama analize tržišta. Korisnicima omogućuje stvaranje personaliziranih pokazatelja i strategija, te daje pristup podacima u stvarnom vremenu o nizu financijskih instrumenata, uključujući kriptovalute.

Aplikacija za navedeni završni omogućava pohranu podataka specifičnih za korisnika i pruža sigurne kontrole pristupa temeljene na ulogama. Iako je *„TradingView“* platforma mnogo učinkovitiji alat za analizu tržišta, ova aplikacija ispunjava zahtjeve administratora da prati korisničke podatke i korisnika koji upravljaju portfeljima. U usporedbi s *„TradingView“* platformom, ova aplikacija je koncentrirana na analizu isključivo kriptovaluta.

3. DIZAJN I ARHITEKTURA SUSTAVA

Planiranje i organiziranje elemenata *„programskog koda“* za postizanje ciljeva projekta, naziva se dizajn i arhitektura sustava. U ovom poglavlju obrađene su teme:

- Arhitektura sustava:
    a) Model: Postoje dvije primarne komponente aplikacije, klijent i poslužitelj. Dok poslužitelj obrađuje zahtjeve, održava podatke i provodi poslovnu logiku, klijent je zadužen za korisničko sučelje i interakciju s korisnikom.
    b) Komponente sustava: Navodi bitne dijelove modela kao što su razredi, metode, objekti.

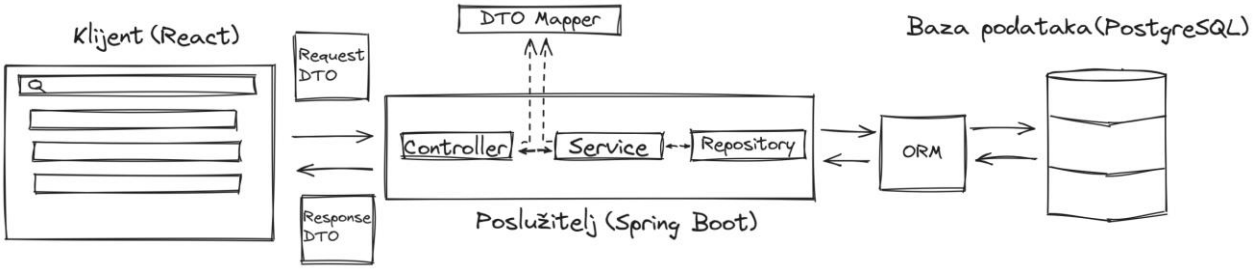
- Dizajn baze podataka:
    a) ER dijagram: Prikaz *„entiteta“* (tablica) u bazi podataka i njihovih međusobnih odnosa.
    b) Struktura tablica: Detaljan opis svake tablice, uključujući stupce, tipove podataka, primarne i strane ključeve.

- Sigurnosni aspekti:
    a) Autentifikacija: Proces provjere identiteta korisnika. Implementacija putem *„JWT“* (*„JSON Web Token“*) koji omogućuje sigurno prijavljivanje korisnika.
    b) Autorizacija: Kontrola pristupa različitim dijelovima aplikacije na temelju korisničkih uloga.
- Dijagram toka:
    a) Dijagram toka za glavne funkcionalnosti: Prikazuje korake koje sustav poduzima za izvršavanje ključnih funkcionalnosti.
    b) Dijagram sekvenci: Prikaz redoslijeda interakcija između različitih komponenti sustava.

3.1 Arhitektura sustava

Model klijent-poslužitelj omogućuje učinkovitu podjelu dužnosti između strane klijenta i strane poslužitelja. Model klijent-poslužitelj temelj je arhitekture za navedenu mrežnu aplikaciju. Ovim modelom omogućeni su skalabilnost, prilagodljivost i laka održivost aplikacije.

Prilikom izrade mrežne aplikacije, implementirana je MVC arhitektura koja je ključna za strukturirani razvoj i održavanje složenih aplikacija. MVC arhitektura omogućuje jasnu podjelu odgovornosti među komponentama aplikacije, što doprinosi modularnosti, lakoći testiranja i održavanja koda.



Slika 3.1.1. – Skema arhitekture navedenog sustava

Klijent postavlja upite poslužitelju preko *„HTTP“* protokola, tako što šalje objekt nazvan *„Request/Response DTO“* (engl. *„Data Transfer Object“* – Objekt za prijenos podataka), a dobivene odgovore prikazuje korisnicima. (Slika 3.1.1)

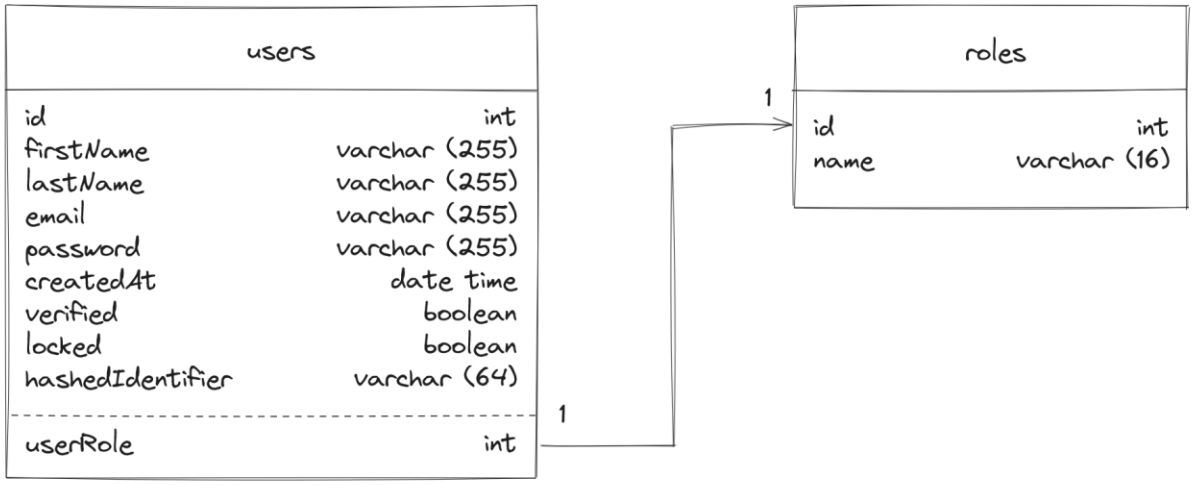
Prema slici 3.1.1, kada upit dođe do poslužitelja, upravljač preusmjerava zahtjev na sloj usluge od kojeg se očekuje da izvrši poslovnu logiku. Poslovna logika diktira kako se postupa s podatcima i kako se oni transformiraju unutar aplikacije. Takozvani sloj usluge djeluje kao most između upravljača i *„repozitorija“*.

U navedenoj aplikaciji, sloj *„repozitorija“* djeluje kao most između baze podataka i aplikacije. Zadužen je za nadzor postojanosti podataka, što uključuje pristup bazi podataka, pohranjivanje, dohvaćanje i manipulaciju. Repozitorij koristi *„Spring Data JPA“* projekt,.prema [1, 5].

U konačnici, aplikacija koristi *„PostgreSQL“* kao trajno spremište i *„H2“* spremište u memoriji.

3.2 Dizajn baze podataka

Dizajn baze podataka je ključna komponenta procesa razvoja svake aplikacije, zato što jamči odgovarajuću pohranu, organizaciju i čitanje. Baza podataka za navedenu aplikaciju koristi značajke koje omogućavaju upravljanje korisnicima i njihovim pravima pristupa. Podatci se čuvaju u relacijskoj bazi podataka, *„PostgreSQL“*. Korisnici i uloge dvije su primarne tablice u navedenoj bazi podataka.



Slika 3.2.1 – ER dijagram baze podataka

„Entiteti“ (tablice) u bazi podataka i njihove veze prikazani su *„ER“* dijagramom (Slika 3.2.1). Korisnici i uloge, dva su *„entiteta“* u *„ER“* dijagramu i povezani su jedan s drugim. *„Entitet“* korisnici prikazuje korisnike aplikacije. *„Entitet“* uloge prikazuje različite uloge, koje netko može preuzeti. Strani ključ *„userRole“* u tablici korisnici, koji se odnosi na *„ID“* primarnog ključa iz tablice uloge, definira odnos između *„entiteta“* korisnici i uloge. Kao rezultat toga, svaka uloga može pripadati jednom korisniku, a svaki korisnik može imati jednu ulogu.

Detaljan opis svake tablice, uključujući stupce, tipove podataka, primarne i strane ključeve prikazan je u nastavku, prema [6].

Prema slici 3.2.2, tablica korisnici pohranjuje informacije o korisnicima aplikacije i sadrži sljedeće stupce:

- Id (int) – Primarni ključ koji jedinstveno identificira svakog korisnika
- firstName (varchar 255) – Ime korisnika
- lastName (varchar 255) – Prezime korisnika
- email (varchar 255) – *„Email“* adresa korisnika koja također služi kao jedinstveni identifikator za prijavu
- password (varchar 255) – Kriptirana lozinka korisnika za sigurnu autentifikaciju
- createdAt (datetime) – Datum i vrijeme kreiranja računa
- verified (boolean) – Zastavica koja označava je li korisnik potvrdio svoju email adresu
- locked (boolean) – Zastavica koja označava je li korisnik blokiran od strane administratora
- hashedIdentifier (varchar 255) – Kriptirani identifikator s pomoću kojeg korisnik potvrđuje svoju email adresu
- userRole (int) – strani ključ koji referencira id iz tablice uloge

```sql
CREATE TABLE users (
    id INT PRIMARY KEY,
    firstName VARCHAR(255) NOT NULL,
    lastName VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    createdAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    verified BOOLEAN NOT NULL DEFAULT FALSE,
    locked BOOLEAN NOT NULL DEFAULT FALSE,
    hashIdentifier VARCHAR(64),
    userRole INT,
);
```

Slika 3.2.2 – SQL kod za stvaranje tablice 'korisnici' u bazi podataka

Tablica uloge pohranjuje informacije o različitim ulogama koje korisnici mogu imati i sadrži sljedeće stupce, (Slika 3.2.3):

- Id (int) – Primarni ključ koji jedinstveno identificira svaku ulogu
- Name (varchar 16) – Naziv uloge (npr. „USER“, „ADMIN“)

```
CREATE TABLE roles (
    id INT PRIMARY KEY,
    name VARCHAR(16) NOT NULL
);
```

Slika 3.2.3 – SQL kod za stvaranje tablice 'uloge' u bazi podataka

Na kraju, potrebno je napraviti vezu između ove dvije tablice, a ostvarena je preko stranog ključa 'userRole' u tablici korisnici koji referencira primarni ključ id u tablici uloge, (Slika 3.2.4):

```
ALTER TABLE users ADD CONSTRAINT fk_user_role
FOREIGN KEY (userRole)
REFERENCES roles(id);
```

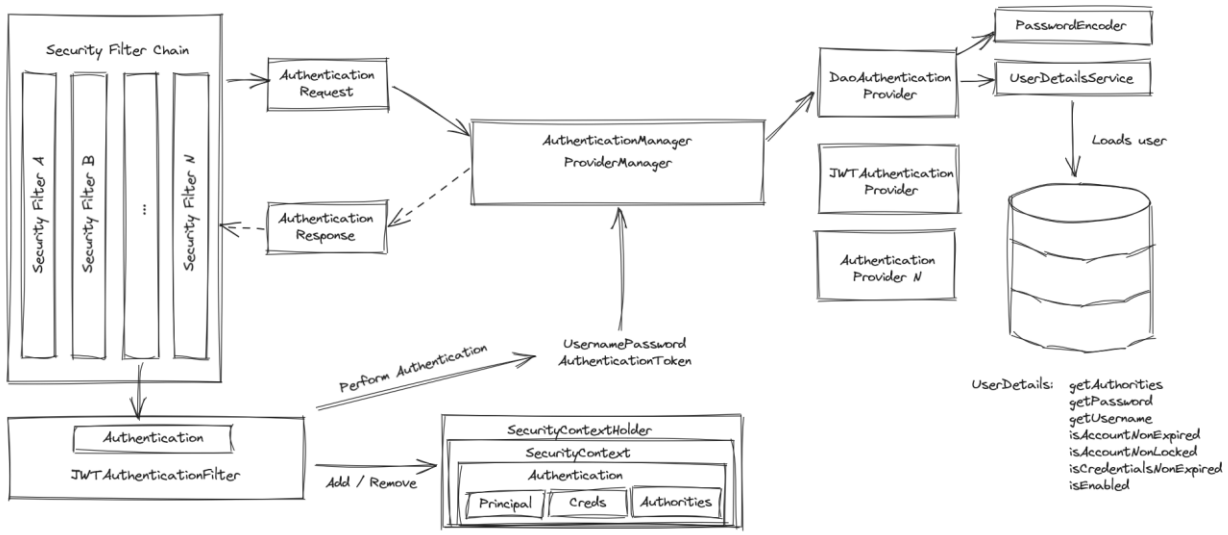
Slika 3.2.4 – SQL kod za stvaranje veze između dva entiteta 'users' i 'roles'

Kako bi odgovarajuće upravljanje i pohrana podataka u web aplikaciji bila zajamčena, dizajn baze podataka je ključan. Ovim postupkom omogućeno je učinkovito i sigurno pohranjivanje korisničkih podataka i uloga korištenjem dobro definirane strukture tablice i interakcije između njih.

Ovaj dizajn baze podataka ispunjava zahtjeve četvrte normalne forme (4NF). Obje tablice su već u 3NF i nemaju višeznačne ovisnosti, čime zadovoljavaju uvjete 4NF.

3.3 Sigurnosni aspekti

U navedenoj aplikaciji implementiran je *„Spring Security 5“*, prema [1, 3], moćan, fleksibilan sigurnosni okvir za zaštitu *„Java“* aplikacija. Ovaj okvir omogućava različite sigurnosne značajke kao što su autentifikacija, autorizacija, zaštita od *„CSRF“* napada, upravljanje sesijama i sl.



