Društvena mreža za ljubitelje satova

Šarčević, Luka

Master's thesis / Diplomski rad

2023

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:915009

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-11**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

Društvena mreža za ljubitelje satova

Diplomski rad

Luka Šarčević

Osijek, 2023.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 13.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Luka Šarčević
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1246R, 07.10.2021.
OIB studenta:	09812877327
Mentor:	izv. prof. dr. sc. Mirko Köhler
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 2:	Miljenko Švarcmajer, mag. ing. comp.
Naslov diplomskog rada:	Društvena mreža za ljubitelje satova
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Zauzeto za studenta: Luka Šarčević. Potrebno je napraviti web aplikaciju za ljubitelje satova. Korisnici nakon prijave mogu objavljivati članke o svojim satovima koji su popraćeni sa njihovim specifikacijama, opisima i maksimalno 10 fotografija. Također mogu komentirati druge članke i ocijeniti ih. Korisnik može pratiti druge korisnike mreže. Na naslovnoj stranici treba prikazati najnovije i oglase sa najvišom ocjenom u posljednjih tjedan dana.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	13.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 04.10.2023.

Ime i prezime studenta:	Luka Šarčević
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1246R, 07.10.2021.
Turnitin podudaranje [%]:	11

Ovom izjavom izjavljujem da je rad pod nazivom**: Društvena mreža za ljubitelje satova**

izrađen pod vodstvom mentora izv. prof. dr. sc. Mirko Köhler

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 studenta:

SADRŽAJ

1. UVOD 1
1.1 Zadatak diplomskog rada 1
2. POSTOJEĆA RJEŠENJA 2
2.1 WatchCrunch društvena mreža 2
2.2 Internet stranica Chrono24.com.hr 2
2.3 Internet stranica Hodinkee.com 3
2.4 Internet stranica TrustedWatch.com 4
2.5 Internet stranica Watchbox.com 5
2.6 Forumi za raspravu o satovima 5
3. TEHNOLOGIJE KORIŠTENE ZA IZRADU 7
3.1 Visual Studio Code 7
3.2 React 7
3.3 PrimeReact 7
3.4 Rust 8
3.5 Diesel 8
3.6 Actix-web 9
3.7 Docker 9
3.8 PostgreSQL 10
4. IZRADA APLIKACIJE 11
4.1 Izrada baze podataka 11
4.1.1 Opis potreba baze podataka 11
4.1.2 Definiranje entiteta i atributa baze podataka 12
4.1.3 Konceptualno oblikovanje 12
4.1.4 Fizičko oblikovanje 12
4.2 Izrada serverskog sučelja 14

4.3 Izrada korisničkog sučelja 20
5. PREGLED ZAVRŠENE APLIKACIJE 27
6. ZAKLJUČAK 35
LITERATURA 36
SAŽETAK 38
ABSTRACT 39
ŽIVOTOPIS 40
PRILOZI 41

1. UVOD

Sa velikim porastom interesa u mehaničke ručne satove diljem svijeta, potreba za raspravom o satovima također raste istom mjerom. Osim usmene komunikacije, društvene mreže dozvoljavaju trenutnu komunikaciju i dijeljenje uspomena sa ostatkom svijeta. Ljubitelji satova zbog tog razloga koriste najpopularnije društvene mreže kao što su Facebook i Twitter. Što ako se osoba ne želi zamarati nepotrebnim stvarima na društvenim mrežama, nego samo raspravljati o satovima u koje je zainteresirana? Zato je potrebna društvena mreža isključivo za ljubitelje ručnih mehaničkih satova pomoću koje će nesmetano dijeliti svoja iskustva sa ostalim ljubiteljima satova u svijetu.

Cilj završnog rada je realizirati tu ideju, tj. napraviti društvenu mrežu za ljubitelje mehaničkih satova sa svim osnovnim karakteristikama obične društvene mreže. Također, ova društvena mreža nema uobičajen oblik društvene mreže nego ima elemente portala jer se na ovoj društvenoj mreži objavljuju recenzije sata.

U drugome poglavlju prikazana su slična rješenja koja također rješavaju zadani problem. Treće poglavlje prikazuje tehnologije korištene za izradu aplikacije. U četvrtom poglavlju obuhvaćen je postupak stvaranja aplikacije koristeći navedene tehnologije. Peto poglavlje prikazuje gotovu aplikaciju i njene mogućnosti. Zadnje poglavlje predstavlja zaključak diplomskog rada u kojem se opisuju dobiveni rezultati.

1.1 Zadatak diplomskog rada

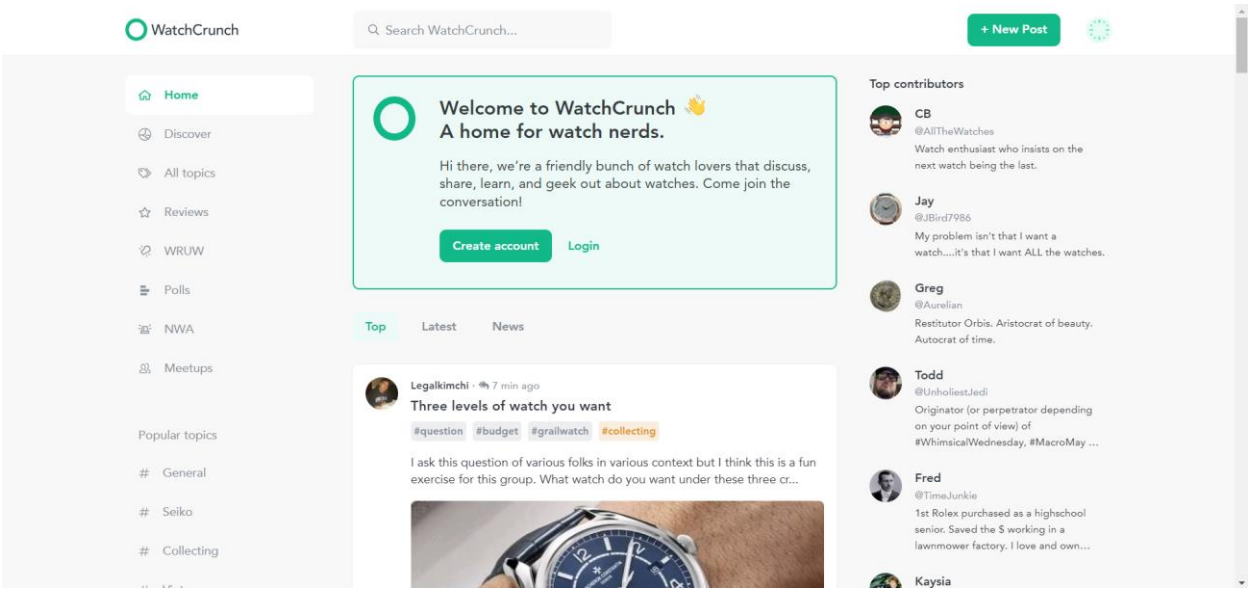
Zadatak diplomskog rada je napraviti društvenu mrežu za ljubitelje ručnih satova uz osnovne funkcionalnosti svake društvene mreže kao što su stvaranje objava (recenzije satova), praćenje autora, komentiranje objava, označavanje objava sa „Sviđa mi se“ te pretraživanje autora i objava.

2. POSTOJEĆA RJEŠENJA

S obzirom da je popularnost ručnih satova nedavno porasla na internetu, ne postoji velik broj sličnih rješenja. To nije sputavalo da se stvori trenutno jedina društvena mreža za ljubitelje ručnih satova.

2.1 WatchCrunch društvena mreža

Na slici 2.1. prikazana je početna stranica društvene mreže WatchCrunch. Ova društvena mreža blisko odgovara cilju ovoga diplomskog rada. Razlika je u tome da ovaj diplomski rad fokusira se na mehaničke ručne satove, dok na WatchCrunch-u korisnik može objavljivati o bilo kojem satu želi, bilo to alarmi za buđenje, kvarcni digitalni ili mehanički satovi. Osim toga WatchCrunch također sadrži osnovne karakteristike društvene mreže.

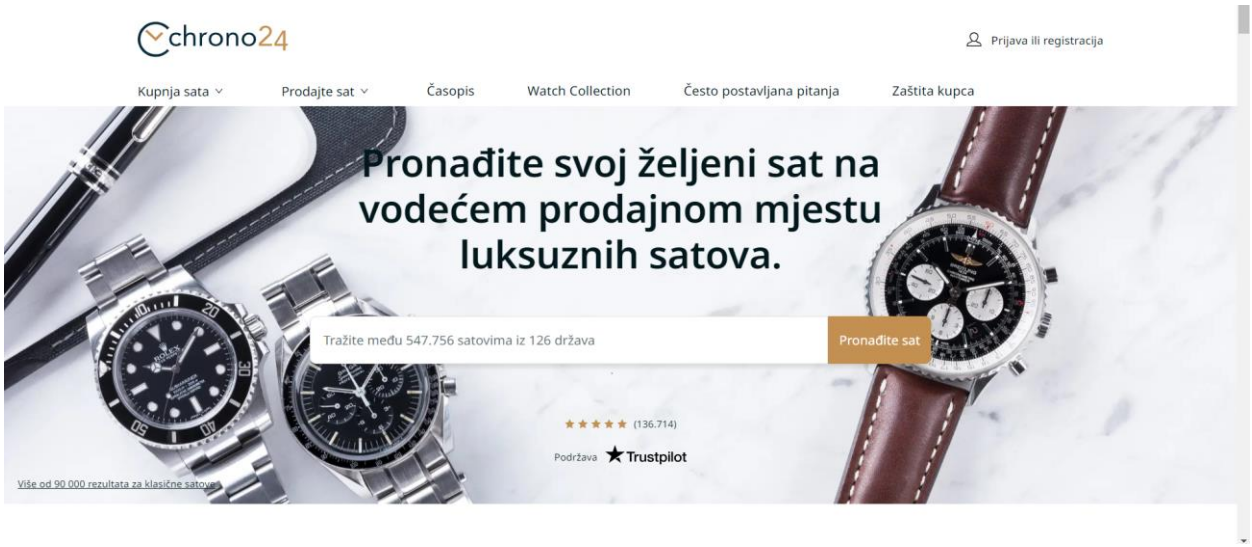


Sl. 2.1. Početna stranica društvene mreže WatchCrunch

2.2 Internet stranica Chrono24.com.hr

Internet stranica Chrono24.com.hr nudi jednostavnu prodaju i kupnju različitih stilova i vrsti satova u različitim cjenovnim rasponima. Međutim, Chrono24.com.hr nije društvena mreža. Ona nudi mogućnost komentiranja satova koji su trenutno u prodaji, ali u obliku ostavljanja osvrta na sat koji se prodaje. Stranica ne omogućuje pokretanje rasprave u kojoj mogu sudjelovati više ljudi. Također, nakon prodaje sata sve ocjene se brišu na toj objavi i ostavljene ocjene se ne čuvaju.

Navedene ocjene se premjeste na isti sat ako je u prodaji. Ključna riječ je „ako“, tj. Chrono24 ne jamči konzistentnost informacija na stranici i korisniku je teško naći informacije koje želi na istom mjestu.



