Razvoj web aplikacije društvene mreže

Šangulin, Ivan

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 465

RAZVOJ WEB APLIKACIJE DRUŠTVENE MREŽE

Ivan Šangulin

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 465

RAZVOJ WEB APLIKACIJE DRUŠTVENE MREŽE

Ivan Šangulin

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 465

Pristupnik:	**Ivan Šangulin (0036526952)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Mario Matijević

Zadatak: **Razvoj web aplikacije društvene mreže**

Opis zadatka:

Cilj ovog diplomskog rada jest razviti društvenu mrežu u obliku web aplikacije. Rad će se prvenstveno fokusirati na istraživanje i primjenu novih programskih alata za razvoj web aplikacija. Razvijena aplikacija sastojati će se od tri povezana dijela: baze podataka, poslužiteljskog dijela i dijela za klijente. Predložit će se efikasno programsko rješenje za osnovne elemente društvene mreže, a to su stvaranje profila, objava sadržaja, mogućnost praćenja drugih korisnika te komunikacija između korisnika. Posebnu važnost u razvoju ove društvene mreže ima primjena web socketa, omogućavajući dinamičnu komunikaciju između korisnika. Integracija web socketa unaprijedit će korisničko iskustvo pružajući brzu razmjenu informacija, poput novih objava ili poruka, što će značajno poboljšati interakciju između korisnika na platformi. U sklopu rada, za razvoj poslužiteljskog i klijentskog dijela, koristit će se JavaScript tehnologije, kako bi se olakšao razvojni proces, te se povećala dosljednost samog koda.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

Uvod

Razvoj društvenih mreža postao je jedna od najvažnijih tehnoloških inovacija u posljednjih nekoliko desetljeća, značajno mijenjajući način na koji ljudi komuniciraju, razmjenjuju informacije i održavaju društvene veze. Društvene mreže, poput Facebooka, Instagrama, Twittera i LinkedIna, ne samo da su preoblikovale osobnu komunikaciju već su također postale ključni alati za poslovanje, marketing, obrazovanje i političku participaciju [11]. S obzirom na rastući značaj ovih platformi, razumijevanje njihovog razvoja i arhitekture, postalo je ključno za sve koji se bave tehnologijom i društvom.

Cilj ovog rada je opisati proces razvoja web aplikacije društvene mreže koja omogućuje korisnicima ne samo osnovne funkcionalnosti poput objavljivanja sadržaja, sviđanja (*lajkanja*) i komentiranja, već i složenije mogućnosti kao što su istovremena komunikacija i dinamično upravljanje korisničkim interakcijama. Takve značajke čine ove aplikacije ne samo alatima za komunikaciju, već i platformama koje potiču angažman i interakciju među korisnicima.

Pri razvoju ove aplikacije poseban naglasak stavljen je na korištenje suvremenih tehnologija koje omogućuju visoku nadogradivost, sigurnost podataka i interaktivno korisničko sučelje. U ovom radu bit će detaljno prikazane korištene tehnologije, uključujući MongoDB [1] za nerelacijsku bazu podataka, Node.js [5] i Express.js [6] za poslužiteljsku stranu aplikacije, te React.js [9] i TailwindCSS [10] za klijentsku stranu. Uz to, istražit će se upotreba WebSocket protokola [7] za implementaciju istovremene komunikacije, što je ključno za poboljšanje interakcije među korisnicima.

1. Društvene mreže

Društvene mreže su online platforme koje omogućuju korisnicima da stvaraju, dijele i razmjenjuju informacije, ideje, interese i druge vrste sadržaja [11]. Ove platforme omogućuju komunikaciju među korisnicima putem tekstualnih poruka, fotografija, videozapisa i drugih multimedijalnih formata. Najpoznatije društvene mreže uključuju Facebook, Twitter, Instagram, LinkedIn i TikTok.

Glavne funkcionalnosti društveni mreža uključuju kreiranje osobnih i poslovnih profila, mogućnost povezivanja s drugim korisnicima, praćenje njihovih aktivnosti i dijeljenje sadržaja s njima, objavljivanje sadržaja poput tekstualnog i multimedijskog sadržaja, komunikacija s drugim korisnicima putem privatnih poruka ili komentiranja sadržaja.

Društvene mreže omogućuju ljudima da ostanu u kontaktu s prijateljima i obitelji, bez obzira na udaljenost. One također pomažu u obnavljanju starih prijateljstava i povezivanju s novim ljudima sa sličnim interesima. Korisnici mogu brzo i jednostavno dijeliti informacije, vijesti, fotografije i videozapise s velikim brojem ljudi. Ovo olakšava širenje svijesti o važnim događajima i povećava pristup informacijama. Mnoge društvene mreže pružaju platforme za edukativni sadržaj i učenje, uključujući vodiče za učenje (*tutorijale*), seminare na internetu (*webinare*) i edukativne videozapise. Korisnici mogu poboljšati svoje vještine i znanje putem takvih resursa. Društvene mreže omogućuju formiranje i sudjelovanje u raznim zajednicama i grupama koje podržavaju zajedničke interese, hobije ili ciljeve. To može biti izuzetno korisno za podršku i dijeljenje iskustava s ljudima koji prolaze kroz slične situacije. Za poduzeća, društvene mreže pružaju moćne alate za promociju proizvoda i usluga, izgradnju brenda te interakciju s potrošačima i klijentima. Ciljano oglašavanje omogućuje točno usmjeravanje na određenu publiku. Kroz razne igre, šale i brzo rastuće trendove, društvene mreže pružaju korisnicima priliku za opuštanje i zabavu.