Slika 3.3.1 – Arhitektura autentifikacije i autorizacije

Prema slici 3.3.1 i prema [3], *„SecurityFilterChain“* je komponenta koja određuje način i redoslijed na koji se filtriraju *„HTTP“* zahtjevi. Pošto aplikacija koristi *„JWT“* kao primarni način za sigurnu autentifikaciju korisnika, definirana je instanca klase *„JWTAuthenticationFilter“*. Takva klasa nudi gotove metode koje izdvajaju *„token“* iz zaglavlja *„HTTP“* zahtjeva, *„validira“ „token“*, te izdvaja korisničke informacije kao što su korisničko ime i uloge.

„SecurityContext“ je centralno mjesto gdje *„Spring Security“* pohranjuje informacije o trenutno autentificiranoj osobi. Kada *„JWTAuthenticationFilter“* propusti zahtjev korisnika, *„SecurityContext“* pohranjuje *„Authentication“* objekt koji sadrži informacije o korisniku uključujući korisničko ime, lozinku i uloge.

„AuthenticationManager“ je *„API“* koji definira način na koji aplikacija vrši autentifikaciju, u ovom slučaju *„JWTAuthenticationFilter“*, a *„ProviderManager“* je najčešće korištena implementacija *„AuthenticationManager“*-a.

„ProviderManager“ je, kako naziv na engleskom jeziku kaže, upravitelj davateljima usluga, a davatelj usluga definira način na koji će „*SecurityFilter“* dobiti informacije o korisniku.

U ovoj aplikaciji korišten je *„DaoAuthenticationProvider“* kao davatelj usluga, koji označava da će se korisnik dohvaćati iz trajne pohrane, kao što je to *„PostgreSQL“* baza podataka, koristeći *„DAO“* (engl. „*Data Access Object“* – Objekt pristupa podatcima). *„DaoAuthenticationProvider“* implementira *„UserDetailsService“*, koji predstavlja skup metoda, koje generiraju SQL naredbe specifično za dohvaćanje korisnika iz baze podataka.

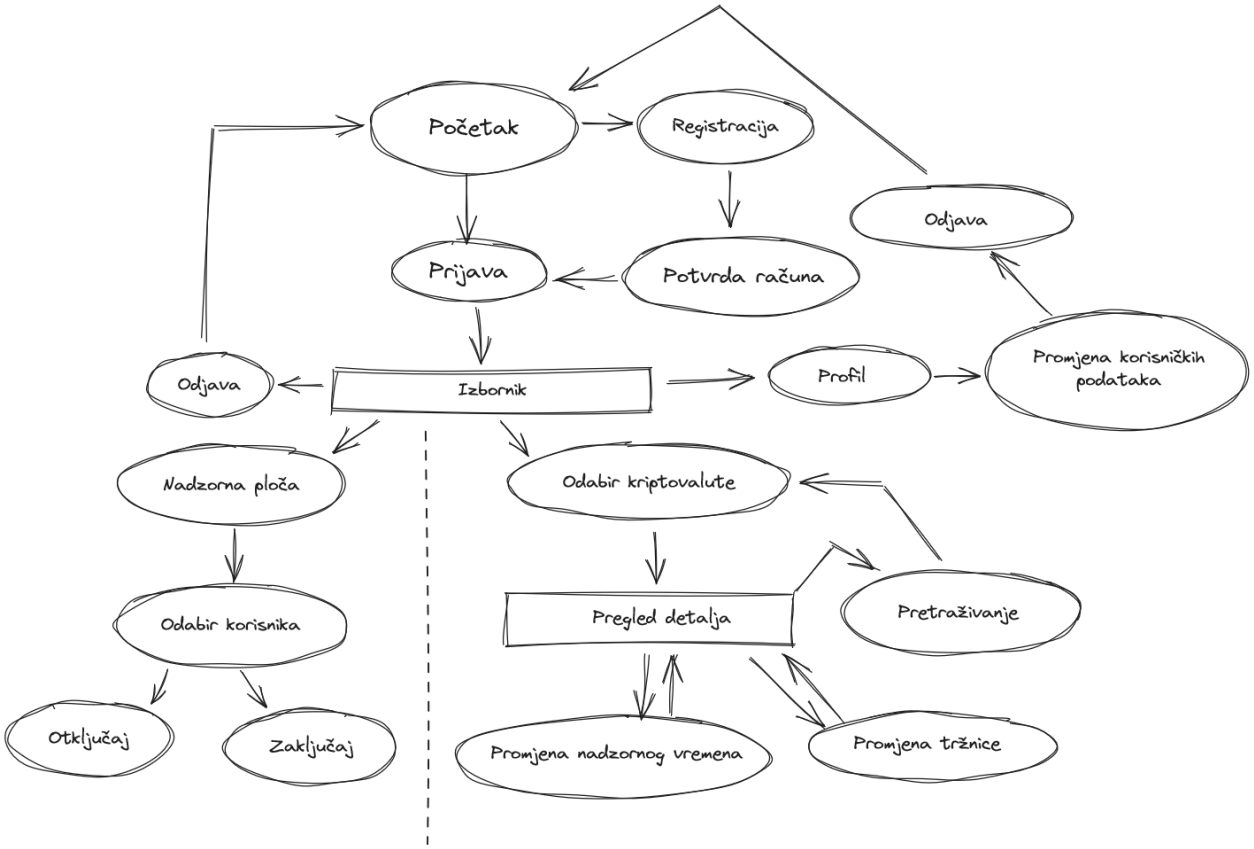
Na kraju, kada su podatci o korisniku dohvaćeni iz trajne pohrane, *„JWTAuthenticationFilter“* korištenjem svojih skupa metoda uspoređuje podatke iz trajne pohrane s podatcima izvučenih iz *„HTTP“* zahtjeva. Ako se podatci podudaraju, filtar potvrđuje identitet korisnika i omogućuje pristup sustavu. U suprotnom slučaju, ako podatci u *„HTTP“* zahtjevu ne odgovaraju niti jednom zapisu u bazi podataka, sustav vraća statusni *„kod“* 401 – Neovlašten. Takav statusni *„kod“* označava da, korisnik koji pokušava pristupiti sustavu nije registriran u sustav, pristupni podatci sustavu nisu ispravni ili je isteklo vrijeme trajanja *„JWT“ „tokena“*.

Prilikom registracije korisnika u sustav, bitno je voditi računa o njegovoj privatnosti i sigurnosti. Iz istog razloga koristi se *„PasswordEncoder“*. Lozinka za pristup sustavu ne smije biti poznata nikome osim korisniku koji pristupa sustavu, a za to se koriste različite tehnike jednosmjernog kriptiranja, gdje jednom kriptirana poruka, više se ne može dekriptirati.

Ova aplikacija koristi *„BCryptPasswordEncoder“* implementaciju *„PasswordEncoder“* sučelja. Kada korisnik pošalje pristupne podatke, specifično ime, prezime, korisničko ime i lozinku, *„BCryptPassowordEncoder“* prima lozinku, jednosmjerno se kriptira i tako kriptirana se sprema u trajnu pohranu. Idućeg puta, kada se tako registrirani korisnik prijavljuje, iz prijavnog zahtjeva *„BCryptPasswordEncoder“* prima lozinku, kriptira ju i u idućem koraku *„JWTAuthenticationFilter“* uspoređuje kriptiranu lozinku iz zahtjeva i kriptiranu lozinku dohvaćenu iz trajne pohrane. Ako se ove dvije lozinke podudaraju i ako je korisnik s navedenim korisničkim imenom registriran u trajnoj pohrani, identitet korisnika je potvrđen i dobija pristup sustavu.

3.4 Dijagram toka

Dijagram toka, grafički je prikaz izbora i radnji potrebnih za dovršenje zadatka ili procesa. Sastoji se od mnogih simbola koji označavaju različite vrste operacija ili koraka, spojenih strelicama koje pokazuju u kojem smjeru proces teče.



Slika 3.4.1 – Dijagram toka glavnih funkcionalnosti

Prema slici 3.4.1, kada korisnik dođe na početnu stranicu, ponuđena su mu dva izbora, prijava ili registracija ako već nema napravljen račun za aplikaciju. Ovaj korak se ne može zaobići i bez prijave korisnik ne može imati pristup sustavu.

Odabirom na registraciju, korisnik unosi podatke: ime, prezime, korisničko ime, lozinku. Nakon prijave preusmjeren je na zaslon kojim dobija poruku da je potrebno potvrditi korisnički račun.

U navedenoj aplikaciji je implementirana značajka slanja *„e-pošte“* u kojoj se nalazi kriptirani identifikator s pomoću kojeg korisnik potvrđuje korisnički račun.

Kada korisnik u svojem spremniku *„e-pošte“* potvrdi korisnički račun, tada putem zaslona za prijavu dobija pristup sustavu. Prijavom, korisnik ima različite izbore. Osnovni izbor je odjava, kojom se iz sesije briše *„JWT“*, korisnika se preusmjerava na početni zaslon, a bez navedenog *„tokena“* više nema pristup sustavu dok se ponovno ne prijavi.

Pošto aplikacija nudi značajke za različite uloge korisnika, ako je on administrator, ima pravo pristupa na zaslon nadzorne ploče. Na ovom zaslonu, administrator ima pravo na uvid u listu korisnika koji su registrirani u aplikaciju, te pristup restrikciji za odabranog korisnika.

Osim nadzorne ploče, ostali korisnici koji su prijavljeni u sustav imaju mogućnost pristupa aplikaciji: analiza kriptovaluta. Nakon prijave, korisnik na početnom zaslonu ima mogućnost da odabere jednu od pet poznatih kriptovaluta za koju želi pregledati detalje. Odabirom, korisnika se preusmjerava na zaslon za pregled detalja gdje je prikazan grafikon. Iduće opcije za korisnika su da mijenja interval grafikona, kojom korisnik dobija ili sklanja detalje na grafikonu. S druge strane, osnovna značajka aplikacije je pregled stanja iste kriptovalute na različitim mjenjačnicama. Zato postoji izbornik gdje korisnik za odabranu kriptovalutu bira iduću mjenjačnicu s koje želi napraviti pregled detalja. Ovim odabirom, korisnik ostaje na zaslonu za pregled detalja, a grafikon mijenja svoje stanje. Preostala značajka na zaslonu za pregled detalja je pretraživanje svih kriptovaluta. Odabirom na opciju pretraživanja, korisniku se pojavljuje prozor gdje je omogućen unos teksta. Unosom teksta korisnik filtrira sve kriptovalute za sve mjenjačnice. Odabirom pretražene kriptovalute, prozor za pretraživanje se sklanja, a korisnik i dalje ostaje na zaslonu za prikaz detalja. S ovim značajkama zaključen je dio dijagrama toka za pregled i analizu kriptovaluta.

Zadnji dio dijagrama toka korisniku omogućava upravljanje nad profilom. Odabirom na profil, na izborniku, korisnika se preusmjerava na drugačiji zaslon. Na ovom zaslonu korisnik ima mogućnost da promjeni pristupne podatke kao što su korisničko ime i lozinka. Kada korisnik unese nove podatke, dobija *„e-poštu“* za potvrdu izmjene podataka, korisnika se odjavljuje iz sustava jer je potrebno generiranje novog *„tokena“*, a korisnik se preusmjerava na početni zaslonu bez prava pristupa sustavu.

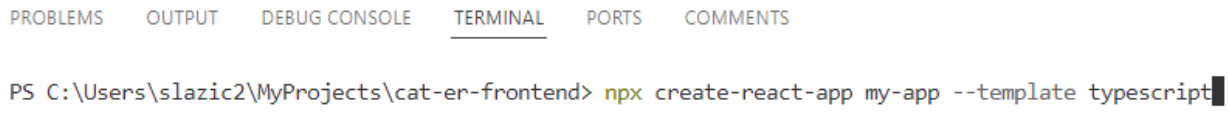
4. IMPLEMENTACIJA SUSTAVA

Implementacija sustava posebnu pozornost pridaje na strukturu datoteke i važne segmente *„koda“*. Kako bi se postigla funkcionalnost i skalabilnost mrežne aplikacije, potrebno je uključiti niz kritičnih faza u implementaciji sustava. Da način bude razumljiviji, implementacija sustava razlikuje strukturu datoteka za sučelje i poslužiteljsku aplikaciju. Dok je poslužitelj aplikacija stvorena korištenjem *„Java Spring“* okvira, sučelje je razvijeno korištenjem *„React“* biblioteke.

Implementacija sustava također prikazuje *„kod“* koji izvršavaju temeljne funkcije. To uključuje upravljanje korisničkim podatcima, dohvaćanje podataka iz vanjskih *„API“*, prikaz podataka na korisničkom sučelju i autentifikaciju korisnika.*“Kod“* je detaljno opisan. Ova metoda nudi temeljito razumijevanje unutarnjeg rada aplikacije.

Na kraju, implementacija sustava demonstrira veze, dijeljenje podataka i komunikaciju između dva dijela navedene aplikacije: sučelja i poslužitelja. Prikazuje primjere zahtjeva za *„REST API“* koji se šalju i odgovori vraćeni od poslužitelja. Povezivanje ova dva dijela zajedno jamči da sustav funkcionira kao cjelina i omogućuje korisnicima interakciju na jednostavan i učinkovit način.