Sl. 2.2. Početna stranica Chrono24.com.hr

2.3 Internet stranica Hodinkee.com

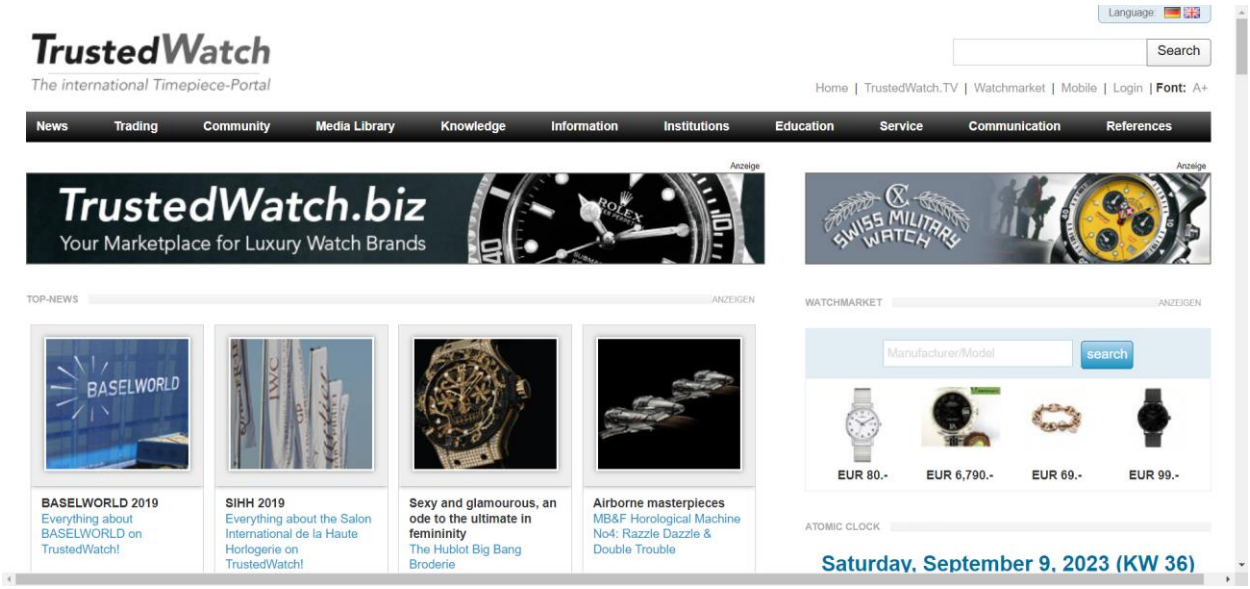
Kao što slika 2.3. prikazuje, Hodinkee.com je stranica koja sadrži novosti u svijetu satova, web trgovinu, prodaju osiguranja za ručne satove te tzv. „tržnicu“ gdje osobe mogu prodati svoje i kupiti tuđe satove.

Sl. 2.3. Početna stranica Hodinkee.com

Stranica sadrži jako puno informacija na jednom mjestu što novom korisniku može biti zbunjujuće. Nadalje, ova stranica ne sadrži određeno mjesto za recenziju satova, što je najbitnija funkcionalnost ovoga diplomskog rada.

2.4 Internet stranica TrustedWatch.com

TrustedWatch je najopširnija od svih stranica koje će biti prikazane u ovom poglavlju. Sadrži sve od prodaje satova, do recenzija satove, do obrazovnog materijala o povijesti pojedinih proizvođača itd. Kao što slika 2.4. prikazuje, najveći problem TrustedWatch stranice je zastarjeli dizajn i neintuitivnost navigacije, zbog velikog broja gumbi koji nisu podijeljeni po grupama nego su postavljeni u jednu navigacijsku traku. Sve kritike će ovaj rad poboljšati te će nuditi moderan dizajn sa lakom navigacijom.

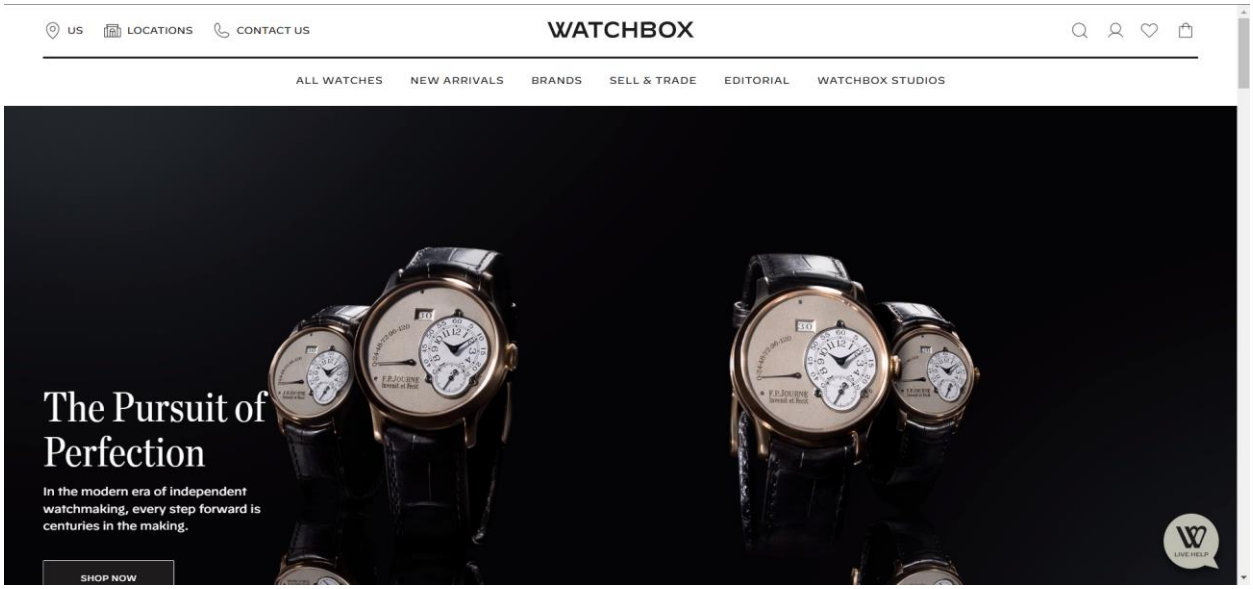


Sl. 2.4. Početna stranica TrustedWatch.com

Kao što je već naglašeno toliko različitih informacija na jednoj stranici zna biti vrlo zbunjujuće. Na stranici se čak nude i servisne usluge kao i različite institucije što je očito previše informacija za jednu web stranicu.

2.5 Internet stranica Watchbox.com

Watchbox.com je stranica za prodaju satova kao Chrono24.com.hr, ali služi samo za prodaju luksuznih i iznimno luksuznih satova. Stranica ima profesionalan dizajn i intuitivna je. Stranica nije toliko opširna kao što je Chrono24.com.hr jer nema opcije ostavljanja komentara što je bitno ako se kupuje skupocjeni sat. Nasuprot tome, stranica je strogo usredotočena na prodaju satova i konzultacijske usluge.



Sl. 2.5. Početna stranica WatchBox.com

2.6 Forumi za raspravu o satovima

Reddit kao najveća forum internet stranica svijeta, sa milijunima ljudi koji raspravljaju o različitim temama, ima i veliku zajednicu ljubitelja satova. Npr. zajednica „r/Watches“ ima preko dva milijuna članova te svaki proizvođač satova ima svoju zajednicu kao što je Seiko sa zajednicom „r/Seiko“ sa preko osamdeset tisuća članova. Postoje forumi posvećeni samo satovima kao što su „watchuseek.com“ sa više od pol milijuna članova ili „rolexforums.com“ sa malo manje od dvjesto pedeset tisuća članova i vise od dvanaest milijuna objava. Očito je da su forumi popularno mjesto za raspravu o satovima jer pružaju lagan pristup velikoj zajednici ljubitelja satova.

Ovaj diplomski rad ipak pruža standardizirani oblik pisanja recenzija i rasprava o satovima, a ne samo stranicu na koju se može pisati što god korisnik hoće.

Sl. 2.6. Stranica zajednice „r/Watches“ na Reddit.com internet stranici

3. TEHNOLOGIJE KORIŠTENE ZA IZRADU

Svaku aplikaciju koja prikazuje veliku količinu informacija koje se konstantno mijenjaju poželjno je napraviti pomoću baze podataka, programa koji komunicira s bazom podataka te ih automatski mijenja u aplikaciji kada se promijene u bazi podataka. Za izradu pristupnog sučelja (engl. *front-end*) aplikacije, tj. djela s kojim korisnik ima interakciju, korišten je React sa TypeScript-om i Primereact biblioteka komponenti, ikona i stilova. Za pozadinski sustav (engl. *back-end*) aplikacije tj. interakciju baze podataka i aplikacije koja je korisniku nevidljiva, odnosno nepoznata, korišteni su Actix-web razvojni okvir i Rust programski jezik te Docker i PostgreSQL baza podataka. Sav kôd pristupnog sučelja aplikacije i dio pozadinskog sustava napisani su u Visual Studio Code-u.

3.1 Visual Studio Code

Visual Studio Code besplatni je uređivač izvornog kôda kojeg je proizveo Microsoft. Visual Studio Code nudi ugrađenu podršku za JavaScript, TypeScript i Node.js. Također omogućuje veliki broj proširenja za druge jezike kao što su C++, Python, PHP, C# i slično. Osim proširenja, omogućuje brzu navigaciju kroz program i njegovo uređivanje, refaktoriranje i ispravak grešaka u programu te njegovo nadopunjavanje i davanje savjeta pri pisanju programa. [1]

3.2 React

React je JavaScript biblioteka koju je napravio Facebook za razvoj SPA (engl. *Single Page Application*) web stranica. [2] SPA stranice daju iluziju da se stranice mijenjaju kada korisnik navigira stranicom ali se na klijentu cijelo vrijeme prikazuje jedna stranica sa izmijenjenim sadržajem. Biblioteka se koristi za stvaranje komponenti koje se mogu ponovo iskoristiti (engl. *reusable*) na mjestima na kojima su potrebne npr. na više mjesta u aplikaciji je potreban gumb, stvara se gumb komponenta te se iskoristi na svim mjestima na kojima je potrebna. [3] Za stvaranje komponenti upotrebljavaju se funkcije te se iz tog razloga komponente se još nazivaju funkcionalne komponente.

3.3 PrimeReact

PrimeReact je biblioteka otvorenog kôda koja je slična drugoj popularnoj biblioteci Bootstrap. Sadrži gotove komponente, stilove i ikone koje se lagano mogu dodati u React aplikaciju i nude ugrađenu responzivnost. [4] PrimeReact sadrži komponente za unos podataka,

uređivač teksta, poruke (engl. *toast*), gumbe, prekidače itd. Također nudi način stiliziranja nalik Tailwind-u, još jednoj popularnoj biblioteci za stiliziranje. Stiliziranje se odvija pomoću gotovih klasa koje se dodaju u *className* parametar komponente. Na slici 3.1. se vidi primjer takvog stiliziranja.

```
<div
  key={post.post.id}
  className="header flex align-items-center justify-content-center"
>
```

Sl. 3.1. Primjer stiliziranja sa bibliotekom PrmeReact

Na HTML oznaku se primjenjuje *flexbox* CSS način prikazivanja te se preko *flexbox*-a ta oznaka vertikalno centrira sa *align-items-center* klasom i horizontalno centrira sa *justify-content-center* klasom.

3.4 Rust

Rust je programski jezik za višenamjenske svrhe orijentiran na brzinu izvođenja, sigurnost memorije i višenitnost. Kao i C programski jezik, nije objektno orijentiran, nego koristi strukture (engl. *Structures*) za držanje vrijednosti sa različitim tipovima, te funkcijama koje odrađuju posao nad tim vrijednostima. Sintaksom je nalik C-u i C++-u ali ne sadrži sakupljač smeća (engl. *Garbage collector*) nego memoriju koja je višak briše tako da broji reference koje se trenutno koriste, a ako se ne koriste briše ih iz memorije. [5] To čini Rust jezikom poželjnim za sistemsko programiranje te bilo koju situaciju u kojoj su bitne performanse, kao što je stvaranje pozadinske strane web aplikacije (engl. *back-end*).

3.5 Diesel

Diesel je Rust paket koji pruža objektno relacijsko mapiranje (engl. *ORM, object relational mapping*) iz Rust-ovih struktura u SQL tablice. Također nudi mogućnost stvaranje kompleksnih

```
                Diesel
let versions = Version::belonging_to(krate)
  .select(id)
  .order(num.desc())
  .limit(5);
let downloads = version_downloads
  .filter(date.gt(now - 90.days()))
  .filter(version_id.eq(any(versions)))
  .order(date)
  .load::<Download>(&mut conn)?;
```

```
        Raw SQL
SELECT version_downloads.*
    WHERE date > (NOW() - '90 days')
        AND version_id = ANY(
            SELECT id FROM versions
                WHERE crate_id = 1
                ORDER BY num DESC
                LIMIT 5
        )
ORDER BY date
```

Sl. 3.2. Usporedba pisanja upita u Diesel-u i SQL-u

upita u bazu podataka na jednostavan način. [6] Kao i Rust, Diesel se oslanja na brzinu izvođenja te je brži nego pisanje upita u C-u.