Društvene mreže imaju i svoje negativne strane, poput nedostatka privatnosti, širenje dezinformacija i ovisnosti. Korištenje društvenih mreža često uključuje dijeljenje osobnih informacija, što može dovesti do problema s privatnošću i sigurnošću podataka. Brzo

širenje lažnih informacija i teorija zavjere može imati negativne posljedice na društvo. Lažne vijesti mogu izazvati paniku, dezinformirati javnost i potkopati povjerenje u provjerene izvore vijesti i informacija. Ljudi mogu stvoriti ovisnost prema društvenim mrežama, što može dovesti do gubitka vremena, smanjenja produktivnosti i zanemarivanja stvarnog života i obaveza.

2. Baza podataka

2.1. MongoDB

MongoDB je visoko nadogradiva NoSQL baza podataka koja koristi dokumentno orijentirani model za pohranu podataka [1]. Umjesto tradicionalnog relacijskog pristupa, MongoDB pohranjuje podatke u BSON (*eng. Binary JSON*) formatu, što omogućuje fleksibilnije upravljanje podacima. Ovaj model pohrane omogućuje ugrađivanje podataka i kompleksne hijerarhije unutar jednog dokumenta, čime se značajno pojednostavljuje rad s podacima i omogućuje efikasno mapiranje objekata. MongoDB je dizajniran da podrži horizontalno skaliranje, što omogućuje distribuciju podataka preko više servera, te automatski balansira opterećenje i osigurava visoku dostupnost i otpornost na kvarove. Dodatno, MongoDB podržava transakcije, omogućujući ACID (*eng. Atomicity, Consistency, Isolation, Durability*) svojstva baze podataka.

2.2. Prisma ORM

Prisma je napredni alat za upravljanje bazama podataka koji koristi ORM (*eng. Object-Relational Mapping*) pristup kako bi pojednostavio interakciju između aplikacija i baza podataka [2]. U kontekstu web aplikacija, Prisma omogućava učinkovitu integraciju između aplikacijskog sloja i baze podataka pružajući tipizirani pristup podacima, što značajno povećava sigurnost i pouzdanost aplikacije. Isto tako, omogućava definiranje odnosa između modela i automatski generira metode za rad s tim odnosima, čime se pojednostavljuje pisanje kompleksnih upita. Modeli i relacije se definiraju u datoteci *schema.prisma*, koja specificira strukturu baze podataka, uključujući tablice, njihove odnose, polja i tipove podataka. Korištenjem Prisma Clienta, mogu se pisati čisti i čitljivi upiti bez potrebe za ručnim pisanjem složenih upita na izvornom jeziku baze podataka, što ubrzava razvoj i olakšava održavanje koda. Uz to, Prisma nudi funkcionalnosti poput automatske migracije i validacije podataka, što pojednostavljuje upravljanje promjenama u bazi podataka tijekom razvoja.

Mana ovog alata je nedostatak funkcionalnosti koje imaju izvorni upiti, što može otežati pisanje samih upita. Vrlo složeni i specifični upiti moraju se pisati izvorno, što ovaj alat podržava. Problem manjka funkcionalnosti se može riješiti i dodatnom obradom podataka u aplikacijskom sloju.

2.3. Shema i izvedba baze podataka

Za rad s bazom podataka odabrana je MongoDB baza podataka i Prisma ORM alat kao poveznica između aplikacijskog sloja i podatkovnog sloja. Razlog za odabirom MongoDB je visoka nadogradivost, dinamična struktura podataka i podrška za Prisma ORM alat. Za razliku od klasičnih relacijskih baza podataka, MongoDB ima dinamičniju strukturu podataka koja se može mijenjati bez utjecaja na ostatak podataka, te se podaci mogu lakše i brže modificirati. Isto tako, u slučaju daljnjeg širenja funkcionalnosti, novu strukturu će biti lakše integrirati u postojeći sustav.

Problem tipiziranja rezultata upita nad bazom podataka riješen je uvođenjem Prisma ORM alata. U *schema.prisma* datoteci definirana je struktura baze podataka, tj. modeli tablica i međusobne relacije. Pomoću te sheme, Prisma ORM alat vraća tipizirane podatke samo onih podataka koje smo striktno naveli u upitu, te izbacuje potrebu za mapiranje rezultata u klase i dohvat nepotrebnih podataka, čime smanjuje mrežni promet. Isto tako, Prisma ORM podržava i druge baze podataka poput relacijskih baza podataka, te u slučaju migracije shema ostaje slična, tj. potrebno je samo promijeniti tipove podataka u bazi podataka kako bi migracija bila uspješna.