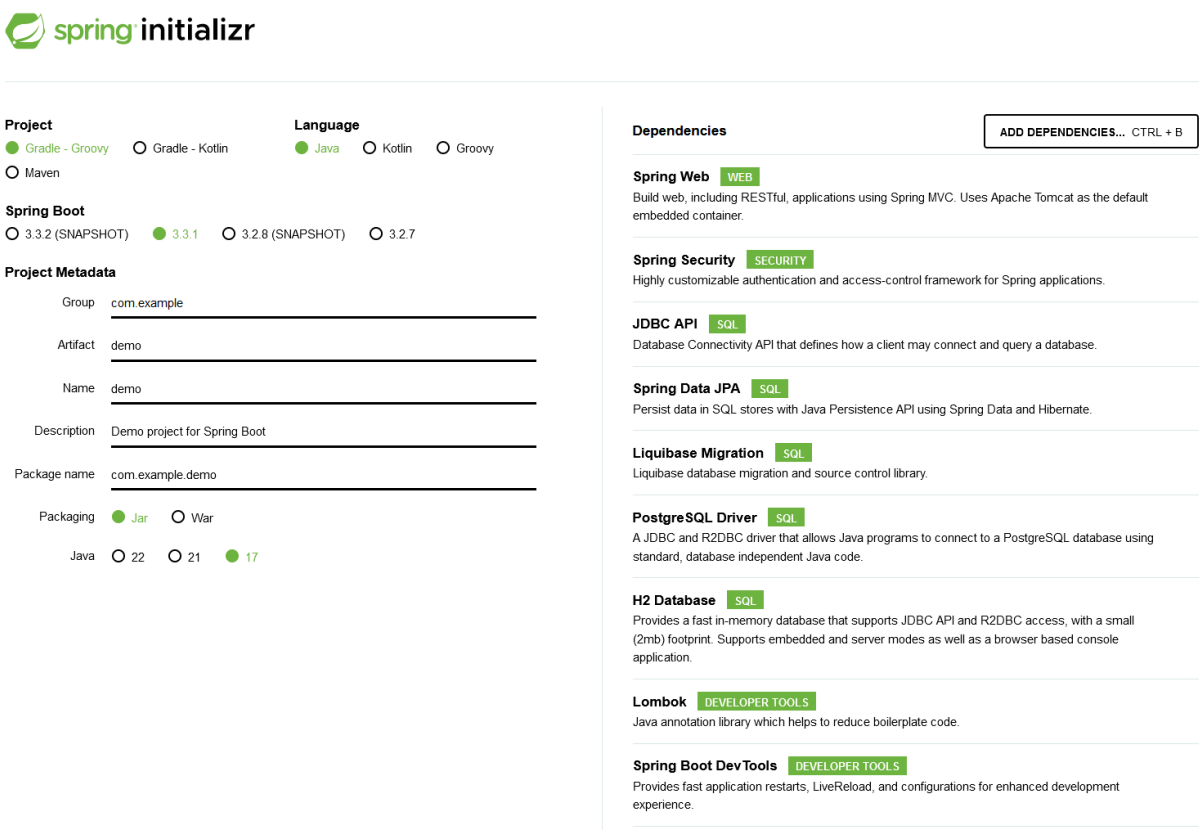
Cilj ovog temeljitog pristupa je dati sveobuhvatnu sliku o izvođenju sustava i temeljito razumijevanje svih njegovih bitnih dijelova.



Slika 4.1 – „Node.js“ naredba za kreiranje „React“ projekta

Za početak razvijanja *„React“* aplikacije, potrebno je izvršiti naredbu prema [7, 8]. (Slika 4.1). Naredba je definirana u *„Node.js“*, prema [7] okruženju a izvršava se s pomoću *„npx“* (engl. „*Node Package eXecute“*). *„React“* aplikacija može se razvijati u *„VS Code“* razvojnom okruženju.

Postupak kreiranja i pokretanja *„Java Spring“* aplikacije je drugačiji. Za kreiranje *„Spring“* aplikacije, najjednostavniji način je koristiti *„Spring Initializr“* prema [9], prikazano na slici 4.2.



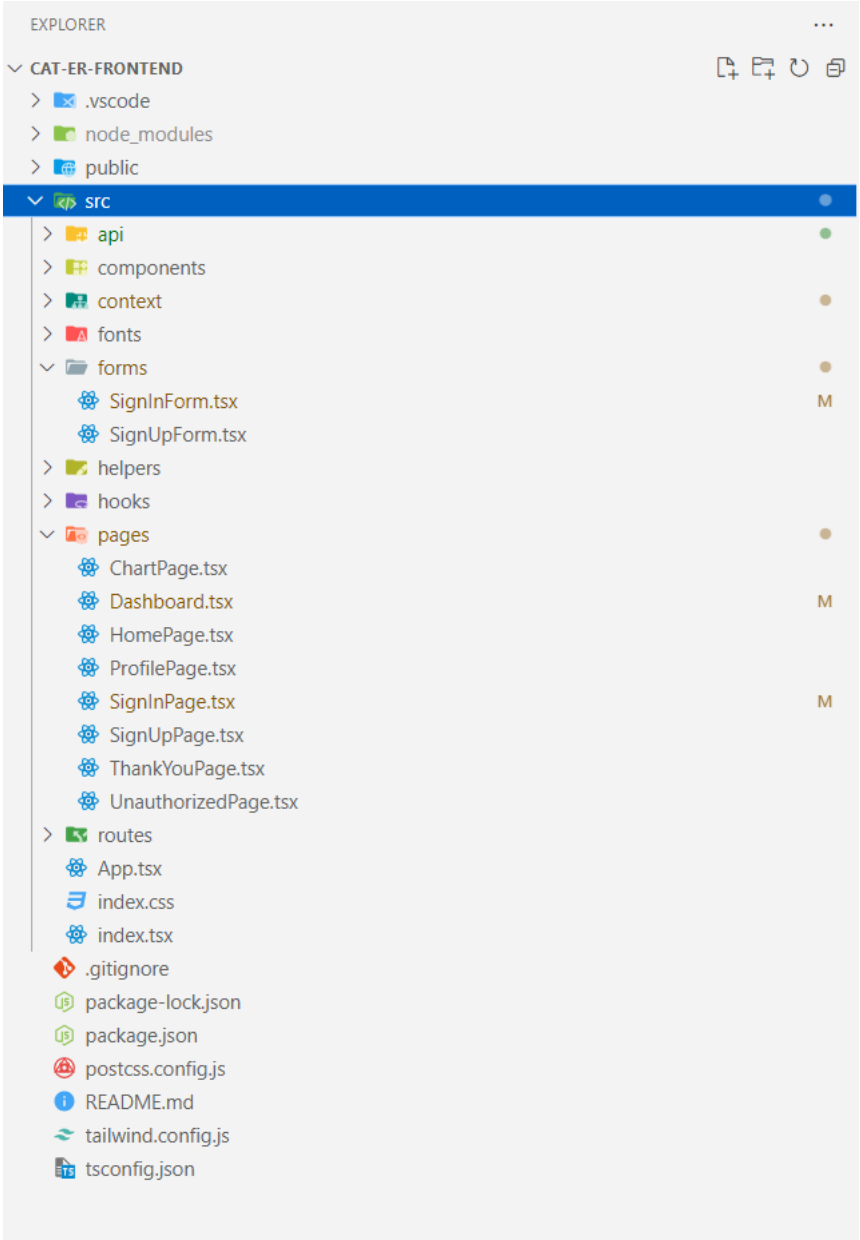
Slika 4.2 – Web aplikacija za kreiranje „Java Spring“ projekta

Pristupanjem na stranicu, korisniku je ponuđen upitnik koji treba popuniti za kreiranje projekta. Za potrebe izrade navedene aplikacije, odabrane su opcije *„Maven“* projekt u *„Java“* programskom jeziku , a *„spring boot“* verzija 3.0.3.

Osim toga korisnik unosi naziv projekta i proizvoljni opis. Za potrebe izrade navedene aplikacije, odabrane su opcije *„Jar“* način pakiranja, a koristi se verzija 17 *„Java“* razvojnog kompleta koji također uključuje okruženje za izvođenje *„Java“* aplikacije.

U konačnici korisnik treba da odabere vanjske pakete koji omogućavaju brže razvijanje, tako da se usredotoči na razvijanje same aplikacije. Odabrani paketi za razvoj navedene aplikacije prikazani su na slici 4.2

4.1 Razvoj klijent aplikacije



Slika 4.1.1 – Struktura „React“ aplikacije

Kada se naredba sa slike 4.1 izvrši, rezultat je struktura aplikacije kao što je prikazano na slici 4.1.1. Osnovni dijelovi strukture su:

- *„node_modules“* - direktorij u kojem su pohranjene predefinirane komponente i metode instaliranih paketa za ubrzano razvijanje aplikacije
- *„public“* – direktorij u kojem su pohranjene statične datoteke, odnosno sredstva koje aplikacija koristi
- *„src“* – direktorij u kojem je pohranjen izvorni *„kod“* aplikacije
- Konfiguracijske datoteke – dio korijena direktorija koji definira način na koji se aplikacija izvršava, organizira pakete i interpretira vanjske naredbe koje su naknadno instalirane kroz pakete.

Posebna pažnja se pridodaje *„src“* direktoriju. U njemu je sadržan izvorni *„kod“*. Struktura i objašnjenje iste, nalazi se u dokumentima, prema [2].

Cilj strukture je da programski *„kod“* drži organiziranim, kako za pojedinca tako i za tim koji razvija aplikaciju. Olakšava prepoznavanje različitih komponenti a grupira ih na različite načine , bilo to po tipu datoteke, tipu komponente ili funkciji komponente.

U svrhu izrade navedene aplikacije, primarno je korišten pristup podjele datoteka prema tipu komponente [2], što znači:

- „api“ – direktorij koji sprema funkcije za komunikaciju s poslužiteljem
- „components“ – direktorij koji sprema funkcijske komponente (dijelove stranica) koje se upotrebljavaju kroz cijelu aplikaciju (navigacijska traka, grafikon, i sl.) na stranicama
- „context“ – direktorij koji sprema definiciju poslužitelja za cijelu aplikaciju
- „forms“ – direktorij koji sprema funkcijske komponente koje opisuju upitnike
- „helpers“ – direktorij koji sprema uslužne funkcije (formatiranje datuma i sl.)
- „hooks“ – direktorij koji sprema funkcije za upravljanje stanjima u aplikaciji
- „pages“ – direktorij koji sprema funkcijske komponente za prikaz stranice

„indeks.tsx“ je datoteka, kao što je prikazano na slici 4.1.2, u kojoj započinje prikaz čitave aplikacije. U navedenoj datoteci, poziva se *„createRoot“* funkcija gdje *„React“* po standardu kreira korijenski element dokumenta, a zatim *„render“* funkcija, kojoj se predaju elementi, koji se u zadanom trenutku prikazuju na ekran u korijenskom elementu dokumenta.

```
ReactDOM.createRoot(document.getElementById("root") as HTMLElement).render(
  <React.StrictMode>
    <AuthProvider>
      <App />
    </AuthProvider>
  </React.StrictMode>
);
```

Slika 4.1.2 – Datoteka „indeks.tsx“ za prikaz aplikacije

```
function App() {
  return (
    <BrowserRouter>
      <Routes>
        {/* PUBLIC ROUTES */}
        <Route path="/sign_up" element={<SignUpPage />} />
        <Route path="/sign_in" element={<SignInPage />} />
        <Route path="/thank_you" element={<ThankYouPage />} />
        <Route path="/unauthorized" element={<UnauthorizedPage />} />
        <Route path="/" element={<HomePage />} />

        {/* PROTECTED ROUTES */}
        <Route element={<ProtectedRoute allowedRoles={["ADMIN", "USER"]} />}>
          <Route path="/profile" element={<ProfilePage />} />
          <Route path="/chart" element={<ChartPage />} />
        </Route>

        {/* ADMIN ONLY */}
        <Route element={<ProtectedRoute allowedRoles={["ADMIN"]} />}>
          <Route path="/dashboard" element={<Dashboard />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

Slika 4.1.3 – Datoteka „App.tsx“ usmjerivač

„App.tsx“ predstavlja usmjerivač na različite komponente u aplikaciji. (Slika 4.1.3). Ovisno o *„url“* putanji (engl. *„Uniform Resource Locator“*), *“BrowserRouter“* vraća komponentu koja treba da se prikaže na ekran kroz korijenski dokument.

```tsx
const AuthProvider: FC<{ children: ReactNode }> = ({ children }) => {
  const [token, setToken] = useState<string | undefined | null>(sessionStorage.getItem("token"));

  const saveToken = (newState?: Jwt): void => {
    if (newState) {
      sessionStorage.setItem("token", newState);
      setToken(newState);
    } else {
      sessionStorage.removeItem("token");
      setToken(undefined);
    }
  };

  return <AuthContext.Provider value={{ token: token, saveToken: saveToken }}>{children}</AuthContext.Provider>;
};
```

Slika 4.1.4 – Datoteka „AuthProvider.tsx“ koja upravlja stanjem autentifikacije

Na kraju, pošto aplikacija zahtjeva autentifikaciju, potrebno je implementirati dostavljača stanja autentifikacije, tako da stanje autentifikacije bude vidljivo kroz cijelu aplikaciju. Zbog toga je implementirana *„AuthProvider“* komponenta, kojom je omotana *„App“* komponenta. *„AuthProvider“* definira stanje *„tokena“* te funkciju koja sprema *„token“* u stanje aplikacije ali i u stanje mrežnog preglednika koje korisnik koristi, a to je kroz sesiju. Na ovaj način, u bilo kojem trenutku razvijanja i u bilo kojoj komponenti, dovoljno je pozvati *„useContext“ „React“* funkciju i specificirati da je potrebno pozvati upravo ovaj *„AuthProvider“*, iz njega pročitati *„token“* i tako upravljati pravima korisnika u klijentskoj aplikaciji. (Slika 4.1.5)