Upiti se pišu kao Rust kôd tako da nema potrebe za pisanje SQL-a. Diesel nudi i pisanje SQL-a sa *sql-query()* funkcijom. Slika 3.2. prikazuje razliku između SQL-a i upita koji je pisan sa Diesel-om. Diesel je konzistentan za pisanje jer se piše Rust-ovom sintaksom i to ga čini lakšim za pisanje u Rust projektima. Diesel također nudi mogućnost stvaranja i pokretanja migracija za brzo i lagano stvaranje te mijenjanje tablica u bazi podataka. [7]

3.6 Actix-web

Actix-web je razvojni okvir za pisanje web stranica stvoren za Rust programski jezik. Kao i sve što je napravljeno za Rust osnovan je na brzini i fleksibilnosti. Actix-web nudi osnovne funkcionalnosti kao što su stvaranje ruta, rukovatelja (engl. *handler*) upita na određenu rutu, međusloja (engl. *middleware*), logiranje korisnikovih poruka itd. Actix-web podržava HTTP/1 i HTTP/2 kao i HTTPS protokole. [8] U Rust projekt se dodaje preko Cargo upravitelja paketa (engl. *package manager*), kao i svi ostali paketi. U tome se razlikuje od ostalih opširnijih razvojnih okvira kao što je SpringBoot. Actix-web se može dodati u projekt kada god to programer želi. Nije potrebno dodatno postavljanje kao što je to potrebno sa SpringInitializr-om za SpringBoot. [9]

3.7 Docker

Docker je softverska platforma koja pruža usluge koje programerima pomažu u razvoju, testiranju, implementaciji i pokretanju programa na uređaju koji ima instalirano Docker. [10] Docker to čini mogućim sa tzv. spremnicima (engl. *container*) koje pokreće Docker Engine. Spremnik je izoliran od svih ostalih procesa koji su pokrenuti na uređaju jer ima svoju jezgru koja ga pokreće. U spremnik je spremljena aplikacija sa svim ovisnostima (engl. *dependency*) tj. svim alatima i paketima koji se koriste u toj aplikaciji, te se ta aplikacija može pokrenuti na bilo kojem uređaju koji podržava Docker. [11] U slučaju ovoga diplomskog rada Docker se koristi kao alat za lagano pokretanje PostreSQL spremnika. Slika 3.3. prikazuje kako se pomoću samo jedne linije pokreće PostgreSQL baza podataka u Docker-u.

```
docker run --name some-postgres -e POSTGRES_PASSWORD=mysecretpassword -d postgres
```

Sl. 3.3. Primjer pokretanja PostgreSQL spremnika u Docker-u

Pomoću ove naredbe pokreće se PostgreSQL slika (engl. *image*) koja služi kao uputa kako stvoriti Docker spremnik. Spremnik ima ime *some-postgres* te zaporku za pristup bazi podataka *mysecretpassword*. Zaporka i ime spremnika su proizvoljni.

3.8 PostgreSQL

PostgreSQL je besplatan sustav otvorenog kôda za opravljanje relacijskim bazama podataka. PostgreSQL koristi i proširuje SQL jezik sa svojim tipovima podataka i ostalim funkcionalnostima kao što su ograničen broj konekcija i njihovo upravljanje, sigurnosne mjere, sigurnosno kopiranje itd. [12] Podržan je na svim operacijskim sustavima i u većini programskih jezika što ga čini jednim od najraširenijih sustavima za upravljanje bazama podataka.

4. IZRADA APLIKACIJE

U slijedećim poglavljima prikazani su koraci izrade aplikacije. Prvo će biti prikazana izrada baze podataka, nakon toga izrada serverskog sučelja, a na kraju izrada korisničkog sučelja aplikacije.

4.1 Izrada baze podataka

Kako bi se izradila baza podataka potrebno je saznati što baza podataka obuhvaća.

4.1.1 Opis potreba baze podataka

Korisnici koji nisu prijavljeni imaju samo mogućnost pretraživanja drugih korisnika, objava i čitanje članaka. Korisniku koji je napravio korisnički račun daje se mogućnost označavanja objava sa „sviđa mi se“, komentiranja, praćenja drugih korisnika i stvaranje svojih recenzija. Recenzija sata sadržava tekst recenzije, detalje o satu (proizvođač, model, proizvođač i marka mehanizma, materijal kućišta, promjer i debljinu remena), maksimalno 10 slika sata veličine 1MB te ocjenu sata koju je dao korisnik koji recenzira sat. Komentari sadrže tekst komentara i ocjenu na sat koji se komentira tj. sat o kojemu je pisana recenzija.

U slučaju da je napisan jedan ili više komentara, na objavi se vidi ocjena korisnika koji je napisao recenziju (ocjena koja se uvijek vidi) te prosječna ocjena svih korisnika koji su komentirali. Tako čitatelji imaju uvid kako recenzent i ostatak čitatelja ocjenjuju sat.

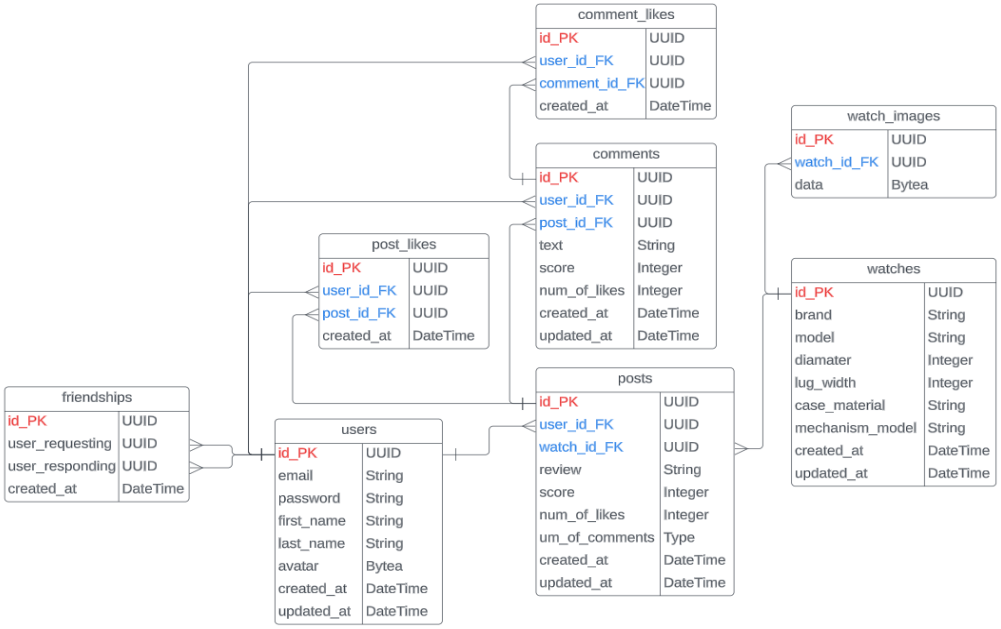
Tablica 4.1. Definirani entiteti i atributi baze podataka

Entitet	Atributi
users	*id_PK, email, password, first_name, last_name, avatar, created_at, updated_at*
posts	*id_PK, user_id_FK, watch_id_FK, review, score, num_of_likes, num_of_comments, created_at, updated_at*
watches	*id_PK, brand, model, diameter, lug_width, case_material, mehanism_model, created_at, updated_at*
watch_images	*id_PK, watch_id_FK, data*
comments	*id_PK, user_id_FK, post_id_FK, text, score, num_of_likes, created_at, updated_at*
friendships	*id_PK, user_requesting, user_responding, created_at*
post_likes	*id_PK, user_id_FK, post_id_FK, created_at*
comment_likes	*id_PK, user_id_FK, comment_id_FK, created_at*

4.1.2 Definiranje entiteta i atributa baze podataka

U tablici 4.1 su definirani svi entiteti i atributi koji pripadaju tim entitetima.

4.1.3 Konceptualno oblikovanje



Sl. 4.1. E/R dijagram

Nakon analize zahtjeva i definicije svih entiteta, atributa i veza, dobiven je E/R dijagram na slici 4.1. u kojem se vide veze između entiteta te pojedini atributi koji pripadaju određenim entitetima.

Veza koja ima više entiteta je prikazana sa tri crtice na kraju veze, dok strana veze koja ima jednu okomitu crticu označava kako postoji samo jedan takav entitet. Crvenom bojom su označeni primarni ključevi entiteta dok su strani ključevi označeni plavom bojom radi lakše prepoznatljivosti. Svi primarni ključevi su tipa UUID (engl. *Universally unique identifier*). Atributi koji predstavljaju slike kao što su „*avatar*“ u entitetu „*user*“ i „*data*“ u entitetu „*watch*_images“ su tipa Bytea što je PostgreSQL tip podatka i služi za spremanje niza okteta tj. bajtova (engl. *byte*).

4.1.4 Fizičko oblikovanje

Za fizičko oblikovanje upotrijebljen je Diesel paket za Rust programski jezik. Diesel nudi stvaranje migracija kroz svoj CLI (engl. *command line interface*). CLI mora se instalirati na računalo pomoću naredbe koja je prikazana na slici 4.3. 2. Instalacija se odrađuje preko Cargo

```
cargo install diesel_cli --no-default-features --features postgres
```

Sl. 4.2. Primjer stvaranja migracije

upravitelja paketa za Rust. Instaliraju se samo značajke za PostgreSQL bazu podataka jer su one jedine potrebne.

```
diesel migration generate create_posts
```

Sl. 4.3. Primjer stvaranja migracije

Nakon instalacije Diesel CLI-a mora se pokrenuti „*setup*“ naredba. Preduvjet stvaranju migracija je već postojeći pokrenuti PostgreSQL spremnik što je opisano u poglavlju 3.7. Osim toga, mora biti stvoren Rust projekt što će biti opisano u poglavlju 4.2.

Naredba „*diesel setup*“ stvara prazan „*migrations*“ direktorij u kojemu će stajati sve stvorene migracije. Stvaranje migracije prikazano je na slici 4.3.

U „*migrations*“ direktoriju stvoren je novi direktorij sa imenom migracije i dvije datoteke „*up.sql*“ i „*down.sql*“. U „*up.sql*“ se piše SQL koji se želi primijeniti na bazu, a „*down.sql*“ se koristi ako se nešto želi obrisati iz baze podataka. Na taj način, kroz migracije, nema potrebe za brisanjem cijele baze nego se kroz vrijeme baza može mijenjati, a uz to se vidi tko je uveo promjene i kada su one uvedene.

```
diesel::table! {
    comment_likes (id) {
        #[max_length = 36]
        id -> Varchar,
        #[max_length = 36]
        user_id -> Varchar,
        #[max_length = 36]
        comment_id -> Varchar,
        created_at -> Timestamptz,
    }
}
```

Sl. 4.4. Primjer stvaranja migracije

Kada su sve migracije stvorene i napisan je SQL u svim „*up.sql*“ i „*down.sql*“ datotekama, migracije se pokreću naredbom „*diesel migraion run*“. Uspješnost migracija se može provjeriti u „*schema.rs*“ datoteci. U toj su datoteci preko Diesel-a u Rust-u napisane sve tablice i njihovi odnosi. Ostatak Rust programa gleda u „*shema.rs*“ datoteku kada se radi upit u bazu podataka. Primjer tablice u „*schema.rs*“ datoteci je na slici 4.4. Tablica je pisana u Rust-u ali tipovi podataka su PostgreSQL-ovi jer ih se uvozi iz Diesel paketa. Na kraju cijela baza podataka i sve njene tablice mogu se pogledati u DBeaver besplatnom alatu za administraciju bazi podataka.

4.2 Izrada serverskog sučelja

Serversko sučelje napravljeno je prema heksagonalnoj arhitekturi ili na engleskom „*Ports and Adapters*“. Rust kôd je podijeljen na direktorij biblioteka i na binarni direktorij. Direktorij biblioteka sadrži više biblioteka (engl. *library*) koje se mogu više puta iskoristiti u kôdu kao što je upravljanje greškama, modeli tablica iz baze podataka i metode nad tim tablicama te konfiguraciju aplikacije. Binarni direktorij sadrži „*main.rs*“ funkciju i ostatak glavne logike aplikacije kao što su rukovatelji (engl. *handler*) upita. Svi direktoriji su podijeljeni u module. Svaki direktorij sadrži „mod.rs“ datoteku u koju su zapisane sve datoteke u tom direktoriju. Taj model pisanja kôda pruža modularnu arhitekturu, očuvanje područja u kojemu je kôd dostupan (engl. *scope*) i vidljivost struktura ili funkcija u datotekama (slično kao vidljivost u OOP jezicima).