U isječku koda 2.1 prikazana je shema strukture baze podataka u *schema.prisma* datoteci. Model reprezentira kolekciju u bazi podataka, odnosno tablicu ako je riječ o drugim bazama podataka, a atributi unutar modela reprezentiraju podatke unutar same kolekcije, njihove tipove podataka i dodatne anotacije ovisno o tipu podataka. Isto tako, unutar modela nalaze se i definicije relacija prema drugim modelima unutar *schema.prisma* datoteke. Kod s oznakom „@@map“ označava mapiranje modela na odgovarajuću kolekciju u bazi podataka.

U modelu, podatak se prvo definira imenom, potom slijedi tip podatka u aplikacijskom sloju, koji može biti opcionalan, te potom slijede dodatne anotacije ako ih podatak ima, poput tipa podatka u bazi podataka, relacije ili zadane vrijednosti.

Modeli koji su korišteni u ovom radu su model korisnika, prijateljstava, objava, sviđanja objava, komentara, poruka, statusa korisnika, zahtjeva za prijateljstvo, obavijesti sviđanja komentara, razgovora i sudionika razgovora.

Svi modeli sadrže jedinstveni identifikator (id), koji je automatski generiran pri kreiranju objekata u bazi podataka.

Model korisnika (User) je najopsežniji model jer je aplikacija orijentirana oko tog modela, te se većina drugih modela veže na njega. Model sadrži korisničko ime (username), email, lozinku (password), identifikator profilne slike (profile_picture_uuid) i oznaku je li profil privatan (public_profile).

Model prijateljstava (Friendship) sadrži korisnikov identifikator (user_id) i identifikator prijatelja (friend_id).

Model objava (Post) sadrži tekst (text), vrijeme stvaranja (created), identifikator stvaratelja (user_id), identifikator podijeljene objave (parent_id), te identifikatore slika (photos).

Model sviđanja objava (PostLike) sadrži podatke o identifikatoru objave (post_id) i identifikatoru korisnika kojemu se objava svidjela (user_id).

Model komentara (Comment) sadrži podatke o identifikatoru stvaratelja komentara (user_id), identifikator objave koja je komentirana (post_id), tekst komentara (text), vrijeme kreiranja komentara (created), identifikator originalnog komentara u slučaju da je trenutni komentar odgovor na neki drugi (parent_id).

Model poruka (Message) sadrži podatke o vremenu kreiranja poruke (created), tekstu poruke (message), identifikatoru pošiljatelja (sender_id), identifikatoru razgovora (chat_id) i vremenu kad je poruka pročitana (read_at).

Model statusa korisnika (UserStatus) sadrži identifikator korisnika (user_id), zastavicu je li korisnik aktivan (is_online) i vrijeme zadnje aktivnosti (last_active).

Model zahtjeva za prijateljstvo (FriendRequest) sadrži identifikator pošiljatelja zahtjeva (from_user_id), identifikator primatelja zahtjeva (to_user_id), vrijeme kreiranja zahtjeva (created), zastavicu je li zahtjev pročitan (read) i zastavicu je li zahtjev prihvaćen (accepted).

Model obavijesti (Notification) sadrži identifikator primatelja obavijesti (user_id), identifikator objave ako postoji (post_id), tekst obavijesti (message), vrijeme kreiranja obavijesti (created), zastavicu je li obavijest pročitana (read) i identifikator korisnika koji je aktivirao obavijest (sender_id).

Model sviđanja komentara (CommentLike) sadrži identifikator komentara koji je označen s oznakom sviđanja (comment_id) i identifikator korisnika koji je označio komentar s oznakom sviđanja (user_id).

Model razgovora (Chat) ne sadrži nikakva dodatna polja osim glavnog identifikatora, ali sadrži dvije relacije prema modelima poruka (messages) i sudionicima razgovora (participants).