```tsx
const { token } = useContext(AuthContext);
```

Slika 4.1.5 - Primjer načina na koji se čita stanje autentifikacije u aplikaciji.

Ovaj pristup je omogućio čistoću programskog koda i optimalno upravljanje stanjem autentifikacije, u suprotnom bilo bi potrebno u svakom trenutku pozivati čitanje stanja sesije i spremati u novu varijablu, za svaku komponentu, što narušava organizaciju i skalabilnost aplikacije. Ono što je problem, jeste taj da, dok veličina aplikacije raste, sve je teže upravljati različitim stanjima unutar aplikacije. Kada se upravljanje stanjem odvaja u zasebne komponente kao što je *„AuthProvider“*, i kada se takvom komponentom omota aplikacija, zagarantirano je organizirano upravljanje, zato što tako aplikacija ima pristup jedinki varijable koja mora uvijek imati jednako stanje u svim dijelovima programskog *„koda“*.

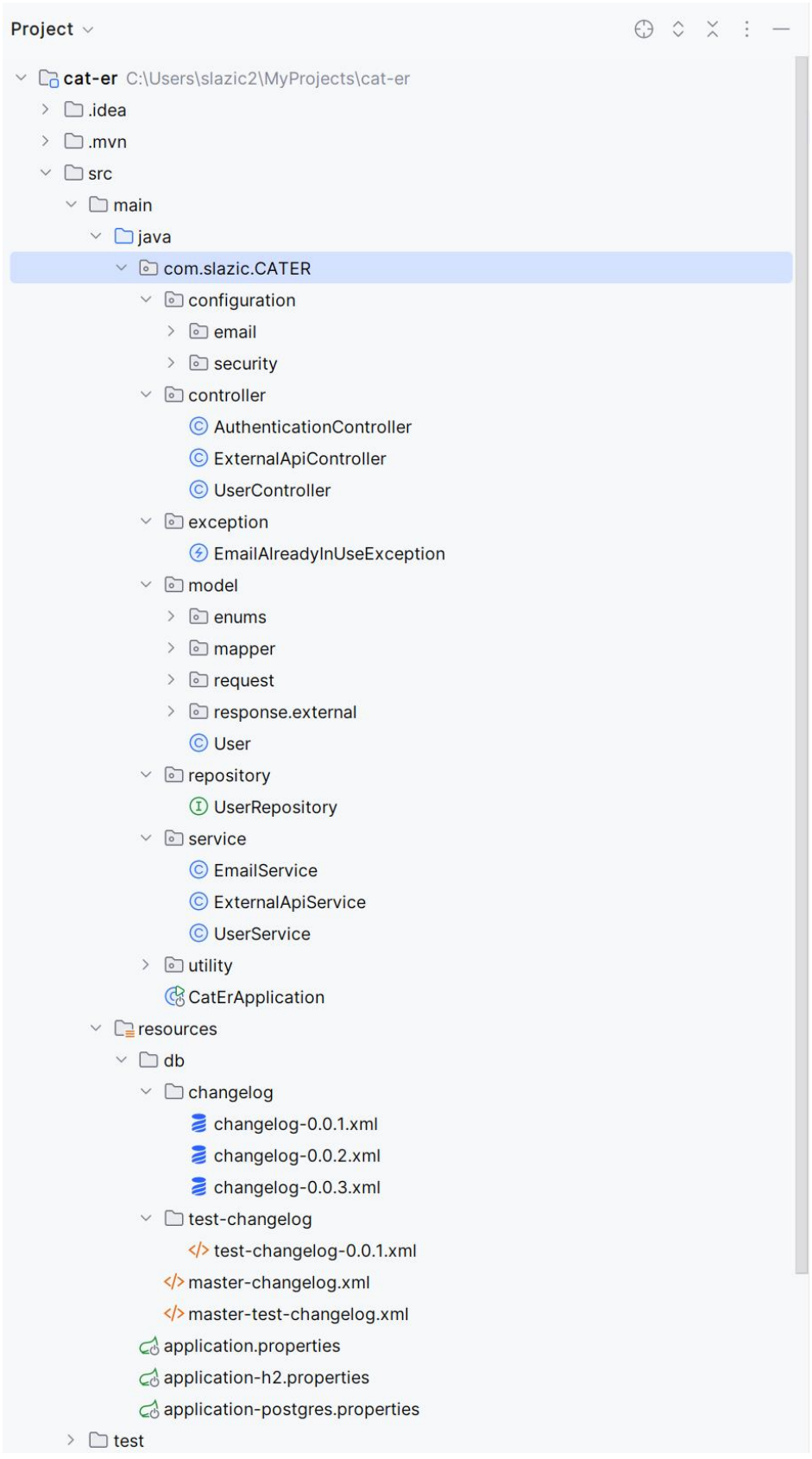
```
const onSubmit = async (values: SignInFormProps, formikHelpers: FormikHelpers<SignInFormProps>) => {
  const user: SignInUser = {
    email: values.email,
    password: values.password,
  };

  await signIn(user)
    .then((response) => {
      responseBody = response;

    })
    .catch((error) => {
      responseError = error.response;
    });
  if (responseError && responseError.status === 403) {
    console.log(responseError.data);
  } else if (responseError && responseError.status === 400) {
    formikHelpers.setFieldError("password", responseError.data);
  } else if (responseError && responseError.status === 423) {
  } else if (responseBody && responseBody.status === 200) {
    saveToken(responseBody.data);
    navigate(from, { replace: true });
  }
};
```

Slika 4.1.6 – Primjer spremanja „tokena“ u stanje sesije i aplikacije

```
<ul className="flex items-center">
  {token ? (
    <>
      {extractUserRole(token) === "ADMIN" && (
        <li className="">
          <NavLink to={"/dashboard"} className={navLinkStye}>
            DASHBOARD
          </NavLink>
        </li>
      )}
      <li className="">
        <NavLink to={"/profile"} className={navLinkStye}>
          PROFILE
        </NavLink>
      </li>
      <li className="">
        <button
          className="font-bold text-base ■text-neutral-700 px-4 ease-in-out duration-300 □hover:text-white hover:text-lg"
          onClick={() => onSignOut()}
        >
          SIGN OUT
        </button>
      </li>
    </>
  ) : (
    <li className="">
      <NavLink to={"/sign_in"} className={navLinkStye}>
        SIGN IN
      </NavLink>
    </li>
  )}
</ul>
```

Slika 4.1.7 –Upotreba „tokena“ za kondicionalno prikazivanje elemenata na navigacijskoj traci

4.2 Razvoj poslužitelj aplikacije



Slika 4.2.1 – Struktura „Java“ aplikacije

Kada se s mrežne aplikacije skine inicijalna verzija *„Java Spring“* projekta, prema slici 4.2, rezultat je struktura aplikacije kao što je prikazano na slici 4.2.1. Osnovni dijelovi strukture su:

- „java“ - direktorij u kojem je pohranjen izvorni kod „Java Spring“ aplikacije
- „resources“ – direktorij u kojem su pohranjene konfiguracijske datoteke za razvojno okruženje, bazu podataka, „docker“ sliku i sl.

„java“ direktorij, sadrži izvorni *„kod“*. Struktura i objašnjenje se nalazi u dokumentima, prema [1].

Cilj strukture je da programski *„kod“* drži organiziranim, kako za pojedinca tako i za tim koji razvija aplikaciju. Olakšava prepoznavanje različitih dijelova, a grupira ih na različite načine , bilo to po tipu datoteke, tipu dijela ili funkciji dijela

U svrhu izrade navedene aplikacije, primarno je korišten pristup podjele datoteka prema tipu dijela [1], što znači:

- „configuration“ – direktorij za konfiguracijske Java klase označene @Configuration
- „controller“ – direktorij za upravljačke klase označene @Controller za preusmjeravanje zahtjeva
- „exception“ – direktorij za prilagođene klase izuzetaka
- „model“ – direktorij za modele koji se pohranjuju u bazu podataka označene @Entity
- „repository“ – direktorij za klase koje definiraju pristup bazi podataka @Repository
- „service“ – direktorij za klase koje izvršavaju poslovnu logiku @Service
- „utility“ – direktorij za statične klase koje pružaju metode bez instanciranja

Potrebno je definirati *„modele“* kojima će se manipulirati u aplikaciju i u bazu podataka. Prema slici 4.2.3, klasa će biti označena s *„@Entity“* anotacijom s pomoću koje *„Spring“* aplikacija raspoznaje klasu kao model i pohranjuje ju u poseban kontejner za upravljanje modelima u aplikaciji.

Na slici je prikazana *„klasa“*, koja prezentira korisnika u aplikaciji, a ona će proširiti *„UserDetails“*, gotovu klasu definiranu u *„Spring Security“* ovisnosti. *„UserDetails“* nudi gotove metode i atribute, koji se kasnije koriste u konfiguracijskim klasama za autentifikaciju, filtriranje zahtjeva poslanih prema poslužitelju i generiranje *„tokena“*.

Korištenjem *„Lombook“* ovisnosti u *„Spring“* aplikaciji, omogućene su dodatne anotacije. Ove anotacije će prilikom pokretanja aplikacije generirati, takozvane *„getter“* i *„setter“* metode. To znači da će ove anotacije ukloniti šablonski *„kod“*, što znači da je *„klasa“* kraća i ljepše napisana.

Atributi koji opisuju korisnika mogu se vidjeti na slici 4.2.3.

```
@Repository  8 usages
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findByEmail(final @NotNull String email);  3 usages

    Optional<User> findByHashedIdentifier(final @NotNull String hash);  1 usage
}
```

Slika 4.2.2 – „@Repository“ „klasa“ za manipulaciju modela „User“

Kada je definiran model za manipulaciju u aplikaciji, kreira se pristupni sloj aplikacije za bazu podataka, a to je *„@Repository“ „klasa“*. Na slici 4.2.2 može se vidjeti *„repository“ „klasa“* koja implementira *„Data JPA“* ovisnost, prema [5]. *„Repository“ „klasu“* je dovoljno kreirati kao sučelje koje će proširiti *„JpaRepository“*, Za potpunu *„CRUD“* manipulaciju modelom, dovoljno je napisati nazive metoda, kao što je *„findByEmail“*, a *„Spring“* će znati generirati *„SQL“* upit koji će označiti korisnika u bazi podataka prema predanom podatku.

```java
@Data  38 usages
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "users")
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String firstName;

    @Column(nullable = false)
    private String lastName;

    @Column(nullable = false)
    private String email;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private LocalDateTime createdAt;

    @Enumerated(EnumType.STRING)
    private UserRole userRole;

    @Column(nullable = false)
    private Boolean verified;

    @Column(nullable = false)
    private Boolean locked;

    @Column(nullable = false)
    private String hashedIdentifier;
```

Slika 4.2.3 – Prikaz modela korisnika kojim se manipulira u aplikaciji

```java
@Service
@Slf4j
public class UserService implements UserDetailsService {

    private final UserRepository userRepository;  11 usages

    private final JwtService jwtService;  2 usages

    private final EmailService emailService;  2 usages

    private final EmailProperties emailProperties;  2 usages

    private final PasswordEncoder passwordEncoder;  3 usages

    @Autowired
    public UserService(
            final @NotNull UserRepository userRepository,
            final @NotNull JwtService jwtService,
            final @NotNull EmailService emailService,
            final @NotNull EmailProperties emailProperties,
            final @NotNull PasswordEncoder passwordEncoder
    ) {
        this.userRepository = userRepository;
        this.jwtService = jwtService;
        this.emailService = emailService;
        this.emailProperties = emailProperties;
        this.passwordEncoder = passwordEncoder;
    }

    @Override  7 usages
    public UserDetails loadUserByUsername(final @NotNull String username) throws UsernameNotFoundException {
        return userRepository.findByEmail(username).orElseThrow(
                () -> new UsernameNotFoundException("User with email " + username + " not found.")
        );
    }
```

Slika 4.2.4 – Izgled uslužne klase za označavanje korisnika u bazi podataka

Prema slici 3.1.1, definira se sloj između upravljača i repozitorija, a to je uslužna klasa. Namjena *„@Service“* klase je da izvršava poslovnu logiku.

Na slici 4.2.4 se vidi uslužna klasa nazvana *„UserService“*, a ona implementira *„UserDetailsService“* sučelje iz *„Spring Security“* ovisnosti, koji pruža *„loadUserByUsername“* metodu za dohvaćanje autentifikacijskih podataka iz baze podataka. To znači da će navedena metoda u stvari pozvati prijašnje implementiranu repozitorij klasu i njezinu *„findByEmail“* metodu, koja generira *„SQL“* naredbu za označavanje korisnika u bazi prema atributu elektroničke pošte.

U ovoj aplikaciji, koristi se *„e-pošta“* kao korisničko ime za prijavu ili registraciju u sustav.

```java
@Configuration
public class AuthenticationManagerConfiguration {

    private final UserService userService;  2 usages

    @Autowired
    public AuthenticationManagerConfiguration(@Lazy final @NotNull UserService userService) {
        this.userService = userService;
    }

    @Bean
    public AuthenticationManager authenticationManager(
            final @NotNull AuthenticationConfiguration configuration
    ) throws Exception {
        return configuration.getAuthenticationManager();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userService);
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        return authenticationProvider;
    }

    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
}
```

Slika 4.2.5 – Konfiguracijska klasa za inicijalizaciju upravitelja autentifikacije

Kad su implementirani *„model“* korisnika, *„repozitorij“* za generiranje *„SQL“* naredbe i uslužna *„klasa“* *„UserService“* koja poziva generiranu *„SQL“* naredbu, prema slici 3.3.1, implementira se arhitektura autentifikacijskog procesa, prema slikama 4.2.5 i 4.2.6. [3]

„Spring Security“ ovisnost zahtjeva niz instanci klasa koje surađuju kako bi autentificirali korisnika u sustav aplikacije. *„@Bean“* anotacijom nad metodom unutar *„@Configuration“* *„klase“*, *„Spring“* zna da je to metoda koja vraća instancu. Implementacija *„AuthenticationManger“* klase zahtjeva pružatelja autentifikacije, to jest instancu *„DaoAuthenticationProvider“* klase, što znači da se korisnika autentificira podatcima iz baze podataka. Osim toga zahtjeva instancu *„UserService“* klase, te instancu *„BCryptPasswordEncoder“* klase koja će korisnikovu lozinku jednosmjerno kriptirati. [3]

```java
@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;  3 usages

    private final UserService userService;  2 usages

    @Autowired
    public JwtAuthenticationFilter(
            final @NotNull JwtService jwtService,
            @Lazy final @NotNull UserService userService
    ) {
        this.jwtService = jwtService;
        this.userService = userService;
    }

    @Override  no usages
    protected void doFilterInternal(
            final @NonNull HttpServletRequest request,
            final @NonNull HttpServletResponse response,
            final @NonNull FilterChain filterChain
    ) throws ServletException, IOException {

        final String authenticationHeader = request.getHeader( s: "Authorization");
        final String jsonWebToken;
        final String userEmail;

        if (authenticationHeader == null || !authenticationHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        jsonWebToken = authenticationHeader.substring("Bearer ".length());
        userEmail = jwtService.extractUsername(jsonWebToken);

        if (SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userService.loadUserByUsername(userEmail);

            if (jwtService.isTokenValid(jsonWebToken, userDetails)) {
                UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(
                        userDetails,
                        credentials: null,
                        userDetails.getAuthorities()
                );

                authenticationToken.setDetails(
                        new WebAuthenticationDetailsSource().buildDetails(request)
                );