U glavnom direktoriju se uz „*main.rs*“ nalazi „*web.rs*“ datoteka u kojoj je definiran i postavljen server te CORS pravila. Na slici 4. prikazana je definicija servera u „*web.rs*“ datoteci. Prilikom stvaranja servera mora se definirati kojim podatcima server ima pristup, a to se odrađuje sa *.app_data()* funkcijom. U nju se predaje funkcija *initialize()* iz biblioteke „infrastructure“. U

```
HttpServer::new(factory: move|| {
    App::new() App<AppEntry>
        // Initialize global state
        .app_data(ext: infrastructure::state::initialize()) App<AppEntry>
        .wrap(mw: setup_cors()) App<impl ServiceFactory<ServiceRequest, Config = …, Response = …, Error = …, InitError = …>>
        .configure(|cfg: &mut ServiceConfig| crate::application::configure(postgres: Arc::clone(self: &postgres), cfg))
}) HttpServer<impl Fn() -> App<…>, …, …, …>
.workers(num: workers) HttpServer<impl Fn() -> App<…>, …, …, …>
.bind(addrs: address)? HttpServer<impl Fn() -> App<…>, …, …, …>
.run() Server
.await
```

Sl. 4.5. Stvaranje servera u „*web.rs*“ datoteci

initialize() funkciji definira se glavna struktura *AppState* te funkcija *postgres_connection()* koja vraća konekciju na bazu podataka, što je vidljivo na slici 4..

```
#[derive(Clone)]
2 implementations
pub struct AppState {···

impl AppState {
    pub fn postgres_connection(&self) -> DbConnection {
        match self.static_state.postgres.get() {···
    }
}
```

Sl. 4.6. *AppState* struktura sa *postgres_connection()* funkcijom

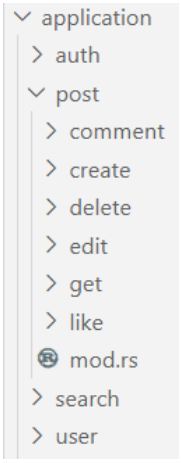
Dva direktorija se nalaze u glavnom binarnom direktoriju, a to su „application“ i „middleware“ direktoriji. Direktorij „middleware“ objašnjiv je sam po sebi tj. sadrži međusloj koji presreće dolazeće zahtjeve. Dolazeći zahtjevi se validiraju tj. provjerava se je li JWT žeton (engl. *JSON web token*) ispravnog oblika i vrijedi li. Ako žeton vrijedi upit se šalje rukovatelju. Ako žeton ne vrijedi korisniku se šalje greška sa kôdom *401* ili *403* ovisno do koje je greške došlo. Dio opisanog međusloja je prikazan na slici 4..

```
let mut split: SplitWhitespace<'_> = header.split_whitespace();
let auth_type: Option<&str> = split.next();
if Some("Bearer") == auth_type {
    if let Some(token: &str) = split.next() {···
    } else {
        Err(Error::Unauthorized("No jwt token".to_string()))
    }
} else {
    Err(Error::Forbidden("No bearer scheme with jwt token".to_string()))
}
```

Sl. 4.7. Upravljanje grešaka u međusloju

Aplikacija koristi JWT žeton za autentifikaciju korisnika sa *Berarer* shemom. *Bearer* shema ili shema nositelja označava da se nositelju žetona dozvoli pristup resursu. [13] JWT žetoni odlično rade sa *Bearer* shemom te se predaju u HTTP zaglavlju „*Authorization*“ sa shemom „Bearer <token>“ gdje <token> označava mjesto za stvoreni JWT žeton. Ako ne postoji Bearer shema, međusloj vraća 403 tj „*Forbidden*“. U slučaju da žeton nije verificiran ispravno, međusloj vraća 401 „*Unauthorized*“. Oba slučaja prikazana su na slici 4.7.

Direktorij „*application*“ sadrži glavnu logiku cijele aplikacije. U njemu se nalaze svi rukovatelji upita. Rukovatelji su podijeljeni na logičke cjeline što je prikazano na slici 4..



Sl. 4.8. Struktura „*application*“ direktorija

Na najnižoj razini, u svakom direktoriju pojedine funkcionalnosti nalaze se datoteke i direktorij: direktorij „*test*“, „*contract.rs*“, „*data.rs*“, „*domain.rs*“, „*http.rs*“, „*infrastucture.rs*“, „*mod.rs*“ i „*setup.rs*“ što se vidi na slici 4.2.5.

Sl. 4.2.5. Struktura „*edit*“ direktorija na najnižoj razini

U „*setup.rs*“ datoteci nalazi se konfiguracija servisa za pojedinu ulogu u aplikaciji. U servisu se postavljaju putanje (engl. *route*), međusloj i rukovatelj upita što se vidi na slici 4..

```
cfg.service(
    factory: web::resource(path: "/comments/{comment_id}/like") Resource
    .route(post().to(handler: handle_like_comment::<···
    .wrap(mw: crate::middleware::AuthLogin)
);
```

Sl. 4.10. Definiranje servisa i „setup.rs“ datoteci

Konfiguracija se odrađuje u funkciji „*routes()*“. Također, u ovoj datoteci se definira metoda koja je prihvatljiva pri upitu na ovaj servis. U tom slučaju to je *POST* te parametar putanje označen u vitičastim zagradama pod „{comment_id}“.

Datoteka „contract.rs“ sadrži sve *'kontrakte'* tj. ugovore u obliku svojstava (engl. *trait*). Svojstva u Rust-u su nalik sučeljima u OOP jezicima. U svojstvima se definiraju metode domene koja sadrži glavnu logiku svakog upita i metode repozitorija koji komunicira sa bazom podataka kao što je prikazano na slici 4..

Datoteka „*data.rs*“ sadrži sve strukture koje predstavljaju DTO (engl. *data transfer object*) strukture tj. strukture sa podatcima koja se dobivaju sa klijentske strane i šalju nazad klijentu.

```rust
pub trait CreateCommentContract {
    async fn create_comment(&self, user_id: &str, post_id: &str, comment_data: UserCommentData) -> Result<DisplayComment, Error>;
}

#[cfg_attr(test, mockall::automock)]
#[async_trait]
pub trait PgRepositoryContract {
    async fn create_comment(&self, comment_data: CreateNewCommentData) -> Result<Comment, Error>;
    async fn increment_posts_comment(&self, post_id: &str) -> Result<usize, Error>;
}
```

Sl. 4.11. Definiranje funkcija u „contract.rs“ datoteci

Strukture su validirane sa *validr* Rust paketom. Validacija podataka služi da se provjeri jesu li svi podatci koji su došli sa klijentske strane ispravno strukturirani te da su ispravnog oblika i tipa.

Datoteka „*http.rs*“ sadrži rukovatelja upita na pozadinsku stranu aplikacije. Poslije međusloja rukovatelj je prvi koji ima doticaj sa upitom. Ako je putanja omotana u međusloj, rukovatelj iz upita može izvaditi informacije o korisniku koje su dekodirane iz JWT žetona. Nadalje, ako putanja sadrži dinamičke parametre oni se izvlače iz putanje i spremaju u posebnu varijablu. Zadnja stvar koja se odrađuje je validacija podataka ako se nalaze u upitu. Nakon toga poziva se funkcija iz domene i njoj se kao parametri šalju podatci potrebni za stvaranje odgovora klijentu. Vraćeni podatci iz domene se šalju klijentu u obliku HTTP koda i JSON podataka ako su potrebni što je vidljivo na slici 4.12.

```rust
let Some(user: DisplayUser) = req.extensions_mut().remove::<DisplayUser>() else {
    return Err(Error::Unauthorized("Not authorized".to_string()));
};

let data: UserCommentData = data.into_inner().validate()?;
let post_id: String = part_from_path::<String>(&req, name: "post_id")?;

let response: DisplayComment =
    service.create_comment(user_id: &user.id, &post_id, comment_data: data).await?;

Ok(HttpResponse::Created() HttpResponseBuilder
    .json(response)
)
```

Sl. 4.12. Glavna logika u rukovatelju iz „http.rs“ datoteke

Datoteka „*domain.rs*“ sadrži glavnu logiku svakog upita. Funkcija iz „*domain.rs*“ datoteke prima podatke iz rukovatelja, obavlja transformacije podataka ako su potrebne te ako je potreban upit u bazu podataka poziva funkcije iz repozitorija. Na slici 4.. vidi se da se zaporka pretvara u „hash“ te se sa tom zaporkom stvara novi korisnik. Korisnik se stvara tako da se poziva servis koji predstavlja repozitorij i njegove funkcije. To je asinkrona funkcija pa se mora sačekati sa *.await* operatorom. Nakon uspješnog stvaranja korisnika, baza vraća cijeli objekt korisnika pa ga se pretvara u tip *DisplayUser* gdje nema zaporke i korisnik ostane siguran. Na kraju se stvori JWT žeton iz korisnikove strukture *auth_user* i vraćaju se rukovatelju.

```
data.password = Some(User::hash_password(&password)?);

let auth_user: DisplayUser = self.service B
    .create_user(data.insertable())
    .await Result<User, Error>
    .map(DisplayUser::from)?;

let token: String = User::generate_jwt_token(&auth_user)?;

Ok(AuthDataResponse {
    user: auth_user,
    token
})
```

Sl. 4.13. Glavna logika u rukovatelju iz „http.rs“ datoteke

Datoteka „*infrastructure.rs*“ sadrži sve implementacije repozitorija. Repozitorij služi kao komunikacija sa bazom podataka koji se poziva u funkcijama unutar „*domain.rs*“ datoteke. Kao što je prikazano na slici 4. najčešće se pozivaju funkcije definirane na modelu iz biblioteke jer se preko modela komunicira sa bazom podataka.

```
#[async_trait]
impl PgServiceContract for PgService {
    async fn create_user(&self, data: CreateNewUserData) -> Result<User, Error> {
        User::create(data, self.pg_pool.connection()?)
    }
}
```

Sl. 4.14. Pozivanje *create()* funkcije na User modelu u „infrastructure.rs“ datoteci

Pri složenijim upitima, npr. pri paginaciji odgovora iz upita, ne poziva se model, nego se upit stvara izravno u „*infrastructure.rs*“ datoteci kao što je vidljivo iz slike 4..

```
let mut query =
    comments::table
        .left_join(users::table.on(users::id.eq(comments::user_id)))
        .left_join(comment_likes::table.on(
            comments::id.eq(comment_likes::comment_id)
            .and(comment_likes::user_id.eq(&user_id))
        ))
        .into_boxed();

query = query
    .filter(comments::post_id.eq(post_id))
    .order(comments::created_at.desc());

query
    .page(attributes.page)
    .per_page(attributes.per_page)
    .paginate(&mut conn)
    .map_err(Error::from)
```

Sl. 4.15. Stvaranje upita u funkciji *get_post_comments_paginated()* unutar „*infrastructure.rs*“ datoteke

Diesel nudi laku mogućnost spajanja (engl. *join*) tablica po glavnim ključevima. Varijabla *query* se označava sa *mut* jer se naknadno mijenja i dodaje filtar te se rezultati redaju po vremenu stvaranja. Na kraju se poziva *paginate()* metoda na *query* varijabli i njoj se predaje konekcija na PostgreSQL. PostgreSQL prema zadanim postavkama prima samo 10 konekcija na bazu. Taj broj se može mijenjati pri postavljanju servera, ali samo na manje od 10 konekcija. Ako se broj konekcija želi povećati mora se mijenjati konfiguracijska datoteka PosgreSQL-a.

Kao što je rečeno modeli se definiraju u biblioteci. Na slici 4.. vidi se definicija modela *User*.

```
#[derive(Insertable, Queryable, Serialize, Deserialize, Identifiable, Selectable, PartialEq, Debug, Clone)]
#[diesel(table_name = users)]
#[diesel(treat_none_as_null = true)]
#[serde(rename_all = "camelCase")]
13 implementations
pub struct User {
    pub id: String,
    pub email: String,
    pub first_name: String,
    pub last_name: String,
    pub password: String,
    pub avatar: Vec<u8>,
    pub created_at: NaiveDateTime,
    pub updated_at: NaiveDateTime,
}
```

Sl. 4.16. *User* model u biblioteci

Na vrhu se nasljeđuju Diesel svojstva kao što su *Insertable*, *Queriable* te *Serialize* i *Deserialize* iz *serde* Rust paketa. *Serde* paket služi da se struktura 'serijalizira' u JSON format i 'deserijalizira' iz JSON formata kada se struktura šalje kao odgovor ili dobiva kao upit. U drugoj liniji koda određuje se koju tablicu u bazi podataka će struktura predstavljati. Ta struktura služi samo za slanje i dobivanje podataka u bazu i iz baze, a tablica iz „*schema.rs*“ datoteke služi za komunikaciju sa tablicama iz baze podataka kao što se vidi na slici 4.17. Tablice iz „*schema.rs*“ datoteke pišu se malim slovima i tako se mogu na slikama prepoznati.