Model sudionika razgovora (ChatParticipant) sadrži identifikator korisnika (user_id) i identifikator razgovora u kojem je sudionik (chat_id).

```
datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id                    String             @id
@default(auto()) @map("_id") @db.ObjectId
  username              String             @unique
  email                 String             @unique
  password              String
  profile_picture_uuid String?
  public_profile        Boolean
  friendshipToFriend    Friendship[]
@relation("friendRelation")
  friendshipToUser      Friendship[]
@relation("userRelation")
  posts                 Post[]
  user_status           UserStatus?
```

```
  friendRequestFrom    FriendRequest[]
 @relation("fromUser")
  friendRequestTo      FriendRequest[]    @relation("toUser")
  postLikes            PostLike[]
  notifications        Notification[]
  comments             Comment[]
  commentLikes         CommentLike[]
  chatParticipants     ChatParticipant[]
  messages             Message[]

  @@map("users")
}

model Friendship {
  id        String @id @default(auto()) @map("_id")
@db.ObjectId
  user_id   String @db.ObjectId
  user      User   @relation(name: "userRelation", fields:
[user_id], references: [id])
  friend_id String @db.ObjectId
  friend    User   @relation(name: "friendRelation", fields:
[friend_id], references: [id])

  @@map("friendships")
}

model Post {
  id        String     @id @default(auto()) @map("_id")
@db.ObjectId
  text      String
  created   DateTime   @default(now())
  user_id   String     @db.ObjectId
  user      User       @relation(fields: [user_id],
references: [id])
  parent_id String?    @db.ObjectId
  photos    String[]
  parent    Post?      @relation(name: "shares", fields:
[parent_id], references: [id], onUpdate: NoAction, onDelete:
NoAction)
  shares    Post[]     @relation(name: "shares")
```

```
  likes     PostLike[]
  comments  Comment[]

  @@map("posts")
}

model PostLike {
  id      String @id @default(auto()) @map("_id")
@db.ObjectId
  post_id String @db.ObjectId
  post    Post   @relation(fields: [post_id], references:
[id])
  user_id String @db.ObjectId
  user    User   @relation(fields: [user_id], references:
[id])

  @@map("post_likes")
}

model Comment {
  id        String        @id @default(auto()) @map("_id")
@db.ObjectId
  user_id   String        @db.ObjectId
  user      User          @relation(fields: [user_id],
references: [id])
  post_id   String        @db.ObjectId
  post      Post          @relation(fields: [post_id],
references: [id])
  text      String
  created   DateTime      @default(now()) @db.Date()
  parent_id String?       @db.ObjectId
  parent    Comment?      @relation(name: "replies", fields:
[parent_id], references: [id], onUpdate: NoAction, onDelete:
NoAction)
  replies   Comment[]     @relation(name: "replies")
  likes     CommentLike[]

  @@map("comments")
}
```

```
model Message {
  id        String    @id @default(auto()) @map("_id")
@db.ObjectId
  created   DateTime  @default(now())
  message   String
  sender_id String    @db.ObjectId
  sender    User      @relation(fields: [sender_id],
references: [id])
  chat_id   String    @db.ObjectId
  chat      Chat      @relation(fields: [chat_id],
references: [id])
  read_at   DateTime?

  @@map("messages")
}

model UserStatus {
  id          String   @id @default(auto()) @map("_id")
@db.ObjectId
  user_id     String   @db.ObjectId @unique
  user        User     @relation(fields: [user_id],
references: [id])
  is_online   Boolean  @default(true)
  last_active DateTime @default(now())

  @@map("user_status")
}

model FriendRequest {
  id           String   @id @default(auto()) @map("_id")
@db.ObjectId
  from_user_id String   @db.ObjectId
  to_user_id   String   @db.ObjectId
  fromUser     User     @relation(name: "fromUser", fields:
[from_user_id], references: [id])
  toUser       User     @relation(name: "toUser", fields:
[to_user_id], references: [id])
  created      DateTime @default(now())
  read         Boolean  @default(false)
  accepted     Boolean?
```

```
  @@map("friend_requests")
}

model Notification {
  id        String   @id @default(auto()) @map("_id")
@db.ObjectId
  user_id   String   @db.ObjectId
  post_id   String?  @db.ObjectId
  message   String
  created   DateTime @default(now())
  read      Boolean  @default(false)
  sender_id String   @db.ObjectId
  sender    User     @relation(fields: [sender_id],
references: [id])

  @@map("notifications")
}

model CommentLike {
  id         String  @id @default(auto()) @map("_id")
@db.ObjectId
  comment_id String  @db.ObjectId
  comment    Comment @relation(fields: [comment_id],
references: [id])
  user_id    String  @db.ObjectId
  user       User    @relation(fields: [user_id], references:
[id])

  @@map("comment_likes")
}

model Chat {
  id           String             @id @default(auto())
@map("_id") @db.ObjectId
  participants ChatParticipant[]
  messages     Message[]

  @@map("chats")
}
```

```
model ChatParticipant {
  id      String @id @default(auto()) @map("_id")
@db.ObjectId
  user_id String @db.ObjectId
  user    User   @relation(fields: [user_id], references:
[id])
  chat_id String @db.ObjectId
  chat    Chat   @relation(fields: [chat_id], references:
[id])

  @@unique([user_id, chat_id])
  @@map("chat_participants")
}
```

Kod 2.1 - Kod strukture baze podataka u *schema.prisma* datoteci

3. Poslužitelj

3.1. Korištene tehnologije

U ovom poglavlju prikazane su tehnologije korištene tijekom razvoja poslužitelja. Detaljan pregled korištenih alata, platformi i programskih jezika pruža uvid u tehničku osnovu rješenja. Analizirane su specifične karakteristike svake tehnologije kako bi se prikazala njihova integracija u cjelokupni sustav.

3.1.1. Javascript

JavaScript je visoko dinamički programski jezik koji se široko koristi za razvoj klijentske strane web aplikacija [3]. Razvijen s ciljem dodavanja interaktivnosti i dinamičnosti web stranicama, JavaScript omogućuje programerima da manipuliraju HTML-om (*eng. HyperText Markup Language*) i CSS-om (*eng. Cascading Style Sheets*) te da reagiraju na korisničke akcije i događaje. Osim osnovnih funkcionalnosti kao što su validacija forme i animacije, JavaScript je ključan za implementaciju kompleksnih značajki poput asinkronog komuniciranja s web serverima. Njegova fleksibilnost i široka podrška u modernim preglednicima čine ga temeljnim alatom za razvoj modernih web aplikacija koje pružaju bogato korisničko iskustvo i visoku interaktivnost.