                SecurityContextHolder.getContext().setAuthentication(authenticationToken);
            }
        }

        filterChain.doFilter(request, response);
    }
}
```

Slika 4.2.6 – Autentifikacijski filtar prema „JWT“

Potrebno je definirati autentifikacijshi filtar za zahtjeve prema poslužitelju (Slika 4.2.6.).

Koristi se *„OncePerRequestFilter“*, što znači da *„FilterChain“* presreće svaki zahtjev i ispituje ga prije nego što dođe do poslužitelja. Ako pristigli zahtjev ne sadrži zaglavlje s *„tokenom“*, to znači da korisnik nije prijavljen u sustav ili ne posjeduje *„token“*.

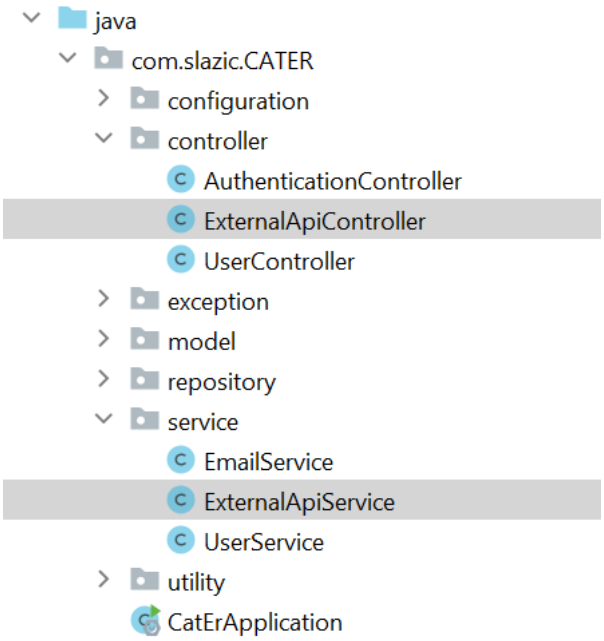
Ako se u zaglavlju zahtjeva nalazi *„token“*, znači da korisnik možda ima pristup sustavu. Ako prijašnje nije prijavljen u sustav, zaglavlje zahtjeva se uspoređuje sa zapisom u bazi podataka, i ako je zaglavlje zahtjeva ispravno, korisnik se zapisuje u *„SecurityContext“*.

Osim presretanja svakog zahtjeva, i izvršavanja autentifikacije, na slici 4.2.7 se vidi način na koji se korisnika zapisuje po prvi put u sustav. Kada korisnik kroz upitnik pošalje korisničko ime, odnosno *„e-poštu“*, njega se pretraži u bazi i zapiše u *„SecurityContext“*, a na klijentsku stranu aplikacije, vraća se novo generirani *„token“*, kojeg ubuduće *„OncePerRequestFilter“* odobrava u sustav.

```
@PostMapping("/login")
public ResponseEntity<Object> login(
        @RequestBody @Valid @NotNull final UserLoginRequestDto userLoginRequest
) throws MessagingException {
    User user = new User();
    try {
        user = userService.loadUserByEmail(userLoginRequest.getEmail());

        // Authenticates the user inside the security context
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                        userLoginRequest.getEmail(),
                        userLoginRequest.getPassword()
                )
        );

        return ResponseEntity
                .status(HttpStatus.OK)
                // Generates JWT token
                .body(userService.authenticate(user));
```

Slika 4.2.7 – Autentifikacija korisnika u SecurityContext

4.3 Integracija s vanjskim API



Slika 4.3.1 – Razredne datoteke za integraciju s vanjskim API

Za integraciju aplikacije s vanjskim *„API“*, te dohvaćanje informacija o cijenama kriptovaluta na različitim mjenjačnicama, implementirane su dvije klase, *„ExternalApiController“* upravljačka *„klasa“* i *„ExternalAPIService“* uslužna *„klasa“*.

Na slici 4.3.2, vidi se primjer upravljačke metode koja definira putanju, za parametre prima simbol kriptovalute po kojem se pretražuje na mjenjačnici, vremenski interval pregleda detalja i naziv mjenjačnice na kojoj se pretražuje. Kao rezultat, metoda vraća objekt u kojem se nalazi lista detalja cijene kriptovalute po vremenskom intervalu.

Upravljačka metoda koja preusmjerava rezultate između klijenta i poslužitelja, zahtjeva instancu uslužne *„klase“* koja će komunicirati s vanjskim *„API“*, i dohvatiti željene rezultate. Ova uslužna *„klasa“* koristi *„RestTemplate“* i njezinu metodu *„getForObject“*, prikazano na slici 4.3.3. Uslužna *„klasa“ „ExternalApiService“* u konstruktoru inicijalizira *„RestTemplate“* objekt pomoću graditeljske metode, a zatim prilikom pozivanja metode *„getForObject“*, predaje joj se putanja vanjskog *„API“* prikazano na slici 4.3.4. Rezultat takvog poziva je objekt definiran od strane *„API“* mjenjačnice.

```java
@GetMapping("/singleCandlestickDataFromSelectedMarket")
public ResponseEntity<Object> getSingleCandlestickDataFromSelectedMarket(
        @RequestParam final String symbol,
        @RequestParam final String timeframe,
        @RequestParam final String exchangeName
) {
    try{
        return ResponseEntity
                .status(HttpStatus.OK)
                .body(apiService.fetchSingleCandlestickDataFromSelectedMarket(symbol, timeframe, exchangeName));
    } catch (final Exception exception) {
        log.error("Failed to fetch.", exception);
        return ResponseEntity
                .status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body(exception);
    }
}
```

Slika 4.3.2 – Upravljačka metoda na poslužitelj strani aplikacije

```java
@Service  3 usages
public class ExternalApiService {

    private final RestTemplate restTemplate;  14 usages

    public ExternalApiService(
            final RestTemplateBuilder restTemplateBuilder
    ) {
        this.restTemplate = restTemplateBuilder.build();
    }
```

Slika 4.3.3 – Primjer inicijalizacije „RestTemplate“ razreda za poziv na vanjski „API“

```
final String oneMinUrlB = "https://api.binance.com/api/v3/uiKlines?symbol="+ symbol +"&interval=1m&startTime=" + last8hStartTime;
final String oneHourUrlB = "https://api.binance.com/api/v3/uiKlines?symbol="+ symbol +"&interval=1h&startTime=" + last1MothStartTimeMinus11days;
final String oneDayUrlB = "https://api.binance.com/api/v3/uiKlines?symbol="+ symbol +"&interval=1d&startTime=" + last2YearsStartTimeMinus8Months;
final String oneWeekUrlB = "https://api.binance.com/api/v3/uiKlines?symbol="+ symbol +"&interval=1w&startTime=" + last9YearsStartTime;
final String oneMonthUrlB = "https://api.binance.com/api/v3/uiKlines?symbol="+ symbol +"&interval=1M&startTime=" + last10YearsStartTime;

final String oneMinUrlK = "https://api.kraken.com/0/public/OHLC?pair="+ symbol +"&interval=1&since=" + last8hStartTime;
final String oneHourUrlK = "https://api.kraken.com/0/public/OHLC?pair="+ symbol +"&interval=60&since=" + last1MothStartTimeMinus11days;
final String oneDayUrlK = "https://api.kraken.com/0/public/OHLC?pair="+ symbol +"&interval=1440&since=" + last2YearsStartTimeMinus8Months;
final String oneWeekUrlK = "https://api.kraken.com/0/public/OHLC?pair="+ symbol +"&interval=10080&since=" + last9YearsStartTime;
final String oneMonthUrlK = "https://api.kraken.com/0/public/OHLC?pair="+ symbol +"&interval=40320&since=" + last10YearsStartTime;

if (Objects.equals(exchangeName, b: "Binance")) {
    externalBinanceResponseData = switch (timeframe) {
        case "1m" -> restTemplate.getForObject(oneMinUrlB, List.class);
        case "1h" -> restTemplate.getForObject(oneHourUrlB, List.class);
        case "D" -> restTemplate.getForObject(oneDayUrlB, List.class);
        case "W" -> restTemplate.getForObject(oneWeekUrlB, List.class);
        case "M" -> restTemplate.getForObject(oneMonthUrlB, List.class);
        default -> externalBinanceResponseData;
    };
```

Slika 4.3.4 – Primjer poziva „getForObject“ metode iz „RestTemplate“ razreda

Nakon što vanjski *„API“* uspješno dostavi rezultat, takav objekt je potrebno na određene načine prilagoditi za prikazivanje na klijentskoj strani aplikacije. Rezultat koji vraća *„API“* za svaku mjenjačnicu je drugačiji.

```
const fetchSingleCandlestickDataFromSelectedMarket = async (
  symbol: string | undefined,
  timeframe: string | undefined,
  exchangeName: string | undefined,
  token: Jwt
) => {
  return await axios.get(BASE_URL + "/fetchExternal/singleCandlestickDataFromSelectedMarket", {
    params: { symbol: symbol, timeframe: timeframe, exchangeName: exchangeName },
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });
};
```

Slika 4.3.5 – Primjer povezivanja klijent i poslužitelj strane koristeći „axios“ paket

Na klijentskoj strani aplikacije, koristi se *„axios“* paket za komunikaciju s poslužiteljskom stranom aplikacije. Na slici 4.3.5 vidi se implementacija takve funkcije. Navedena funkcija poziva se na promjenu bilo koje od navedenih stavki, kao što su: simbol po kojem se pretražuje mjenjačnica, vremenski interval pregleda detalja ili naziv mjenjačnice na kojoj se pretražuje.

4.4 Korisničke uloge i prava pristupa

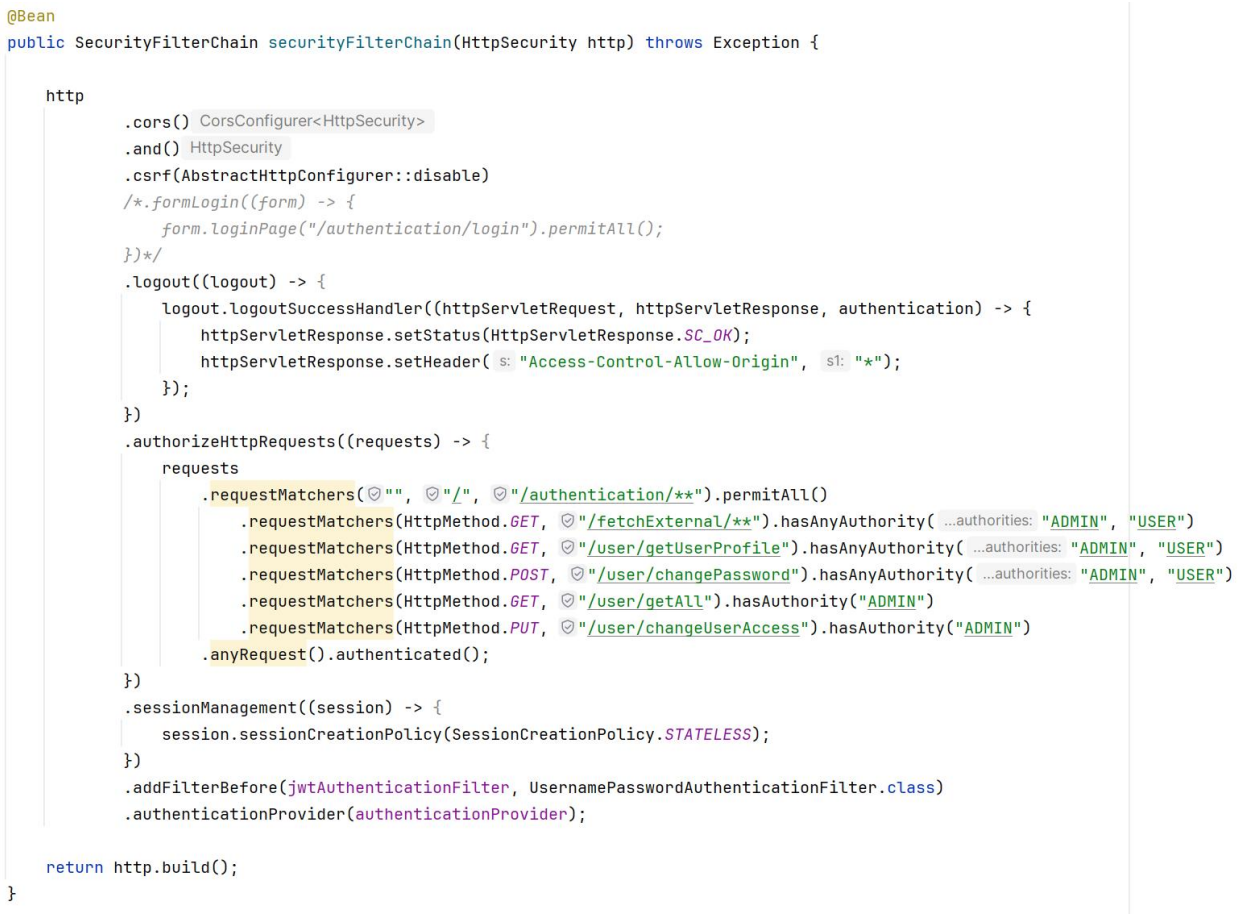
Za upravljanje pravima pristupa dijelovima klijentske aplikacije, implementirana je *„ProtectedRoute“* komponenta. (Slika 4.4.1). Komponenta prima listu korisničkih uloga koje imaju pravo pristupa naređenoj komponenti (Slika 4.1.3.). Ova komponenta također koristi *„useContext“* funkciju koja poziva dostavljača stanja autentifikacije *„AuthContext“*, čita *„token“*, dekodira ga i na osnovu toga prepoznaje korisničku ulogu. Ako se korisnička uloga, iz *„tokena“*, nalazi u listi predanih korisničkih uloga komponenti, korisnik ima pravo pristupa i navedena komponenta mu se prikazuje na ekran.

Ako je korisnik prijavljen u sustav, ali nema pravo pristupa na osnovu predane liste korisničkih uloga, korisniku se umjesto odabrane komponente, prikazuje komponenta koja nagovještava da nema pravo pristupa. Na kraju, ako korisnik uopće nije prijavljen u sustav, korisniku se prikazuje zaslon s obrascem za prijavu u sustav.