Primjer metode za stvaranje korisnika također se vidi na slici 4..

```
pub fn create(data: CreateNewUserData, mut connection: DbConnection) -> Result<User, Error> {
    diesel::insert_into(target: users::table) IncompleteInsertStatement<…>
        .values(records: data) InsertStatement<table, <(…) as Insertable<…>>::Values>
        .get_result::<User>(conn: &mut connection) Result<User, Error>
        .map_err(op: Error::from)
}
```

Sl. 4.17. Funkcija *create()* na *User* modelu

Tablica *users* predaje se u *insert_into()* funkciju. Nakon toga predaju se vrijednosti koje želimo ubaciti u tablicu preko *values()* funkcije. *Get_result::<User>()* funkcija koristi se za dobivanje rezultata koji se predaju kao tip te funkcije, u ovom slučaju to je *User* struktura. Na kraju se poziva *map_err()* funkcija koja u slučaju pojave greške poziva upravitelj grešaka iz biblioteke. Upravitelj grešaka uzima grešku iz Rust-ovih paketa i pretvara ih u standardizirani oblik kao što je vidljivo na slici 4.18.

```rust
pub struct ErrorBody {
    pub message: Option<String>,
    pub code: String,
    pub cause: Option<String>,
    pub payload: Option<serde_json::Value>,
}
```

Sl. 4.18. *ErrorBody* struktura koja predstavlja standardizirani oblik greške

U tijelu greške *message* atribut predstavlja poruku koja se može predati korisniku, *code* je HTTP status kod, *cause* je uzrok greške koju programer može spremiti u grešku te *payload* koji predstavlja JSON objekt koji se može poslati korisniku.

4.3 Izrada korisničkog sučelja

Osnova React aplikacije je *package.json* datoteka koja sadrži sve pakete i podatke koji su potrebni za pokretanje aplikacije. Osim *package.json* datoteke bitna je *main.tsx* datoteka u koju se dodaju sve vanjske biblioteke kao što je PrimeReact. U njoj se poziva *App.tsx* komponenta koja predstavlja cijelu aplikaciju, tj. sve stranice aplikacije. Uz te datoteke još postoje direktoriji: „*api*“, „*components*“, „*layouts*“, „*pages*“, „*provider*“, „*routes*“ i „*utils*“. Svaki direktorij će biti detaljnije pojašnjen u slijedećim poglavljima.

Za sigurno korištenje aplikacije potrebno je napraviti kontekst (engl. *context*) koji služi kao centralno spremište informacija te mu se iz bilo kojeg dijela aplikacije može pristupiti. Kontekst se nalazi u „*provider*“ direktoriju. Informacije koje će se spremiti su JWT žeton i *User* objekt dobiven sa *back-end-a*. Svaki kontekst ima opskrbljivača (engl. *provider*) koji taj kontekst otkriva ostalim komponentama. Kao i sve ostalo u React-u kontekst je funkcijska komponenta te vraća JSX u *return()* naredbi. Funkcijska komponenta može sadržavati druge funkcije koje odrađuju

```tsx
const [token, setToken] = useState<string | undefined | null>(
  sessionStorage.getItem("token")
);
```

Sl. 4.19. *useState* primjer u *AuthProvider* komponenti

neki posao sa „stanjem“ konteksta (engl. *state*). „Stanje“ predstavlja objekt koji sprema informacije bitne za tu komponentu. To stanje se može lako mijenjati sa *setState()* metodom unutar te komponente, kao što je prikazano na slici 4.19.

Opskrbljivač može otkriti metode definirane u kontekstu što se vidi na slici **Error! Reference source not found.** *Children* u vitičastim zagradama predstavlja da se bilo koja komponenta koje je dijete ove komponente može služiti sa svime što joj opskrbljivač pruža na korištenje. Na slici 4.20. vidi se kako cijela aplikacija ima pristup vrijednostima koje pruža opskrbljivač na slici.

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

Sl. 4.20. Pružanje vrijednosti djeci opskrbljivača

U *App.tsx* komponenti definirane su sve rute aplikacije pomoću *BrowserRouter* komponente. *BrowserRouter* omogućuje navigiranje unutar aplikacije i usklađivanje ruta sa korisničkim sučeljem. Unutar *BrowserRouter-a* postavljena je *Routes* komponenta koja prolazi kroz sve dostupne rute. Pomoću *map()* metode svaku dostupnu rutu postavlja se u *Route* komponentu, koja spada u *react-router-dom* paket kao što je prikazano na slici 4.. Opisano rutiranje je postavljeno u „*routes*“ direktorij.

```
{RouteConfiguration.map((route) => {
  if (canShowRoute(token, route.visibility))
    return (
      <Route
        element={<route.component />}
        key={Math.random()}
        path={route.path}
      />
    );
  else return null;
})}
```

Sl. 4.21. Prolaženje kroz rute i stvaranje *Route* komponenti

U „*layouts*“ direktoriju nalazi se *Default.tsx* komponenta koja predstavlja osnovni poredak komponenti na ekranu (engl. *layout*). To je jednostavna komponenta koja na vrh ekrana uvijek stavlja navigacijsku traku. Ona kao i kontekst prima varijablu *children* te ih postavlja ispod navigacijske trake. Varijablu *children* komponenta *DefaultLayout* dobiva kao svojstvo (engl.

```
function DefaultLayout({ children }: props) {
  return (
    <>
      <NavBar />
      {children}
    </>
  );
}
```

Sl. 4.22. Primjer svojstva u funkcijskoj komponenti

prop). Svojstvo se predaje kao u običnu funkciju te se u funkcijskoj komponenti može koristiti kao parametar funkcije u C-u što se vidi na slici 4..

Direktorij „*pages*“ sadrži sve stranice koje su dostupne u aplikaciji. *DefaultLayout* komponenta postavljena je na sve stranice, osim stranice za prijavu i registraciju korisnika jer tamo navigacijska traka nije potrebna. U *Page* komponentama postavljaju se komponente iz „*components*“ direktorija.

Direktorij „*components*“ sadrži sve komponente koje se koriste u izradi aplikacije, tj. ono što se vidi na ekranu te čini najveći direktorij aplikacije. Direktorij je podijeljen na više podvrsta, npr. „*forms*“, „*lists*“, „*navigation*“ i sl. Također, navedene podvrste mogu se dijeliti dalje na manje podvrste. Cilj takve arhitekture je da se stvori što manja komponenta koja će se koristiti više puta u aplikaciji. Za primjer može se uzeti komponenta *PostList.tsx* koja sadrži listu svih objava. Kako bi se smanjila količina koda koji se mora pisati, pomoću *map()* funkcije se prolazi kroz svaku objavu i objekt pojedine objave se predaje *PostItem.tsx* komponenti koja je zaslužna za prikaz pojedine objave, što se vidi na slici 4.23.

```
{posts?.map((data) => {
  return <PostItem post={data} />;
})}
```

Sl. 4.23. Ispisivanje cijele liste pomoću *map()* funkcije

Liste se prikazuju kao beskonačno listanje (engl. *infinite scroll*) tj. korisnik ne treba klikati na broj stranice da se učitaju nove objave, nego se objave učitaju kada se listanjem dođe do kraja trenutnih objava. Na isti način su implementirane sve liste u aplikaciji kao što je lista komentara, što se vidi na slici 4..

Svojstvo *next* u komponenti *InfiniteScroll* služi za pozivanje funkcije *handleFetchPostComments()* koja učitava slijedeću stranicu komentara te ih sprema u listu komentara koja je varijabla stanja (engl. *state*) u komponenti *CommentList*. Na kraju uspješnog

izvođenja funkcije *handleFetchPostComments()*, stranica koja je pri učitavanju komponente *CommentList* postavljena na 1, povećava se za 1 i tako se može učitati slijedeća stranica.

Slika 4.. također prikazuje uvjetno prikazivanje komponenti (engl. *conditional rendering*) koje omogućuje JSX vrsta datoteka. U JSX ili TSX datotekama (ako se koristi Typescript kao u ovom slučaju) varijable koje su dostupne unutar komponente mogu se postaviti u *return()* naredbu komponente. To omogućuje postavljanje uvjeta pomoću „&&“ operatora ili „? : :“ ternarnog operatora. Komponente se mogu prikazati na temelju istinitosti tih uvjeta, npr. ternarni operator na varijabli *isLoading*. Ako je *isLoading* istinit prikazuje se komponenta *ProgressSpinner*, a ako nije prikazuje se lista komentara.

```
return isLoading ? (
  <ProgressSpinner />
) : (
  <InfiniteScroll
    next={() => {
      handleFetchPostComments();
    }}
    hasMore={Math.ceil(total / 15) === page}
    loader={<p></p>}
    endMessage={
      <p style={{ textAlign: "center" }}>···
      </p>
    }
    dataLength={total}
    scrollableTarget={"post-card-item"}
  >
    <div className="overflow-scroll scroll-container" id="post-card-item">
    </div>
  </InfiniteScroll>
);
```

Sl. 4.24. Implementacija beskonačnog listanja liste komentara

Pri prvom prikazu komponenti, *useEffect()* funkcija (engl. *hook*) se automatski pokreće i učitava prvu stranicu komentara te ih sprema u stanje *comments* koje je niz komentara. Funkcija *useEffect()* može imati ovisnu varijablu (engl. *dependency*) te se *useEffect()* pokreće nakon svake promjene te varijable kao što se vidi na slici 4.25. Kada se *createdComment* varijabla promjeni pokreće se *useEffect()*.

```
useEffect(() => {
  handleFetchPostComments(1);
// eslint-disable-next-line react-hooks/exhaustive-deps
}, [createdComment]);
```

Sl. 4.25. Recikliranje jedne komponente za sve objave u listi

Lista komentara se ponovo učitava od početne stranice kada se stvori novi komentar jer se novi komentar želi ubaciti u postojeću listu.

Pozivi na serversko sučelje odvijaju se preko axios-a. Axios je HTTP klijent koji služi za stvaranje upita iz preglednika. Axios omogućuje stvaranje instance koja se može konfigurirati. Najvažnija konfiguracija su presretači (engl. *interceptor*). Presretači se mogu postaviti na zahtjev na serversko sučelje i odgovor sa serverskog sučelja. Stvaranje i konfiguriranje axios instance smanjuje količinu koda koji se mora napisati. Na slici 4.26. vidi se presretač na zahtjev, koji u slučaju da je korisnik prijavljen u aplikaciju i da JWT žeton prisutan u klijentu, priprema HTTP zaglavlje „Authorization“ i postavlja žeton u *Bearer* shemu te šalje zahtjev na serversko sučelje.

```
axiosInstance.interceptors.request.use((request) => {
  if (token) {
    request.headers["Authorization"] = `Bearer ${token}`;
  }
  return request;
});
```

Sl. 4.26. Presretač za zahtjev na axios instanci

Axios instanca stvorena je u prilagođenoj React funkciji (engl. *hook*). Prilagođene tj. korisničke React funkcije stvaraju se na način da počinju sa riječi „*use*…“ te se nastavlja ime funkcije koju programer želi. U ovom slučaju najviše smisla ima *useAxios()*. Prilagođeni „*hook*“ je koristan jer svo stanje i utjecaji na to stanje su kompletno izolirani od ostatka programa.

Na slici 4.27. prikazan je axios poziv na serversko sučelje koji dohvaća objavu sa određenim identifikatorom.

```
axiosInstance
  .get(PostRoutes.USER_POST(postId))
  .then((response) => {
    setPostData(response.data);
    setLiked(response.data.post.isLikedByUser);
    setLikeCount(response.data.post.numOfLikes);
  })
  .catch((error) => {
    console.error(error);
  })
  .finally(() => {
    setLoading(false);
  });
```

Sl. 4.27. Axios poziv na *back-end*

Preko *axiosInstance* varijable, koja je rezultat *useAxios()* prilagođenog „*hook-a*“, poziva se metoda GET za dohvaćanje resursa. U GET metodu predaje se ruta sa koje želimo dohvatiti resurs. Sve rute su u datoteci „*endpoints.ts*“ i sadrže više varijabli. Jedna od njih je vidljiva na slici 4.27. i zove se PostRoutes. Ona sadrži sve rute koje se tiču objave. U ovom slučaju ruta je nazvana *USER_POST* i predaje joj se identifikator objave. Način stvaranja rute vidljivo je na slici 4..

```
USER_POST: (postId?: string) => `posts/${postId}`,
```

Sl. 4.28. Stvaranje rute za dohvaćanje objave sa određenim identifikatorom

USER_POST varijabla je „*arrow*“ funkcija koja prima identifikator i pomoću „šablon“ string-a (engl. *template string*) ubacuje se u putanju.