3.1.2. TypeScript

TypeScript je programski jezik razvijen kao nadogradnja nad JavaScriptom, fokusiran na dodavanje statičkog tipiziranja i naprednih značajki koje olakšavaju razvoj naprednih aplikacija [4]. Razvijen od strane Microsofta, TypeScript omogućuje programerima da eksplicitno definiraju tipove podataka za varijable, funkcije i objekte, što poboljšava sigurnost i pouzdanost koda. Kod se prevodi u čisti JavaScript tijekom izvođenja, čime se omogućava korištenje postojećih JavaScript biblioteka i programskih okvira. Ovaj jezik je posebno koristan u velikim projektima i timovima, gdje doprinosi boljoj dokumentaciji, olakšava održavanje aplikacija te smanjuje učestalost grešaka u programiranju.

3.1.3. Node.js

Node.js je otvorena platforma za izvršavanje JavaScript koda izvan web preglednika [5]. Node.js je razvijen 2009. godine i od tada je postao ključna tehnologija za razvoj server-side aplikacija, omogućujući korištenje jednog programskog jezika (JavaScript) za cjelokupni razvojni stog (*eng. stack*), što je poznato kao „JavaScript everywhere" paradigma.

Platforma koristi modularni sustav temeljen na CommonJS standardu za starije module i na ES modulima za suvremenije pristupe, koji omogućuje razdvajanje koda u manje, ponovo iskoristive module. Svaki modul može izvoziti funkcije, objekte ili vrijednosti koje mogu biti uvezene i korištene u drugim modulima. Node Package Manager (NPM) je upravitelj paketa za Node.js koji omogućuje jednostavno instaliranje, upravljanje i dijeljenje biblioteka i alata. NPM ima najveći ekosustav otvorenog koda na svijetu, s milijunima dostupnih paketa.

Node.js je često korišten za razvoj web servera i RESTful API-ja (*eng. Application Programming Interface*) zbog svoje sposobnosti upravljanja velikim brojem istovremenih veza s minimalnim troškovima. Također je idealan za aplikacije u stvarnom vremenu (*eng. real-time applications*) koje zahtijevaju brzu razmjenu podataka između klijenta i servera, poput chat aplikacija, online igara i kolaborativnih alata. Zbog svoje lagane prirode i visokih performansi, Node.js je često korišten za razvoj mikroservisnih arhitektura koje omogućuju nadogradivost i fleksibilnost u razvoju kompleksnih sustava.

3.1.4. Express.js

Express.js je minimalistički web programski okvir za Node.js koji se ističe svojom jednostavnošću i efikasnošću u razvoju nadogradivih server-side web aplikacija i API-ja [6]. Osnovna prednost Express.js-a leži u njegovoj sposobnosti jednostavnog definiranja ruta za različite HTTP (*eng. HyperText Transfer Protocol*) zahtjeve poput GET, POST, PUT i DELETE, omogućujući programerima da brzo implementiraju različite krajnje točke (*eng. endpoints*) za interakciju s klijentskim aplikacijama ili drugim servisima.

Jedna od ključnih karakteristika Express.js-a je njegova fleksibilnost u upravljanju posredničkim slojem (*eng. middleware*), što omogućuje integraciju različitih funkcionalnosti kao što su autentifikacija, autorizacija, validacija zahtjeva te obrada

pogrešaka. Posrednički sloj u Express.js-u omogućuje programerima da strukturiraju i kontroliraju tok zahtjeva i odgovora na temelju specifičnih zahtjeva aplikacije ili poslovne logike.

Isto tako, Express.js pruža jednostavno posluživanje statičkih datoteka kao što su slike, CSS i JavaScript datoteke, što je ključno za razvoj modernih web aplikacija.

3.1.5. WebSocket

WebSocket protokol predstavlja tehnologiju koja omogućava dvosmjernu, kontinuiranu komunikaciju između klijenta i poslužitelja putem jedne, dugotrajne veze [7]. U usporedbi s tradicionalnim HTTP protokolom, koji uspostavlja novi konekciju za svaki zahtjev i odgovor, WebSocket uspostavlja stalnu vezu koja omogućava brzu i efikasnu razmjenu podataka u stvarnom vremenu. Ovaj protokol koristi inicijalni HTTP zahtjev za uspostavljanje veze, nakon čega se komunikacija prebacuje na WebSocket protokol, omogućavajući dvosmjernu razmjenu poruka bez potrebe za ponovnim uspostavljanjem veze. WebSocket protokol je posebno koristan za aplikacije koje zahtijevaju često ažuriranje podataka ili interakciju u stvarnom vremenu. S obzirom na svoju nisku latenciju i minimalne zahtjeve za propusnost, WebSocket protokol omogućava iznimno brzu i učinkovitu komunikaciju.

3.1.5.1 Socket.IO

Socket.IO biblioteka nadograđuje WebSocket protokol, pružajući dodatne značajke kao što su automatsko ponovno povezivanje, detekcija isključenja klijenata, i rezervni mehanizam (*eng. fallback mechanism*) za slučajeve kada WebSocket nije dostupan [8]. Dok WebSocket omogućava osnovnu dvosmjernu komunikaciju, Socket.IO pojednostavljuje implementaciju složenih scenarija poput emitiranja poruka na više klijenata, upravljanja sobama i razmjene binarnih podataka. Ova biblioteka također integrira podršku za događaje, omogućavajući rukovanje različitim vrstama komunikacije u stvarnom vremenu.