```
const ProtectedRoute: FC<ProtectedRouteProps> = ({ allowedRoles }) => {
  const { token } = useContext(AuthContext);
  const location = useLocation();
  let claims: JwtClaims | undefined;

  if (token) {
    claims = jwtDecode(token);
  }

  return claims?.role ? (
    allowedRoles.includes(claims?.role) ? (
      <Outlet />
    ) : (
      <Navigate to={"/unauthorized"} state={{ from: location }} replace />
    )
  ) : (
    <Navigate to={"/sign_in"} state={{ from: location }} replace />
  );
};
```

Slika 4.4.1 – „ProtectedRoute“ komponenta za upravljanjem pravom pristupa

Za dekodiranje *„tokena“* koristi se funkcija *„jwtDecode“*, koja se može naći u *„Node.js“* paketu „jwt-decode“, prema [10].



Slika 4.4.2 – SecurityFilterChain instanca za upravljanje pravom pristupa

Na poslužiteljskoj strani, autorizacija korisnika izgleda drugačije, ali također funkcionira na principu definiranja linkova na koje korisnik ima, odnosno nema pravo pristupa.

Kad je implementiran odabrani sigurnosni filtar zahtijeva, u ovom slučaju *„OncePerRequest“ „JWT“* filtar, predaje ga se lancu filtara, ispred ostalih filtara. Pošto je aplikacija zatvorenog tipa, pokreće se isključivo na lokalnom računalu, razvijena u svrhe istraživanja i učenja, tehnike protiv CORS i CSRF slabosti su isključene.

Na kraju, lancu filtara se zadaju razine autorizacije. Svaka metoda unutar svake *„@Controller“* upravljačke klase mora imati definiranu putanju putem koje joj se pristupa. Potrebno je zadati autorizacijska prava za svaku postojeću *„@Controller“* metodu, po ulogama, prema slici 4.4.2. [1, 3]

5. REZULTATI

Ishodi i postignuća tijekom procesa razvoja softvera, predstavljeni su rezultatima koji pokazuju koliko su ciljevi projekta ispunjeni. Faza razvoja i posebni ciljevi utječu na ove rezultate.

Rezultate je važno prikazati, zato što su mjera uspjeha projekta, a pokazuju zadovoljava li program očekivanja korisnika i jesu li ciljevi projekta ispunjeni. Pomažu u demonstriranju povrata vremena, novca i truda uloženog u razvoj softvera. Dopuštaju timu da nauči iz prošlih pogrešaka i poboljša proces razvoja. Kvaliteta ima izravan utjecaj na zadovoljstvo korisnika, što je bitno za komercijalni uspjeh softvera.

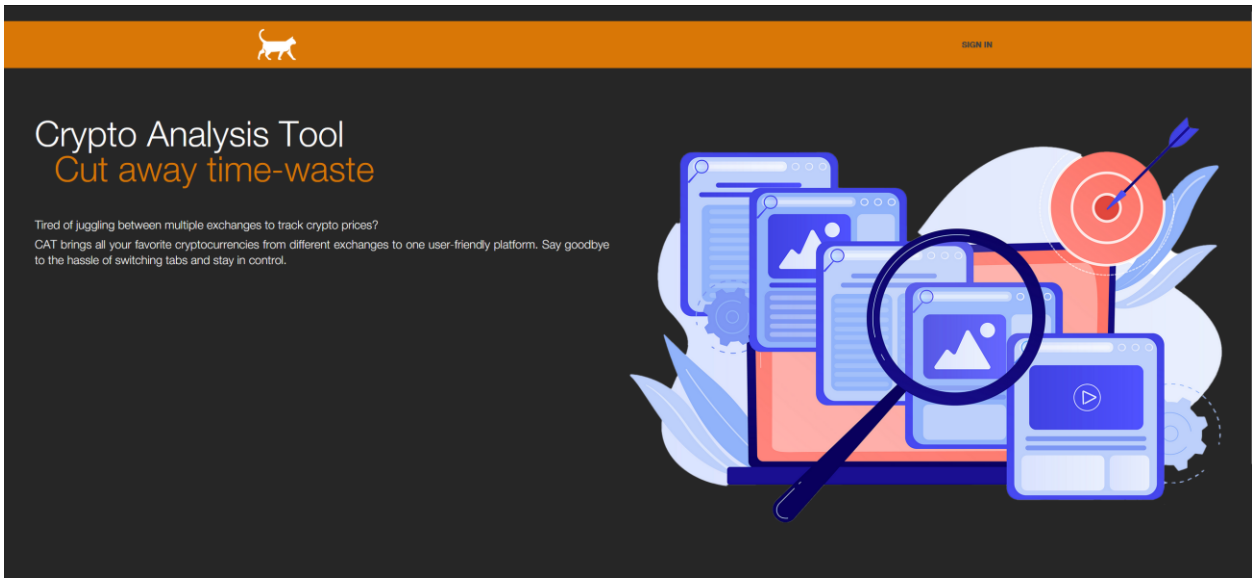
U sljedećem poglavlju prikazan je konačni izgled aplikacije. Ovaj dio završnog rada posebnu pažnju pridodaje na dizajn korisničkog sučelja i korisničko iskustvo, s naglaskom na to kako se dizajnira intuitivna i ugodna korisnička interakcija, navigacija i vizualni aspekti. Objašnjene su dizajnerske odluke koje pomažu aplikaciji da izgleda suvremeno i korisno.

U poglavlju testiranja i evaluacije ispitane su različite strategije testiranja koje se koriste kako bi se zajamčila kvaliteta i stabilnost aplikacije. Objašnjeno je testiranje jedinice, ističući kako su ti procesi pomogli u prepoznavanju i isključivanju potencijalnih problema prije nego što aplikacija postane dostupna korisnicima.

5.1 Postignuti rezultati

Prema slici 3.4.1 (dijagram toka), početno stanje aplikacije je *„landing page,“* odnosno početna stranica aplikacije. Svrha početne stranice je da privuče pažnju posjetioca i da mu pruži osnovne informacije o aplikaciji, odnosno, da ga potakne na određenu akciju.

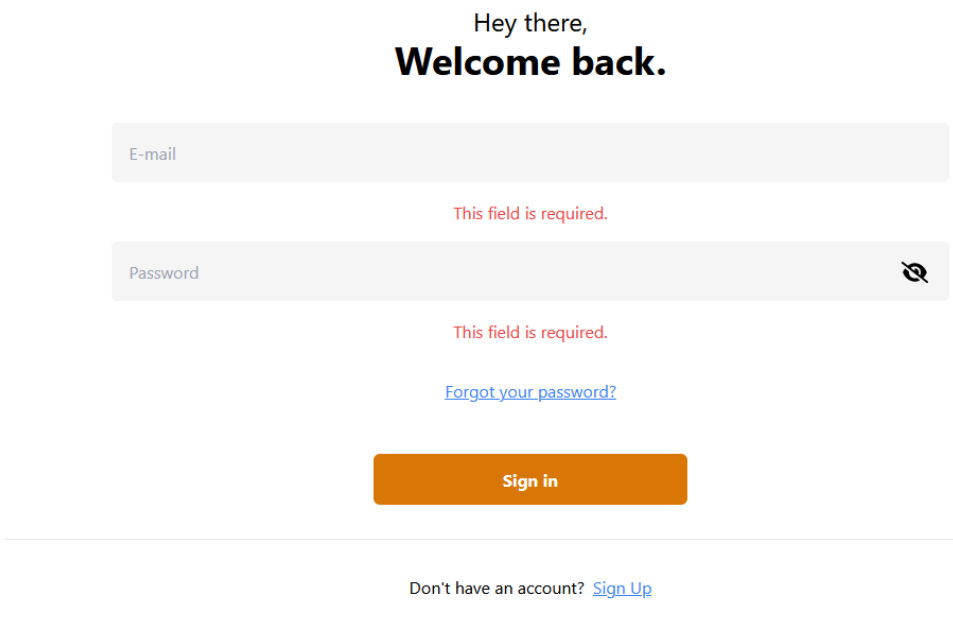
Na slici 5.1.1, vidi se početna stranica mrežne aplikacije vezane za analizu kriptovaluta. Početna stranica pruža informaciju, o tome za što aplikacija služi. Na prvi utisak izgleda jednostavno, moderno i ukratko prenosi ideju.



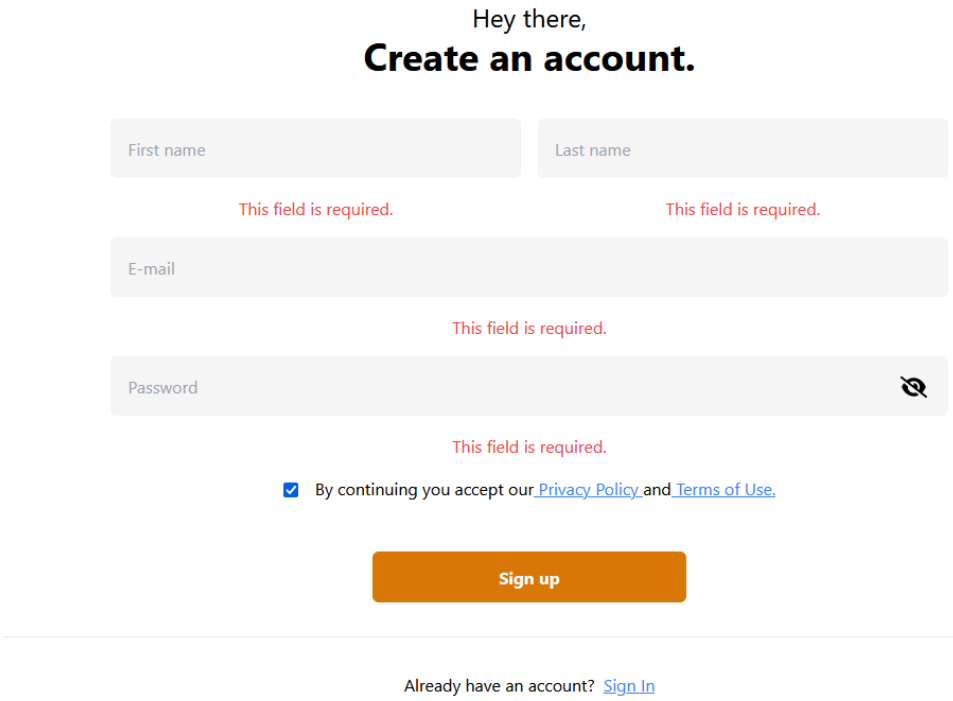
Slika 5.1.1 – Početna stranica aplikacije

Na početnoj stranici (Slika 5.1.1), korisnik nema prava za pregled aplikacije, osim opcije za prijavu u sustav koja se nalazi u gornjem desnom kutu sučelja.

Kada korisnik odabere jedinu ponuđenu opciju, *„sign in“*, preusmjerava ga se na novi zaslon sučelja. Prema slici 5.1.3, vidi se obrazac koji korisnik mora ispuniti kako bi dobio pristup aplikaciji. Polja *„e-pošta“* i lozinka su obavezna. Ukoliko korisnik nema račun za prijavu, na dnu sučelja postoji opcija *„sign up“*. Klikom na opciju *„sing up“*, za korisnika se otvara sučelje registracije, prema slici 5.1.4. Obavezna su polja za unos imena, prezimena, *„e-pošte“* i lozinka.



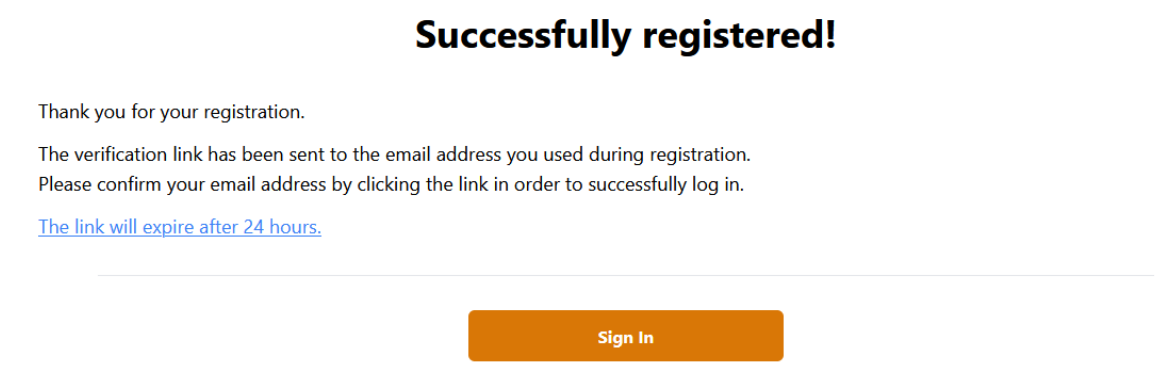
Slika 5.1.2 – Stranica za prijavu korisnika



Slika 5.1.3 – Stranica za registraciju korisnika

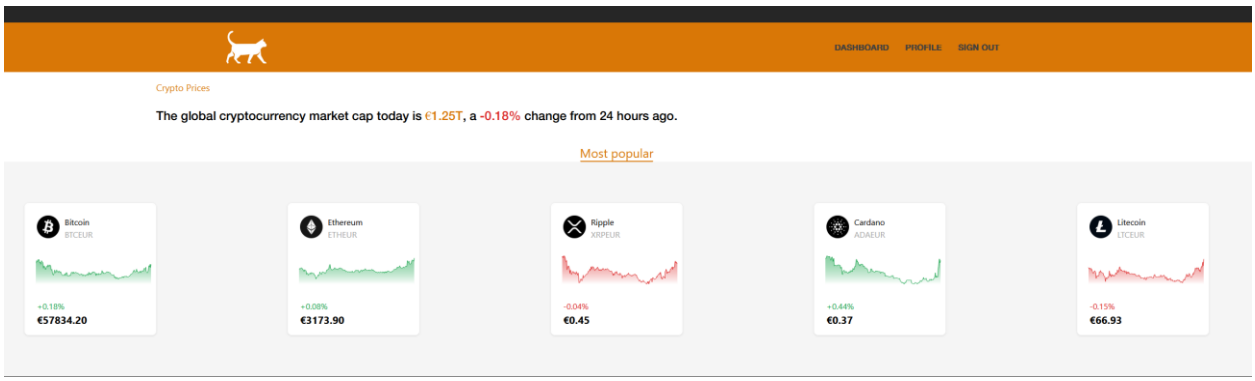
Korisnik je uspješno ispunio registracijski obrazac, ako je upisao jedinstvenu *„e-poštu“*, koja se ne podudara niti s jednom u bazi podataka. Prema slici 5.1.4, vidi se obavijest koju korisnik dobija. Nakon toga, korisnik i dalje nema pristup sustavu, nego mora potvrditi upisanu *„e-poštu*, tako što će otvoriti vezu koja se nalazi u poštanskom pretincu, unutar 24 sata.

Ako je korisnik potvrdio otvaranje računa klikom na vezu, za njega se u bazi podataka, zastavica *„verified“*, odnosno zastavica za potvrđenu *„e-poštu“*, postavlja na istinu. Korisnik klikom na tipku *„sign in“*, ispunjava korisničke podatke (Slika 5.1.2), sustav generira *„JWT“* i korisnik dobija pristup sustavu.



Slika 5.1.4 – Stranica obavijesti za uspješnu registraciju