Pošto je axios poziv asinkron, odgovor se čeka u *then()* bloku, i ako je upit uspješan odgovor se destrukturira i sprema u odgovarajuće varijable stanja. U ovom slučaju to su *postData*, *liked*, *likeCount* varijable. Ako odgovor sadrži grešku, sa njom se upravlja u *catch()* bloku. U *catch()* blok ulaze samo greške koje su namjerno puštene iz prilagođene axios instance zbog prikaza grešaka korisniku. U ovom slučaju bilo koja greška koja bi do korisnika došla nije štetna i greška se ispisuje u konzolu zbog evidencije. Blok *finally()* se uvijek izvršava na kraju upita i to je najbolje mjesto za postavljanje boolean stanja *isLoading* na *false* što označava da je upit gotov.

Najbitnije komponente u ovom diplomskom radu su obrasci. Obrasci znaju biti kompliciran kada imaju puno unosa, ali Formik znatno olakšava upravljanje obrazaca. Formik je besplatna biblioteka za React koja koristi svoje komponente za stvaranje, upravljanje, osvježavanje i slanje obrazaca na intuitivan način. Na slici 4.. prikazan je jednostavan obrazac napravljen pomoću Formik-a.

```
<Formik
  initialValues={initialValues}
  onSubmit={(values) => {
    handleSearchUser(values);
  }}
>
  {({…
  }) => (
    <form onSubmit={handleSubmit} className="flex">
      <div className="flex w-full p-input-icon-right">
        <i
          className="pi pi-times cursor-pointer"
          onClick={() => setFieldValue("searchTerm", "")}
        />
        <InputText
          id="searchTerm"
          name="searchTerm"
          value={values.searchTerm}
          onChange={handleChange}
        />
      </div>
      {checkErrors(errors, touched, "searchTerm")}

      <Button className="ml-2" type="submit" icon="pi pi-search" />
    </form>
  )}
</Formik>
```

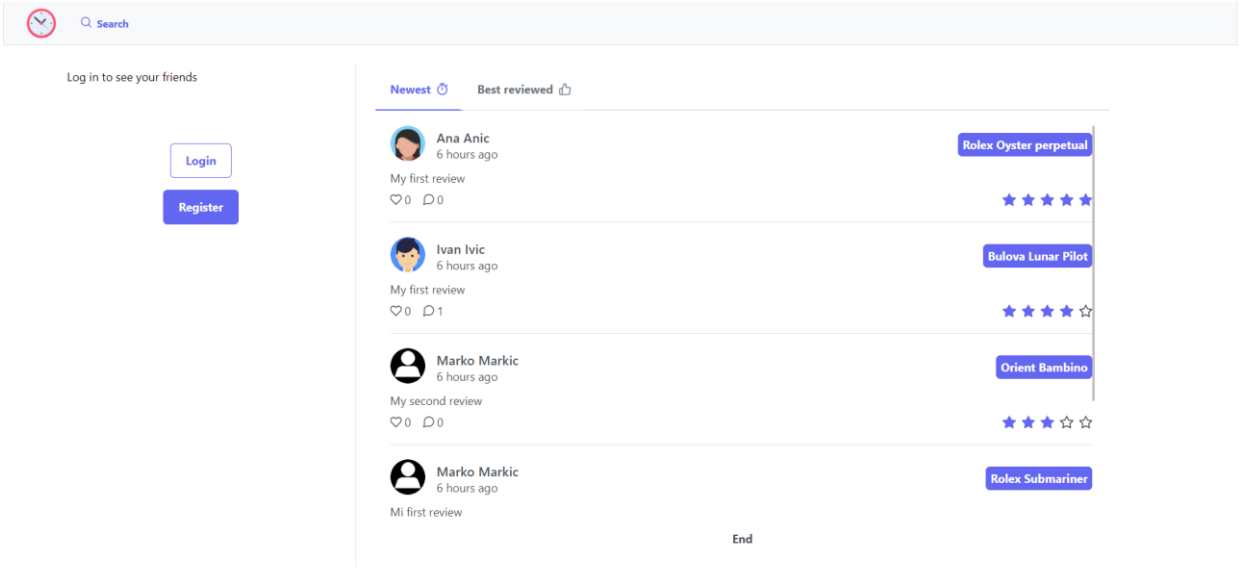
Sl. 4.29. Primjer obrasca napravljene sa Formik-om

Glavna komponenta *Formik* predstavlja konfiguraciju obrasca. Samo dva svojstva su bitna, a to su *initialValues* i *onSubmit*. *InitialValues* predstavlja objekt sa svim početnim vrijednostima koje obrazac prima, u ovom slučaju to je objekt sa jednim atributom tipa *string* jer obrazac sadržava samo jedan unos teksta. Početna vrijednost je prazan *string*. *InputText* je komponenta iz biblioteke *PrimeReact* koja odlično radi sa Formik-om. Formik preko svojstva *name* prepoznaje komponentu i mijenja joj vrijednosti preko *handleChange* funkcije koju pruža Formik. Formik također prepoznaje gumb koji je tipa „*submit"* te klikom na gumb automatski poziva *handleSearchUser()* funkciju i predaje vrijednosti forme u *values* objektu. Na gumb „*submit"* se pokreće i validacija podataka obrasca pomoću paketa Yup.

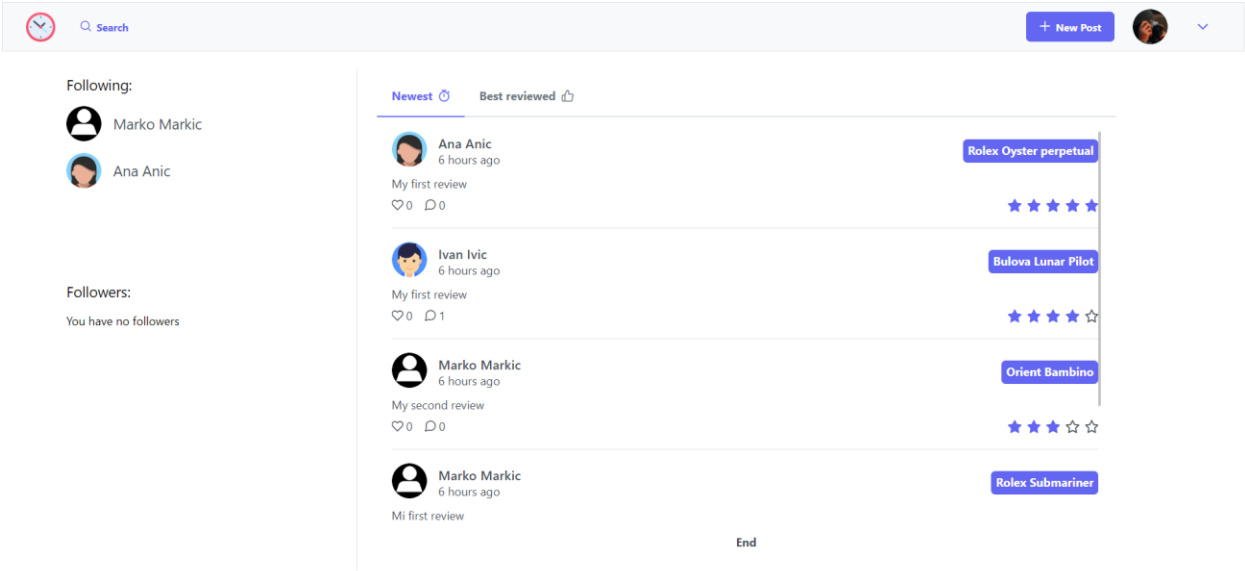
5. PREGLED ZAVRŠENE APLIKACIJE

U zadnjem poglavlju prikazane su slike završene aplikacije i sve njezine funkcionalnosti. Uz prikaz na računalu također će biti prikazan i mobilni prikaz aplikacije.

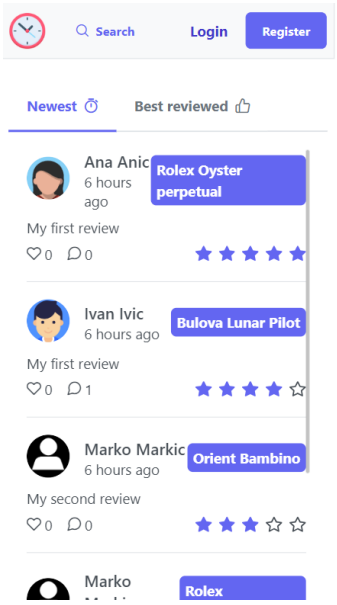
Na slikama 5.1., 5.2. i 5.3. prikazana je početna stranica aplikacije. Na prve dvije slike prikazana je aplikacija u računalnom prikazu, a na trećoj aplikacija u mobilnom prikazu.



Sl. 5.1. Početna stranica, računalni prikaz, neprijavljeni korisnik



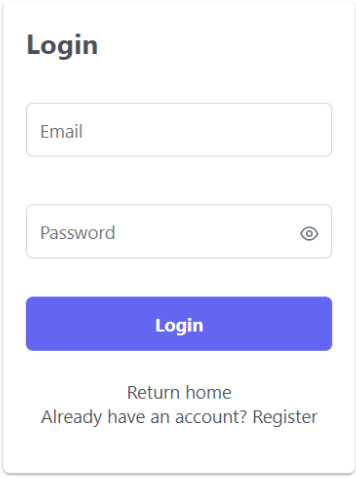
Sl. 5.2. Početna stranica, računalni prikaz, prijavljeni korisnik



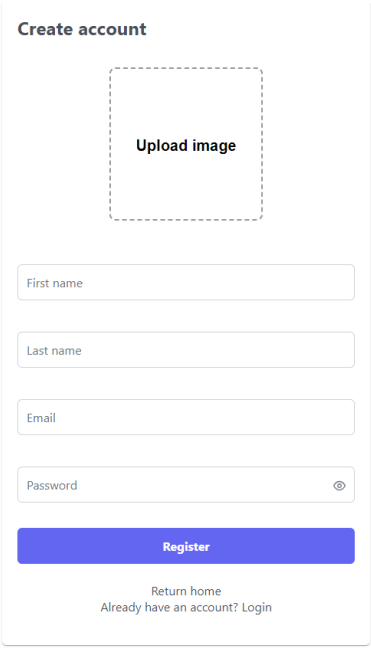
Sl. 5.3. Početna stranica, mobilan prikaz, neprijavljeni korisnik

Kada korisnik nije prijavljen, sa lijeve strane u računalnom prikazu su prikazani gumbi za prijavu i registraciju. U mobilnom prikazu gumbi su prikazani u navigacijskoj traci. Kada se korisnik prijavi, u računalnom prikazu na mjestu gumba za prijavu i registraciju prikazuje se kratki popis korisnika koje prijavljeni korisnik prati i koji prate njega. U glavnom (desnom) dijelu stranice prikazani su najnovije i najbolje ocjenjene objave u zadnjih tjedan dana. Dvije kategorije se ne prikazuju u isto vrijeme, nego se navigira preko „*Tab*“ prikaza. Objave su prikazane u obliku beskonačne liste. Informacije koje pojedina objava sadrži su slika, ime korisnika koji je napisao objavu, vrijeme objave, proizvođača i model sata postavljen u ljubičastu oznaku, prvih 200 znakova teksta recenzije, broj oznaka „sviđa mi se“, broj komentara te ocjenu autora u obliku 5 zvjezdica.

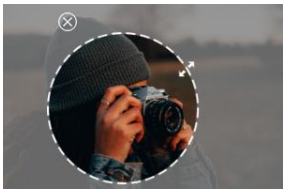
Slike 5.4. i 5.5. prikazuju obrasce za prijavu i registraciju korisnika u računalnom i mobilnom obliku. Obrasci su isti u mobilnom i u računalnom prikazu pa nema potrebe za odvojene slike. Obrazac za prijavu je jednostavan i sadrži samo dva unosa teksta, jedan za e-mail a drugi za zaporku korisnika. Unos za zaporku sadrži ikonicu za prikaz i sakrivanje zaporke. Obrazac za registraciju je složeniji. On sadrži unos za profilnu sliku korisnika, te četiri tekstualna unosa: ime, prezime, e-mail i zaporka. Pri unosu slike pojavljuje se krug koji prikazuje dio slike koji se može učitati kao profilna slika. Veličina i pozicija isječka je proizvolja što je prikazano na slici 5.6.