3.2. Struktura poslužitelja

3.2.1. REST API

REST API (*eng. Representational State Transfer Application Programming Interface*) je arhitektonski stil koji omogućuje komunikaciju između klijenta i poslužitelja putem HTTP protokola. Temelji se na konceptu resursa, gdje svaki resurs ima jedinstveni URI (*eng. Uniform Resource Identifier*) preko kojeg je dostupan. Ključna karakteristika REST API-ja je korištenje standardnih HTTP metoda poput GET, POST, PUT i DELETE za manipulaciju resursima. Metoda GET služi za dohvat, POST za kreiranje, PUT za ažuriranje i DELETE za brisanje podataka. Ovaj pristup omogućuje jednostavan način interakcije s web servisima, pružajući jasnu i dosljednu strukturu za pristup podacima i operacijama. REST API je dizajniran tako da ne posjeduje stanje, što znači da svaki zahtjev od klijenta prema poslužitelju mora sadržavati sve potrebne informacije za obradu tog zahtjeva, čime se olakšava nadogradivost i održavanje sustava. Osim toga, REST API omogućuje predmemoriranje odgovora kako bi se poboljšale performanse i smanjilo opterećenje poslužitelja.

3.2.2. Rute i kontroleri

Rute definiraju način na koji HTTP zahtjevi upućeni aplikaciji trebaju biti obrađeni, povezujući URL (*eng. Uniform Resource Locator*) putanje s odgovarajućim funkcijama kontrolera. Kada su rute spojene s kontrolerom, kao što je slučaj u ovom radu, kontroler direktno upravlja obradom tih zahtjeva, obavljajući operacije poput dohvaćanja podataka, validacije ili generiranja odgovora. U ovom pristupu, svaka ruta u aplikaciji upućuje zahtjev na funkciju unutar kontrolera koja obavlja potrebne zadatke i šalje odgovor klijentu.

Kontroleri su ključne komponente koje upravljaju obradom HTTP zahtjeva i generiranjem odgovora. Kontroleri djeluju kao posrednici između korisničkih zahtjeva i poslovne logike aplikacije, omogućujući strukturiran i modularan pristup u razvoju web aplikacija. Svaki kontroler obično sadrži metode koje odgovaraju specifičnim HTTP rutama, kao što su GET, POST, PUT ili DELETE. Ove metode preuzimaju dolazne zahtjeve, obrađuju podatke i generiraju odgovore koji se šalju natrag klijentima. Kontroleri

pomažu u organiziranju koda, tako da se logika obrade zahtjeva odvaja od poslovne logike, čime se poboljšava čitljivost i održivost aplikacije. Također, omogućuju lakše testiranje i proširivanje aplikacije, jer se funkcionalnosti mogu razvijati i mijenjati neovisno o drugim dijelovima sustava.

Isječak koda 3.1 prikazuje rutu i funkciju za obradu zahtjeva, koji služe kako bi korisnik učitao svoju sliku profila. Validiraju se ekstenzija, veličina i količina učitanih datoteka, tj. u ovom slučaju potrebno je učitati samo jednu datoteku. Ako je datoteka uspješno validirana, preimenuje se u jedinstveni identifikator, te se sama datoteka sprema na poslužitelj, a njen identifikator se sprema u bazu podataka.

```
userRouter.post(
  "/upload-profile-picture",
  RequiresAuth,
  async (req: Request, res: Response) => {
    try {
      const userId = req.userId as string;
      const maxFileSize = 8 * 1024 * 1024;
      const allowedExtensions = [".jpg", ".jpeg", ".png"];
      const allowedMimeTypes = ["image/jpeg", "image/png",
"image/jpg"];
      const form = formidable({
        uploadDir: path.join(__dirname,
"../../public/image/profile_picture"),
        keepExtensions: true,
        maxFileSize: maxFileSize,
        maxFiles: 1,
      });
      const [_, files] = await form.parse(req);
      const file = files.photo instanceof Array ?
files.photo[0] : files.photo;
      if (!file) {
        return res.status(400).send("No file uploaded");
      }
      const extension = path.extname(file.originalFilename ??
"").toLowerCase();
      const mimeType = file.mimetype;
      const oldPath = file.filepath;
```

```
      if (
        !allowedExtensions.includes(extension) ||
        !mimeType ||
        !allowedMimeTypes.includes(mimeType)
      ) {
        fs.unlinkSync(oldPath);
        return res.status(400).json({
          error: "Invalid file type. Only .jpg, .jpeg, and
.png are allowed.",
        });
      }
      const newFileName = uuidv4() + extension;
      const newPath = path.join(__dirname,
"../../public/image/profile_picture", newFileName);
      fs.rename(oldPath, newPath, async (err) => {
        if (err) {
          return res.status(500).json({ error: "Failed to
save file" });
        }
        await changeProfilePicture(userId, newFileName);
        return res.sendStatus(200);
      });
    } catch (err) {
      console.log(err);
      return res.status(500).send("Couldn't upload profile
picture");
    }
  }
);
```