Slika 5.1.5 prikazuje izgled početne stranice kada korisnik ima pristup sustavu. Ponuđene su nove opcije, ovisno o ulozi, kao što su „dashboard“, odnosno nadzorna ploča, ukoliko račun ima privilegiju administratora, odabir jedne od pet poznatih kriptovaluta, pregled profila ili odjava iz sustava.

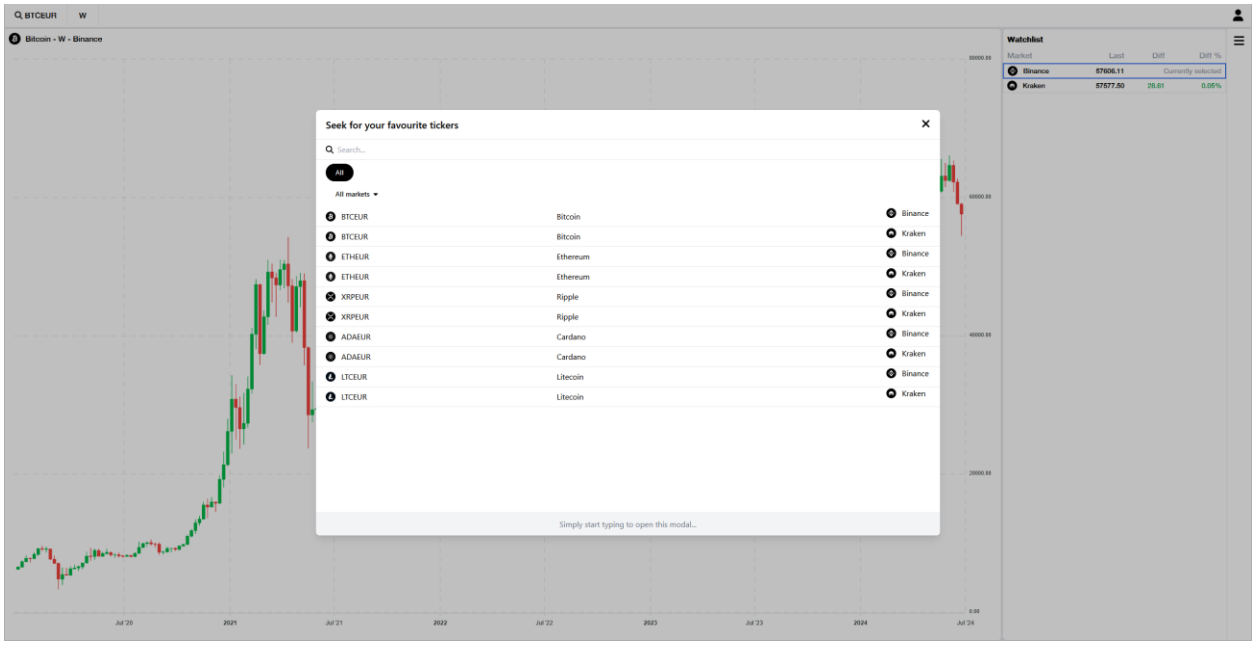


Slika 5.1.5 – Početna stranica nakon prijave – odabir kriptovalute

Kad korisnik odabere željenu valutu, koja je prikazana na početnoj stranici, preusmjeren je na novo sučelje koje prikazuje grafikon, odnosno povijest cijene i najnoviju cijenu kriptovalue. (Slika 5.1.6)

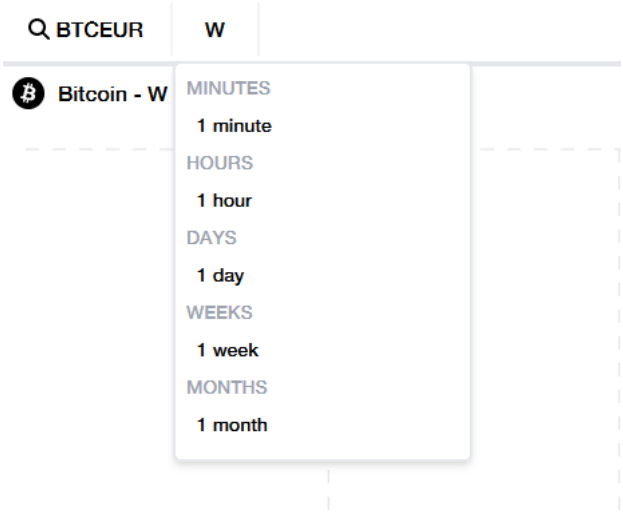
Slika 5.1.6 – Prikaz detalja pojedine kriptovalute

Kad korisnik želi pogledati povijest ostalih kriptovaluta, prema slici 5.1.7, ima opciju pretraživanja u gornjem lijevom kutu sučelja.



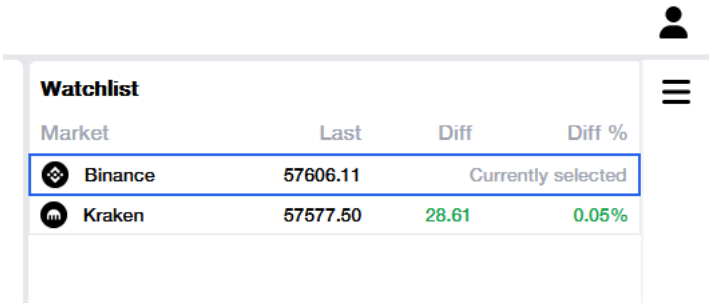
Slika 5.1.7 – Izbornik za pretraživanje svih kriptovaluta na svim tržnicama

Povijest cijene kriptovalute prikazana je u ovisnosti o odabranom intervalu. Tako postoje intervali od, npr., jedne minute ili jednog sata, gdje jedan stupac na grafikonu predstavlja kretanje cijene u odabranoj minuti ili u odabranom satu. Na slici 5.1.8, vidi se padajući izbornik putem kojeg korisnik odabire željeni interval za pregled povijesti cijene kriptovalute.



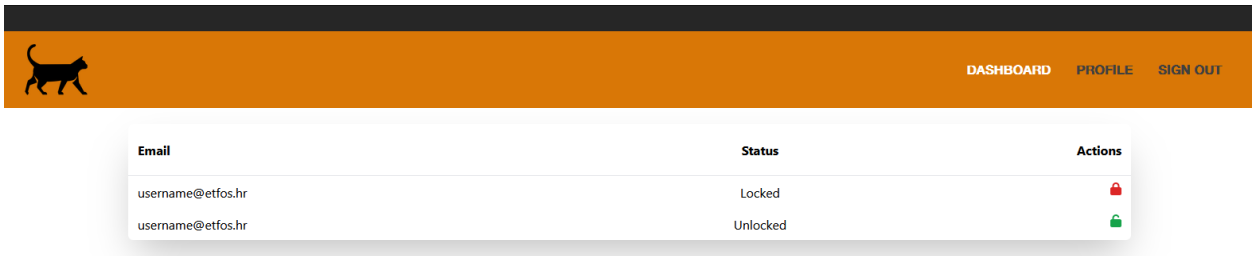
Slika 5.1.8 – Izbornik za odabir intervala pregleda detalja

Osim toga, korisnik na desnoj strani sučelja ima jednostavan izbornik gdje, za trenutno odabranu kriptovalutu, može vidjeti cijenu iste na ostalim mjenjačnicama. Prema slici 5.1.9, korisnik ne mora tražiti cijenu kriptovalute tako što će kontaktirati „online“ posrednik poput *„Kraken“*, zatim *„Coinbase“*, nego na istoj listi cijenu vidi na oba posrednika.



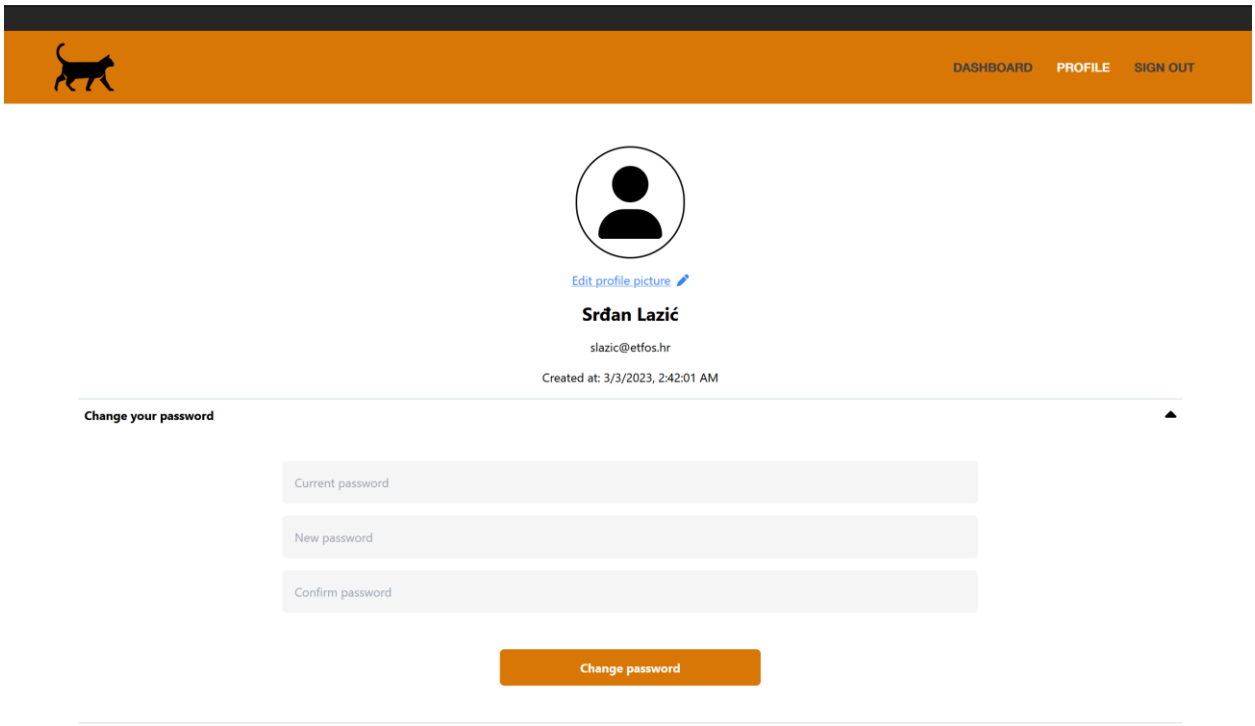
Slika 5.1.9 – Izbornik za promjenu tržnice odabrane kriptovalute

Kad korisnik ima privilegiju administratora, tad ima pristup već spomenutoj upravljačkoj ploči na glavnom izborniku. Na slici 5.1.10, vidi se mogućnost upravljanja korisnicima koji su registrirani u sustav. Odabirom na katanac, pod sekcijom *„actions“*, odnosno akcije, administrator onemogućava pristup odabranom korisniku, odnosno omogućava pristup , ako je prijašnje bio onemogućen.



Slika 5.1.10 – Nadzorna ploča za upravljanje pristupom korisnika

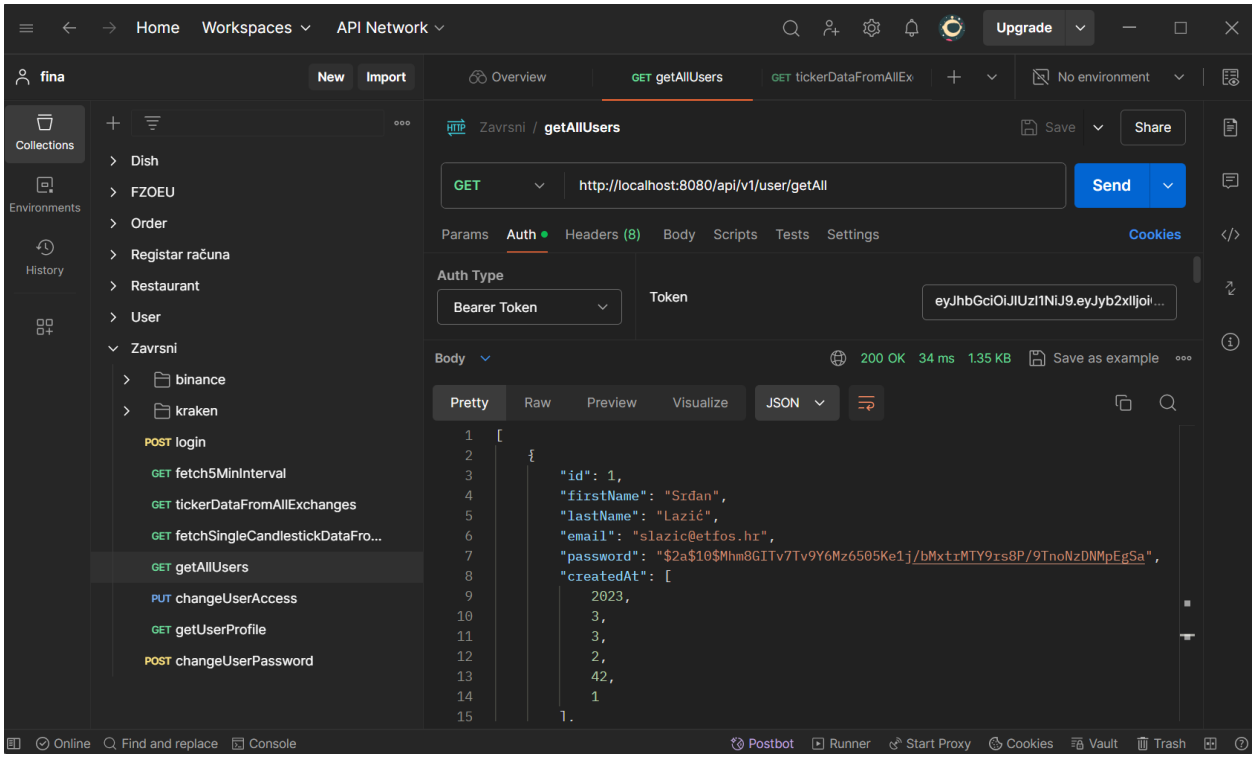
Konačno, za sve korisnike, prema slici 5.1.11, postoji opcija upravljanja korisničkim računom i pristupnim podatcima za sustav. Unosom trenutne lozinke i unosom nove željene lozinke, u bazi podataka se za korisnika mijenjaju pristupni podatci, a korisnik dobija novi *„JWT“* za pristup.



Slika 5.1.11 – Stranica profila korisnika

5.2 Testiranje i evaluacija

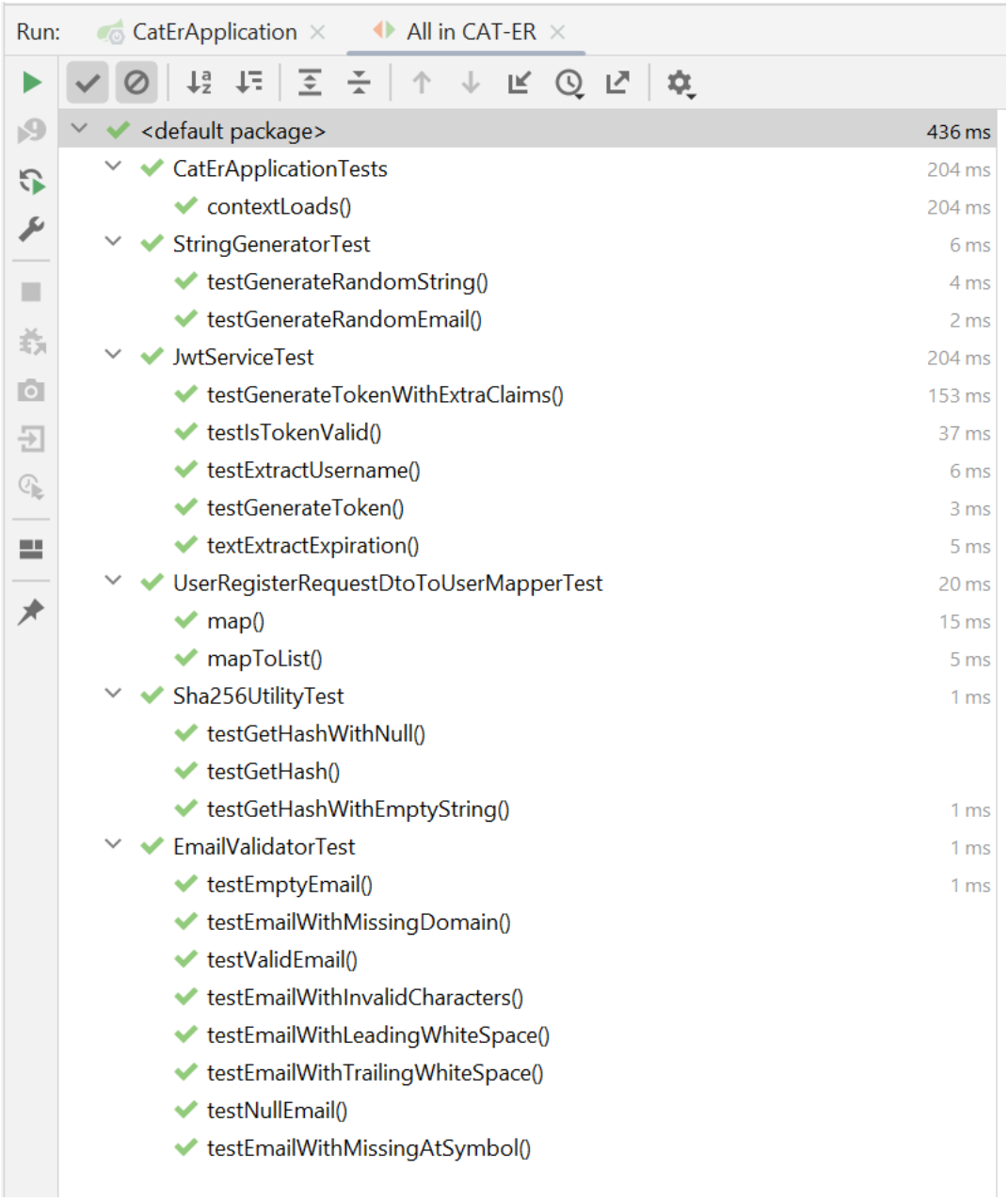
Prilikom izrade ove aplikacije korištene su različite metode testiranja. Jedna od metoda testiranja aplikacije je putem *„Postman“* programa.



Slika 5.2.1 – Primjer getAllUsers simuliranog zahtjeva

„Postman“ je program koji korisniku omogućava stvaranje simuliranih zahtjeva prema poslužitelju, tako što se opcijom kreira novi zahtjev, dodjeli mu se ime kao što je to na slici 5.2.1. Potrebno je definirati metodu zahtjeva, u ovom slučaju GET. Potrebno je definirati putanju na koju se zahtjev referencira, u odnosu na definiranu putanju poziva se upravljačka metoda poslužitelja. Na kraju, ukoliko je potrebno, definira se autentifikacijsko i autorizacijsko zaglavlje. Pošto aplikacija koristi *„JWT“* kao sustav autentifikacije i autorizacije, u programu se odabire *„Bearer Token“*. Na dnu su prikazani rezultati koje bi poslužitelj dostavio klijentskoj strani aplikacije. Informacije sadržane u *„JWT“* mogu se pročitati prema [12]

Osim korištenja *„Postman“* aplikacije, poslužiteljska strana projekta testirana je pisanjem automatiziranih testova koristeći *„JUnit 5“* testnog okvira.



Slika 5.2.1 – Lista napisanih automatiziranih testova

Na slici 5.2.1 vidi se lista napisanih testnih metoda. Bitne za spomenut su: validacija *„JWT“*. Potrebno je osigurati da poslužitelj pravilno interpretira *„token“* i njegove parametre. Osim toga Sha256 *„koder“* koji jednosmjerno kriptira i uspoređuje lozinku.

Na kraju validacija elektroničke pošte gdje je bitno provjeriti da korisnik unosi pravilnu adresu zato što aplikacija šalje poštu putem koje se potvrđuje identitet korisnika za pristupsustavu.

6. ZAKLJUČAK

U ovom seminaru obrađena je detaljna rasprava o izradi web aplikacije za administraciju korisničkih računa i istraživanju cijena kriptovaluta na mnogim burzama, koja koristi *„Java Spring“* i *„React“* tehnologiju.

Ispitani su glavni teorijski temelji aplikacije u stvarnom svijetu i poteškoće u području mrežnih aplikacija kroz tematska poglavlja, s fokusom na sigurnost, skalabilnost i korisničko iskustvo. *„React“* omogućava da aplikacija ima dinamično korisničko sučelje, što povećava interaktivnost i učinkovitost aplikacije. *„Java Spring“* pokazuje se kao snažan okvir za razvoj na razini poduzeća, nudeći otpornost i lakoću povezivanja s brojnim uslugama. Zbog zaštite korisničkih podataka u financijskim sustavima, ugrađene su sigurnosne značajke definiranjem korisničkih uloga i korištenjem *„JWT“* za autentifikaciju.