Sl. 5.4. Stranica za prijavu, računalni i mobilni prikaz

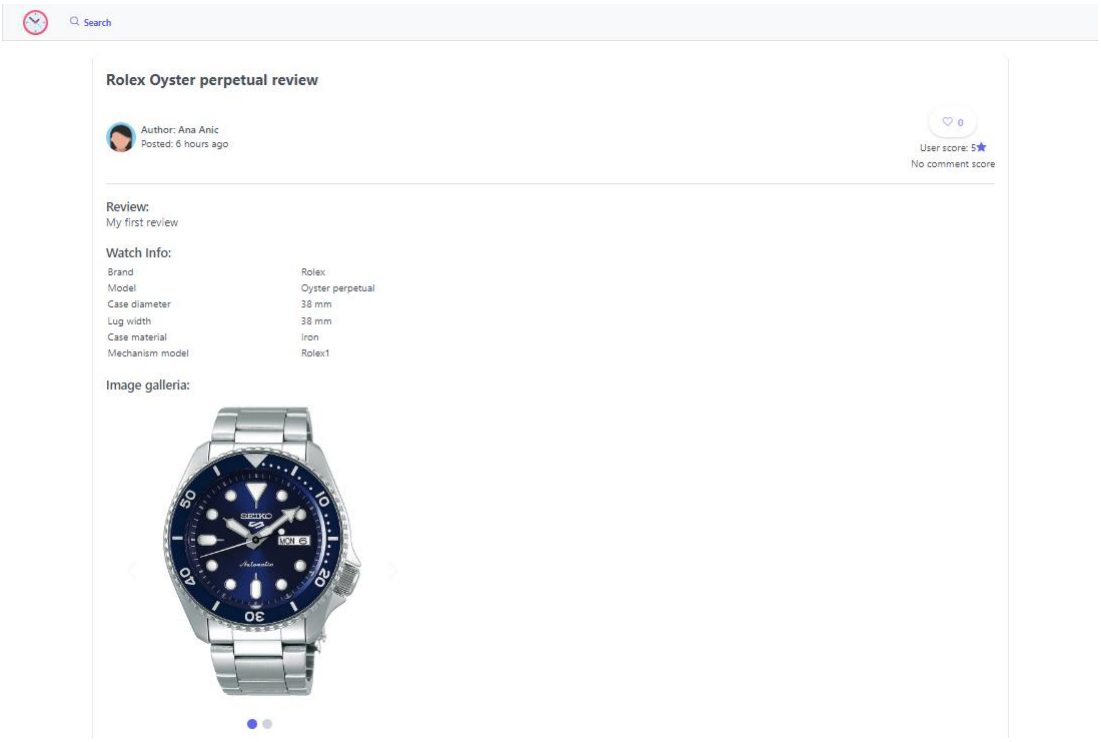


Sl. 5.5. Stranica za prijavu, računalni i mobilni prikaz



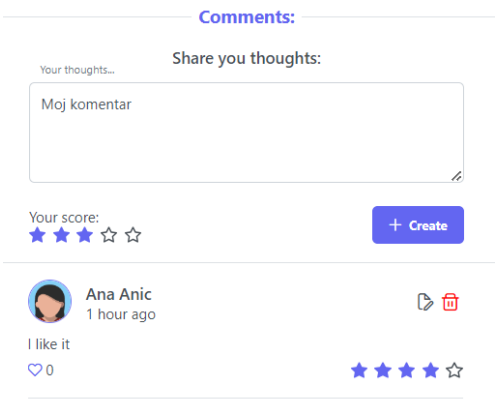
Sl. 5.6. Odabir veličine i pozicije isječka pri odabiru profilne slike

Na slikama 5.7. 5.8. prikazana je stranica pojedine objave u računalnom i mobilnom obliku. Na vrhu stranice prikazan je autor objave i kada je ona objavljena te gumb „sviđa mi se“ koji je deaktiviran ako korisnik nije prijavljen. U gumbu „sviđa mi se“ prikazan je broj korisnika koji su tu objavu označili sa „sviđa mi se“. Ispod gumba „sviđa mi se“ prikazana je ocjena autora i prosječna ocjena korisnika koji su komentirali objavu ako objava ima jedan ili više komentara.



Sl. 5.7. Stranica objave, računalni prikaz

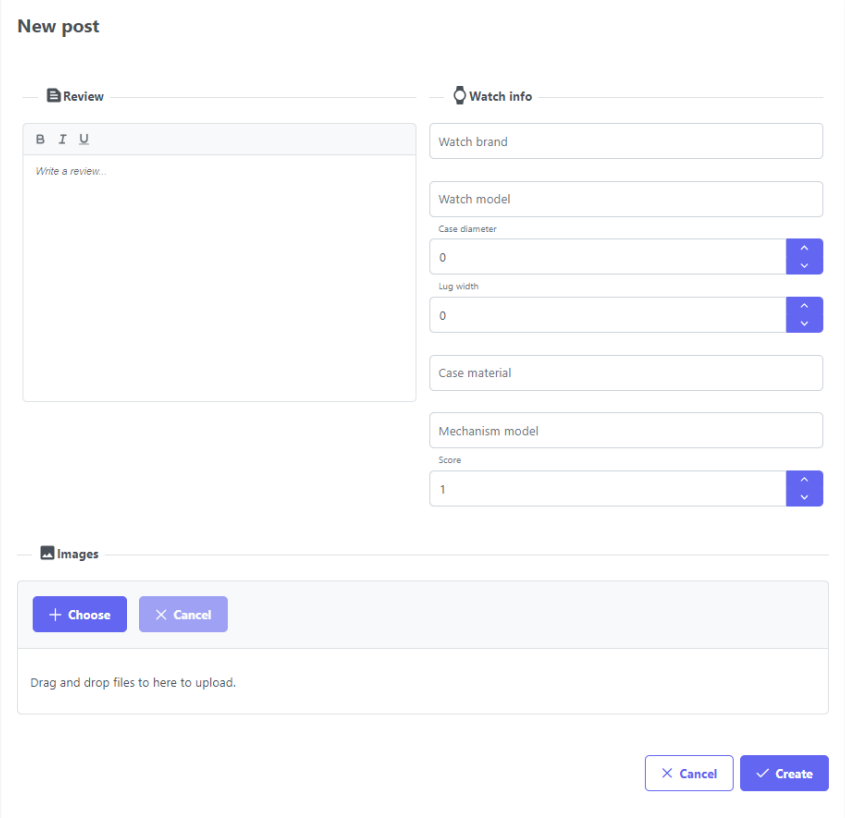
Ispod tog odjeljka nalazi se recenzija sata, ispod recenzije tablica sa detaljima o samom satu te na kraju galerija slika sata koji se recenzira. Slike imaju mogućnost listanja. Na dnu stranice postavljen je odjeljak za komentare. Prijavljeni korisnik ima mogućnost pisanja novog komentara, što je neprijavljenom korisniku onemogućeno (engl. *disabled*). Ispod obrasca za stvaranje novog komentara prikazana je lista svih komentara što se vidi na slici 5.8. .



Sl. 5.8. Odjeljak komentara na stranici objave

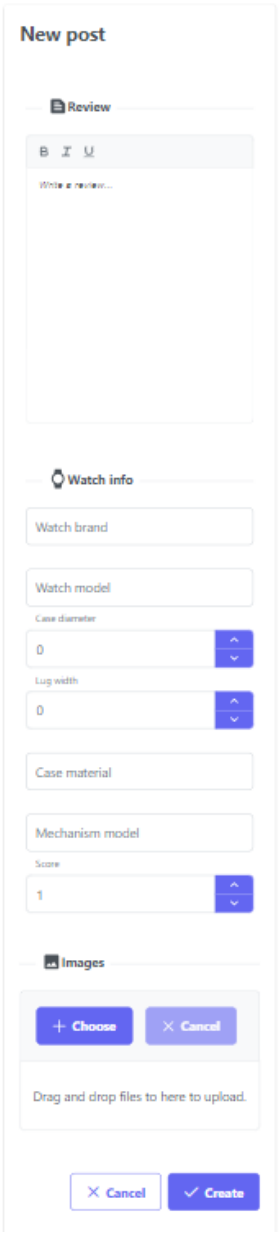
Komentar se stvara unosom teksta komentara i davanja ocjene te se klikom na gumb „Create“ lista komentara ažurira i novi komentar je dodan na vrh liste. Korisnik ima mogućnost mijenjanja svog komentara klikom na ikonicu dokumenta sa olovkom u gornjem desnom kutu. Korisnik ima i mogućnost brisanja svog komentara. Svi prijavljeni korisnici imaju mogućnost označavanja komentara sa „sviđa mi se“ kako bi korisnici mogli dobiti povratnu informaciju na svoje komentare.

Na slici 5.2. u navigacijskoj traci nalazi se gumb „New Post“, profilna slika prijavljenog korisnika i padajući izbornik koji nudi mogućnost odjave i odlaska na svoj profil. Kikom na gumb „New Post“ učitava se stranica sa obrascem za stvaranje nove objave što je prikazano na slikama 5.9. i 5.10.



Sl. 5.9. Obrazac za stvaranje novog članka, računalni prikaz

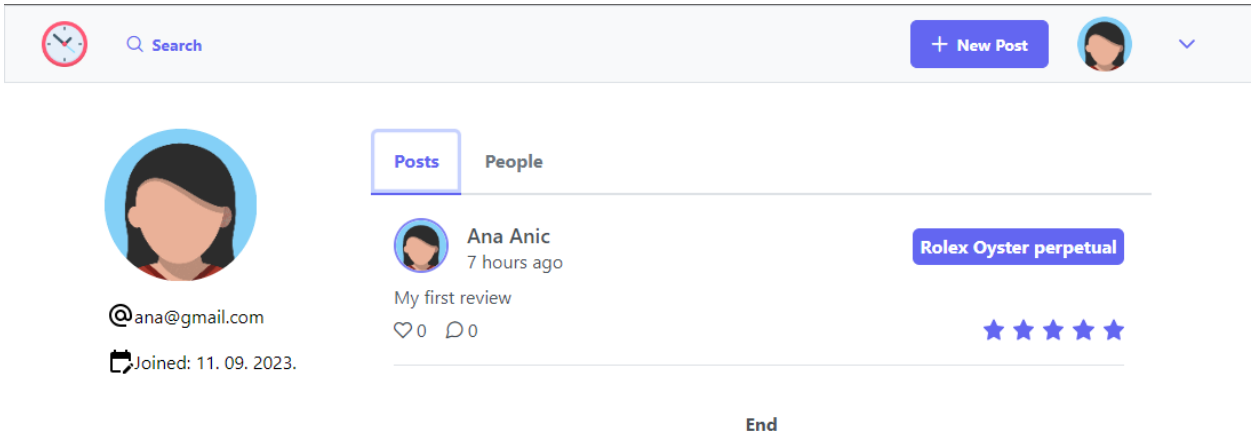
Obrazac sadrži uređivač teksta za recenziju, unose za detalje o satu te obrazac za unos slika. Pri unosu svih relevantnih informacija, klikom na gumb „Create“ prvo se pokreće validacija podataka. Ako su podatci uspješno validirani članak se stvara i korisnika se šalje na početnu stranicu gdje može vidjeti članak na vrhu popisa najnovijih članaka. Korisnik ima mogućnost odustajanja od stvaranja članka klikom na gumb „Cancel“ te se također šalje na početnu stranicu aplikacije.



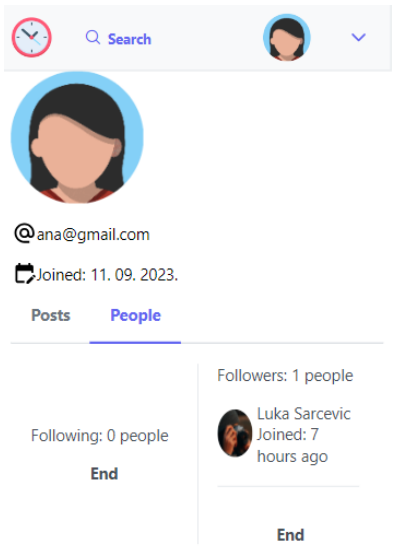
Sl. 5.10. Obrazac za stvaranje novog članka, mobilni prikaz

Stranica za uređivanje recenzije ima isti oblik kao i stranica za stvaranje objave samo su unosi i uređivač teksta popunjeni sa postojećim tekstom i brojevima. Iz tog razloga nema potrebe za prikazom te stranice. Jedina razlika je da uređivanje članka ne daje mogućnost mijenjanja slika, slike koje su odabrane pri stvaranju članka ostaju nepromjenjene.