Kod 3.1 – Ruta i funkcija za obradu zahtjeva iz kontroler datoteke

3.2.3. Posrednički sloj

Posrednički sloj (*eng. middleware*) predstavlja ključnu komponentu za upravljanje obradom HTTP zahtjeva i odgovora između klijenta i servera. Posrednički sloj se koristi za implementaciju raznih funkcionalnosti koje se izvršavaju prije nego što zahtjev dosegne kontroler, ili nakon što odgovor napusti aplikaciju. Posrednički sloj omogućava izvršenje operacija kao što su autentifikacija i autorizacija korisnika, validacija i čišćenje ulaznih

podataka, upravljanje sesijama, te primjena specifičnih poslovnih pravila ili sigurnosnih politika. Ovaj sloj djeluje kao posrednik između različitih komponenti aplikacije, omogućujući modularan i fleksibilan pristup obradi zahtjeva. Time se ostvaruje bolja organizacija i održavanje koda, jer funkcionalnosti koje se često koriste mogu biti centralizirane i ponovno korištene u različitim dijelovima aplikacije. Posrednički sloj također omogućuje lako proširivanje aplikacije, jer nove funkcionalnosti mogu biti dodane u obliku dodatnih funkcija posredničkog sloja, bez potrebe za značajnim promjenama u postojećoj strukturi koda. Ova arhitektura poboljšava nadogradivost i sigurnost aplikacije**.**

Isječak koda 3.2 prikazuje posrednički sloj za autentikaciju korisnika. Ovaj posrednički sloj koristi se u većini aplikacije kao sigurnosna provjera prije izvršavanja raznih kontroler funkcija, čime je izrazito bitan dio aplikacije. U autentikacijskom posredničkom sloju prvo se provjerava prisutnost JWT-a (*eng. JSON Web Token*). JWT je standard za stvaranje tokena koji se koristi za sigurno prenošenje informacija između dvije strane. Nakon provjere prisutnosti, JWT se dekriptira i verificira korištenjem *jwt.verify* funkcije, te se iz njega izvlači identifikator korisnika. U slučaju isteka ili greške pri verifikaciji tokena, posrednički sloj odgovara s 401 Neovlašteni pristup. Nakon verificiranja i dekriptiranja, provjerava se postojanje identifikatora u bazi podataka, te se identifikator sprema u objekt zahtjeva *req.userId*, koji se potom može koristiti u kontroler funkciji kod obrade zahtjeva.

```
export const RequiresAuth = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const cookie: string | undefined =
req.signedCookies.session;
    if (!cookie) {
      return res.sendStatus(401);
    }
    const secret: string | undefined =
env.SECRET_ACCESS_TOKEN;
    if (!secret) {
      throw new Error();
```

```
    }
    jwt.verify(cookie, secret, async (err, decoded) => {
      if (err) {
        return res.status(401).send("Session expired");
      }
      const token = decoded as JwtPayload;
      const { id } = token;
      req.userId = (await findMyself(id)).id;
    });
    next();
  } catch (err) {
    return res.status(500).send("Error occured verifying
session");
  }
};
```

Kod 3.2 – Posrednički sloj za autentikaciju korisnika

3.2.4. Servisi

Servisi predstavljaju sloj koji upravlja poslovnom logikom i interakcijom s podacima. Oni djeluju kao posrednici između kontrolera i sloja za pristup podacima, omogućujući centraliziranu obradnu logiku i obavljanje specifičnih funkcionalnosti aplikacije. Servisi obično sadrže metode za izvršavanje operacija kao što su obrada podataka, primjena poslovnih pravila i upravljanje transakcijama. Ovaj pristup omogućuje jasnu podjelu odgovornosti, čime se poboljšava čitljivost i održavanje koda. Kontroleri pozivaju odgovarajuće servise kako bi obradili zahtjeve, dok servisi upravljaju složenijim poslovnim operacijama i komunikacijom s bazom podataka. Razdvajanjem poslovne logike od kontrolera, servisi omogućuju bolju modularnost i ponovnu upotrebu koda, kao i lakše testiranje i proširivanje aplikacije.

Isječak koda 3.3 prikazuje funkciju za kreiranje i slanje nove obavijesti. U prvom dijelu isječka koda u slučaju obavijesti sviđanja ili komentiranja objave ili komentara, provjerava se je li postoji ista obavijest za istu objavu ili komentar koja je nastala u zadnjih pet minuta. Ako postoji ista obavijest, ne dolazi do kreiranja i slanja. Razlog te provjere je što ne želimo korisnike preplaviti obavijestima. Zatim dolazi do kreiranja obavijesti u bazi podataka i slanja same obavijesti. Slanje se vrši putem WebSocket protokola, čime obavijest korisniku dolazi u stvarnom vremenu.

```
export const createNotification = async (
  userId: string,
  postId: string | undefined,
  message: string,
  senderId: string
) => {
  if (postId && (message.includes("like") ||
message.includes("comment"))) {
    const lastFiveMinutes = subMinutes(new Date(), 5);
    const prevNotification = await
prisma.notification.findFirst({
      where: {
        user_id: userId,
        post_id: postId,
        message: message,
        sender_id: senderId,
        created: {
          gte: lastFiveMinutes,
        },
      },
    });
    if (prevNotification) {
      return;
    }
  }
  const notification = await prisma.notification.create({
    data: {
      user_id: userId,
      post_id: postId,
      message: message,
      sender_id: senderId,
    },
    select: {
      id: true,
      createdDescriptive: true,
      message: true,
      post_id: true,
      read: true,
      sender: {
```

```
          select: {
            username: true,
            profile_picture_uuid: true,
          },
        },
      },
    });
    io.to(userId).emit("notification", { notification });
  };
```