Integracijom s API-ima s mrežnih stranica, kao što su *„Binance“*, *„Coinbase“* i *„Kraken“*, analiza kriptovalute u stvarnom vremenu omogućava korisnicima da gledaju tržišne trendove i prilike, istovremeno pružajući sveobuhvatnu snimku svoje imovine na jednom mjestu. Konačno, demonstrirane su prednosti prilagodljivih alata za izradu grafikona i strategija koristeći instance platforme *„TradingView“*. Osim što je prikazano kako se tehnologije mogu implementirati, u ovom projektu je također pokazano kako se različite komponente integriraju u stvaranju sveobuhvatnog rješenja koje zadovoljava potrebe administratora i korisnika za upravljanjem financijskim portfeljima.

Seminar je zaključen isticanjem značaja spajanja tehnoloških inovacija s funkcionalnošću i sigurnošću za izradu vrhunske mrežne aplikacije koja zadovoljava zamršene potrebe tržišta kriptovaluta i nudi potrošačima sučelje prilagođeno korisniku i pouzdanu analizu financijskih podataka.

LITERATURA

[1] Spring Framework, spring.io, dostupno na: https://docs.spring.io/spring-framework/reference/overview.html [02.09.2024.]

[2] React, reactjs.org, dostupno na: https://legacy.reactjs.org/docs/getting-started.html [19.06.2024.]

[3] Spring Security, spring.io, dostupno na: https://docs.spring.io/spring-security/reference/servlet/authentication/index.html [02.09.2024.]

[4] TradingView Web Application, tradingview.com, dostupno na: https://www.tradingview.com/chart/ [09.07.2024.]

[5] Spring Data JPA, spring.io, dostupno na: https://docs.spring.io/spring-data/jpa/reference/jpa.html [02.09.2024.]

[6] PostgresSQL, postgresql.org, dostupno na: https://www.postgresql.org/docs/ [19.06.2024.]

[7] Node.js, nodejs.org, dostupno na: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs [09.07.2024]

[8] Create React App, create-react-app.dev, dostupno na: https://create-react-app.dev/docs/adding-typescript/ [09.07.2024.]

[9] Spring Initializr, start.spring.io, dostupno na: https://start.spring.io/ [09.07.2024]

[10] NPM, npmjs.com, dostupno na: https://www.npmjs.com [03.09.2024.]

[11] Liquibase, liquibase.com, dostupno na: https://docs.liquibase.com/home.html [19.06.2024.]

[12] JSON Web Tokens, jwt.io, dostupno na: https://jwt.io/ [19.06.2024.]

SAŽETAK

Web aplikacija za upravljanje računima *„online“* mjenjačnica omogućava korisnicima analizu kriptovaluta s različitih mjenjačnica na jednoj stranici, uz mogućnost pretraživanja i prikaza podataka u različitim vremenskim intervalima. Razvijena je korištenjem *„React“*-a za klijent stranu i *„Java Spring“* za poslužitelj stranu. Aplikacija se spaja na *„PostgreSQL“* bazu podataka, dok se *„H2“* baza koristi za testno okruženje. Za praćenje promjena u bazi podataka koristi se *„Liquibase“*. Naglasak aplikacije je na sigurnosnoj arhitekturi, uključujući potvrdu e-pošte prilikom registracije, razmjenu *„JWT“* i postupke autorizacije. Aplikacija podržava dvije korisničke uloge: administratora i korisnika, omogućavajući specifična prava pristupa i upravljanje računima te transakcijama u skladu s dodijeljenom ulogom.

Ključne riječi: analiza kriptovaluta, baza podataka, react, sigurnost, spring framework

ABSTRACT

Online exchange account management web application allows users to analyze cryptocurrencies from different exchanges on one page, with the ability to search and display data at different time intervals. It was developed using React for the frontend and Java Spring Boot for the backend. The application connects to the PostgreSQL database, while the H2 database is used for the test environment. Liquibase is used to track changes in the database. The application's emphasis is on security architecture, including email confirmation during registration, JWT token exchange, and authorization procedures. The application supports two user roles: administrator and user, enabling specific access rights and management of accounts and transactions in accordance with the assigned role.

Key words: cryptocurrency analysis, database, react, security, spring framework

ŽIVOTOPIS

Srđan Lazić je student računalne znanosti i informacijskih tehnologija. Svoje osnovno obrazovanje, završio je u Osnovnoj školi Markušica, 2016. godine. Srednju školu pohađao je u Vukovaru, Tehnička škola Nikole Tesle, smjer računalni tehničar, do 2020. godine.

Primarno znanje i iskustvo koje posjeduje je razvijanje web aplikacija u modernim tehnologijama kao što su: *„TypeScript“*, *„React“*, *„Java“*, *„Spring“*, *„PHP“*, koje uključuju manipulaciju podataka u bazi podataka kao što je *„PostgreSQL“*, koristeći *„SQL“* i *„Liquibase“* za praćenje verzije. Posjeduje znanje u razvijanju klijent aplikacija koje zahtijevaju ugodno korisničko iskustvo i sučelje razvijano uz pomoć *„Bootstrap“*, *„Tailwind CSS“*, *„Ant Design“*.

Osim toga, bavi se održavanjem računalnih sustava i servisiranjem računalnih komponenti.

Godine 2019., u sklopu pohađanja srednje škole, imao je priliku izaći na *„MTA Networking Fundamentals (98-366)“* ispit, koji je *„Microsoft“* pružio svim srednjoškolcima strukovnih škola, uspješno ga položio i osvojio certifikat.

Detaljne informacije mogu se pronaći na stranicama *„LinkedIn“* aplikacije na poveznici: https://linkedin.com/in/lazicc9

Ostali projekti mogu se pronaći na stranicama *„GitHub“* i *„GitLab“*:

https://github.com/lazic7

https://gitlab.com/slazic

PRILOZI

Poveznica na *„GitLab“* repozitorij pozadinske aplikacije: https://gitlab.com/slazic/cat-er

Poveznica na *„GitLab“* repozitorij sučelja aplikacije: https://gitlab.com/slazic/cat-er-fontend