Na sve profilne slike (engl. *avatar*) korisnika u aplikaciji se može kliknuti. Klikom na profilnu sliku učitava se stranica sa više detalja o tom korisniku koja se vidi na slikama 5.11. i 5.12. Stranica prikazuje profilnu sliku, e-mail i datum pridruživanja korisnika sa lijeve strane. Sa desne strane je „Tab“ prikaz koji prikazuje sve korisnikove objave u „Tab-u“ „Posts“ te korisnike koje korisnik prati ili koji prate njega u „Tab-u“ „People“.



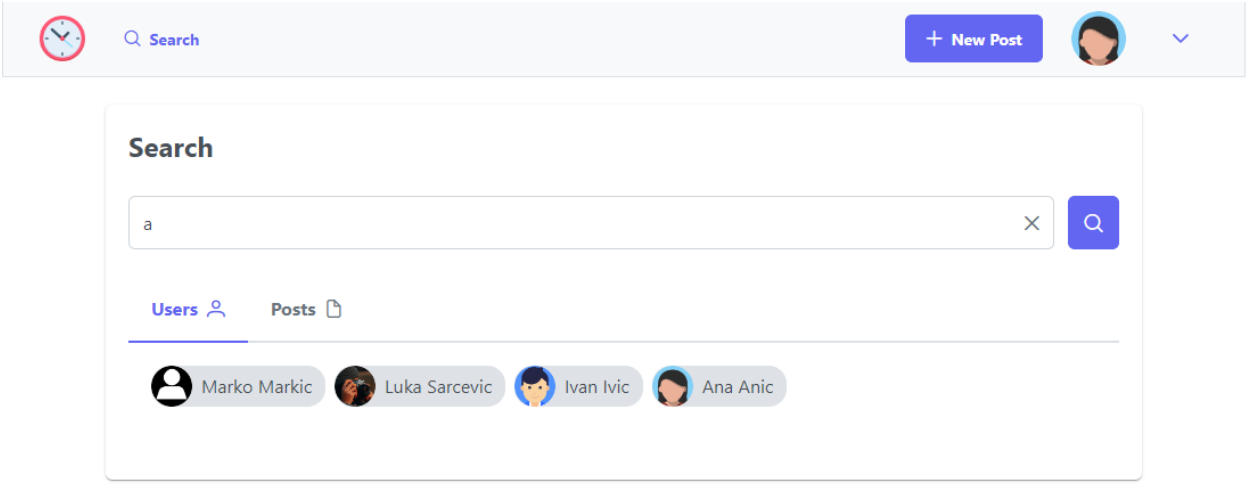
Sl. 5.11. „About“ stranica o pojedinom korisniku, računalni prikaz „Tab Posts“



Sl. 5.12. „About“ stranica o pojedinom korisniku, mobilni prikaz „Tab People“

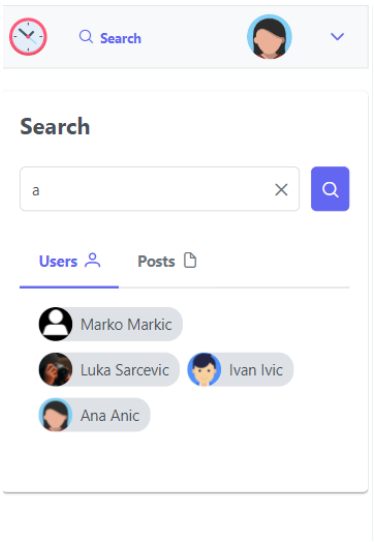
Na slici 5.. je prikazana stranica sa više detalja o korisniku u mobilnom obliku i prikazuje drugi „Tab“, „People“.

Zadnja stranica, stranica pretraživanja, prikazana je na slikama 5.13. i 5.14. Ona sadrži unos teksta u koji se mogu upisati ime korisnika ili proizvođač sata. Rezultati pretraživanja su također podijeljeni u dva „Tab-a“. Prvi je „Users“ a drugi „Posts“. „Users Tab“ prikazuje sve korisnike u obliku „Chip“ komponente koja sadrži malu ikonicu profilne slike te ime i prezime



Sl. 5.13. „Search“ stranica, računalni prikaz „Tab Users“

korisnika. „Posts Tab“ sadrži listu svih objava koje sadrže ime proizvođača sata koje se pretražuje. Lista objava je identičnog oblika kao i u ostatku aplikacije i zbog tog razloga popis objava nije prikazan na slikama. U slučaju da nije pronađen niti jedan korisnik ili objava ispisuje se poruka „No users found“ ili „No posts found“, ovisno što se pretražuje.



Sl. 5.14. „Search“ stranica, mobilni prikaz „Tab Users“

Način pretraživanja je odrađen tako da se ne mora upisati cijelo ime korisnika ili proizvođača sata, nego je dovoljno upisati dio imena.

6. ZAKLJUČAK

Cilj ovog diplomskog rada bio je dizajn i izrada društvene mreže sa osobinama portala gdje će korisnici moći dijeliti svoja mišljenja o svojim satovima, komentirati recenzije drugih korisnika i pratiti autore koji su im zanimljivi. Aplikacija na intuitivan način pruža korisniku sa računom stvaranje recenzije sata. Korisniku se na početnoj stranici prikazuju najnovije i najbolje ocjenjene recenzije, što korisniku daje uvid u novosti i dobro recenzirane satove. Korisnik može recenziju pročitati, prolistati kroz slike sata, označiti recenziju sa „sviđa mi se“, komentirati ju i u komentaru ostaviti svoja mišljenja o satu. Komentari autoru daju uvid u njegovu recenziju i mišljenja drugih ljubitelja satova. Stranica pruža pretraživanje na jednostavan način gdje se automatski pretražuju korisnici, ali i objave iz istog izraza koji se pretraživao tako da korisnik ne mora više puta pretraživati po istom izrazu. Stranica također nudi mogućnost mijenjanja postojećih komentara i objava, uz uvjet da im je autor trenutno prijavljeni korisnik.

Ova aplikacija je spoj prednosti aplikacija navedenih u poglavlju 2. Aplikacija nudi intuitivnost, jednostavnost korištenja, stvaranja recenzija te praćenje drugih korisnika. To je postignuto jer je aplikacija usredotočena na jednu stvar, a to su korisnici koji žele saznati više o pojedinom satu i podijeliti svoja mišljenja sa ostatkom svijeta.

LITERATURA

[1] Microsoft, »code.visualstudio.com/docs,« 10 09 2023. [Mrežno]. Available: https://code.visualstudio.com/docs. [Pokušaj pristupa 10 09 2023].

[2] Facebook, »Facebook; react, learn,« 10 09 2023. [Mrežno]. Available: https://react.dev/learn. [Pokušaj pristupa 10 09 2023].

[3] w3schools, »w3schools; React,« 10 09 2023. [Mrežno]. Available: https://www.w3schools.com/REACT/DEFAULT.ASP. [Pokušaj pristupa 10 09 2023].

[4] PrimeFaces, »PrimeReact,« 10 09 2023. [Mrežno]. Available: https://primereact.org/installation/. [Pokušaj pristupa 10 09 2023].

[5] MIT, »Introduction to Rust,« 10 09 2023. [Mrežno]. Available: https://web.mit.edu/rust-lang_v1.25/arch/amd64_ubuntu1404/share/doc/rust/html/book/first-edition/README.html. [Pokušaj pristupa 10 09 2023].

[6] Diesel, »DIesel ORM,« 10 09 2023. [Mrežno]. Available: https://diesel.rs/. [Pokušaj pristupa 10 09 2023].

[7] Diesel, »DIesel; getting started,« 10 09 2023. [Mrežno]. Available: https://diesel.rs/guides/getting-started.html. [Pokušaj pristupa 10 09 2023].

[8] T. A. Team, »Actix-web; documentation,« 10 09 2023. [Mrežno]. Available: https://actix.rs/docs. [Pokušaj pristupa 10 09 2023].

[9] VMware, »SPringInitializr,« 10 09 2023. [Mrežno]. Available: https://start.spring.io/. [Pokušaj pristupa 10 09 2023].

[10] D. Inc., »Docker documentation,« 10 09 2023. [Mrežno]. Available: https://docs.docker.com/get-started/overview/. [Pokušaj pristupa 10 09 2023].

[11] D. Inc., »Deocker Containers,« 10 09 2023. [Mrežno]. Available: https://docs.docker.com/get-started/. [Pokušaj pristupa 10 09 2023].

[12] T. P. G. D. Group, »Postgeres About,« 10 09 2023. [Mrežno]. Available: https://www.postgresql.org/about/. [Pokušaj pristupa 10 09 2023].

[13] Swagger, »Bearer authentication,« 10 09 2023. [Mrežno]. Available: https://swagger.io/docs/specification/authentication/bearer-authentication/. [Pokušaj pristupa 10 09 2023].

SAŽETAK

Tema diplomskog rada je izraditi društvenu mrežu za ljubitelje mehaničkih ručnih satova. Za izradu korisničkog sučelja koristili su se React sa Typescript-om uz PrimReact biblioteku gotovih komponenti, ikona i CSS stilova. Za serversko sučelje koristili su se Rust, Actix-web razvojni okvir, Diesel paket za objektno relacijsko mapiranje te PostgreSQL baza podataka. U radu su objašnjene tehnologije korištene za izradu te je postupak izrade aplikacije također detaljno objašnjen uz mnoštvo slika koje prikazuju dijelove programa. Aplikacija se sastoji od 6 responzivnih stranica: početne stranice sa najnovijim i najbolje ocjenjenim objavama, stranica objave, stranica korisnika, stranica za stvaranje i uređivanje objava i stranica za pretraživanje korisnika i objava.

Ključne riječi: mehanički ručni satovi, društvena mreža, portal, Rust, React, PostgreSQL

ABSTRACT

Social media for watch lovers

Theme of this graduate thesis is a social network for mechanical wristwatch enthusiasts. Technologies to build the front-end are React with Typescript and PrimeReact library with a set of React components, icons and CSS styles. Technologies to build the back-end are Rust, Actix-web framework, Diesel object relational mapper package and PostgreSQL database. Paper offers information on the technologies used to create the application and a detailed explanation of the process of making the application with pictures that show pieces of code. Application is made out of six responsive pages: main page that shows the newest and best rated posts, post page with post information, user account page, page for creation and editing of a post and lastly search page for searching users and posts.

Key words: mechanical wristwatches, social media, portal, Rust, React, PostgreSQL

ŽIVOTOPIS

Luka Šarčević rođen je 25. kolovoza 1999. godine u Pakracu. Završio je Osnovnu školu braće Radića u Pakracu 2014. godine. Za vrijeme osnovnog školovanja, 2013. godine završava Osnovnu glazbenu školu u Pakracu, instrument gitara. Iste godine upisuje Srednju školu Pakrac, smjer opća gimnazija koju završava 2018. godine sa vrlo dobrim uspjesima. Nakon položene mature upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku (FERIT), smjer preddiplomski studij računarstva. Upisuje diplomski smjer Programsko Inženjerstvo na FERIT-u 2021. godine. Posjeduje znanje u govoru, čitanju i pisanju engleskog i njemačkog jezika te vozačku dozvolu B kategorije. Također posjeduje znanje programskih jezika HTML, CSS, PHP, MySQL, PostgreSQL, Bootstrap, PrimeReact, C, C++, C#, JAVA, Matlab, Rust, tehnologija Visual Studio, Android Studio i upravljanje Microsoft Office alatima.

Luka Šarčević

_____

PRILOZI

- Uz printanu verziju završnoga rada nalazi se i optički disk
    - Optički disk sadrži: .docx i .pdf formate završnoga rada i mapu izvornog kôda
- Kôd se također može pronaći na Git repozitoriju na linku:
  https://github.com/SarcevicLuka/diplomski-rad-backend - serversko sučelje aplikacije
  https://github.com/SarcevicLuka/diplomski-rad-frontend - korisničko sučelje aplikacije