Kod 3.3 – Funkcija u servis datoteci za kreiranje i slanje nove obavijesti

3.3. Komunikacija u stvarnom vremenu

Komunikacija u stvarnom vremenu odnosi se na mogućnost trenutnog prijenosa i primanja informacija, poruka ili interakcija između korisnika bez zamjetnog kašnjenja. Ova tehnologija omogućuje korisnicima da komuniciraju i dijele sadržaj u stvarnom vremenu, čime se značajno povećava njihova angažiranost i aktivnost na društvenim mrežama. Primjeri komunikacije u stvarnom vremenu uključuju chat i poruke, gdje korisnici mogu odmah slati i primati tekstualne poruke, slike ili videozapise, kao i video prijenose u stvarnom vremenu (*eng. live video streaming*) koji omogućuju emitiranje sadržaja uživo uz neposredne reakcije publike. Isto tako, obavijesti u stvarnom vremenu korisnicima omogućuju trenutnu reakciju na označavanja, poruke ili druge interakcije, što doprinosi dinamičnosti i održavanju interesa unutar zajednica na društvenim mrežama. Komunikacija u stvarnom vremenu igra ključnu ulogu u poboljšanju korisničkog iskustva, omogućujući bržu i prirodniju povezanost među korisnicima.

3.3.1. Implementacija u ovom radu

U ovom radu implementirana je komunikacija u stvarnom vremenu u vidu slanja tekstualnih poruka, primanja obavijesti i zahtjeva za prijateljstvo. Korisnici mogu međusobno slati i primati tekstualne poruke u stvarnom vremenu, čime se povećava angažiranost korisnika u samoj aplikaciji. Isto tako, mogu primati obavijesti o reakcijama na njihove objave i komentare i zahtjevima za prijateljstvo. Funkcionalnosti su implementirane putem WebSocket protokola i Socket.IO biblioteke.

WebSocket protokol funkcionira putem događaja (*eng. event*) i pratitelja (*eng. listener*). Na poslužitelju i klijentu se postave pratitelji događaja, te se emitirani događaji mapiraju na pratitelje. Pratitelji i događaji se mapiraju preko zajedničkog imena, npr. ako smo postavili pratitelj imena *primjer*, kako bi se događaj povezao s tim pratiteljem, moramo ga emitirati pod istim imenom, u ovom slučaju *primjer*. Pratitelj sadrži obraditelja događaja (*eng. event handler*), koji predstavlja funkciju s kodom koji želimo izvršiti nakon određenog događaja. Isto tako, WebSocket protokol podržava kreiranje soba (*eng. rooms*). Sobe u kontekstu WebSocket protokola predstavljaju logičke skupine unutar WebSocket komunikacijskog kanala, kojima se korisnici mogu pridružiti, te u njima komunicirati. Događaji se mogu emitirati globalno ili po sobama. U ovom radu svaki korisnik ima svoju sobu u koju mu se emitiraju događaji, čime je osigurana privatnost informacija.

U prvom dijelu isječka koda 3.4 prikazan je kod koji se izvodi nakon uspješnog uspostavljanja WebSocket konekcije. Korisnik se pridružuje u sobu sa svojim identifikatorom, te mu se u bazi podataka ažurira status aktivnosti, tj. korisnik postaje aktivan. Drugi dio isječka predstavlja obraditelj događaja i pratitelj za događaj poruke. Pomoću *socket.on* metode inicijaliziramo pratitelj za određeni događaj, te mu prosljeđujemo obraditelj događaja. U obraditelju događaja, pristigla poruka sprema se u bazu podataka, te nakon uspješne pohrane, emitira se događaj u sobu korisnika kome je inicijalno poruka poslana. U zadnjem dijelu isječka prikazan je obraditelj događaja i pratitelj za događaj zatvaranja WebSocket konekcije. Korisniku se ažurira status aktivnosti u bazi podataka, tj. korisnik postaje neaktivan. Uklanjaju se svi pratitelji koji su povezani s tom konekcijom, te korisnik izlazi iz sobe.

```
io.on("connection", async (socket: ISocket) => {
    const userId = socket.userId as string;
    await updateStatus(userId, true);
    console.log(`New client connected - ${userId}`);
    socket.join(userId);

    const messageListener = async (
      {
        chatId,
        message,
        created,
```

```
      }: {
        chatId: string;
        message: string;
        created: string;
      },
      ack: (success: boolean) => void
    ) => {
      try {
        const createdDate = new Date(created);
        const [{ myData, unreadCount }, participants] = await
Promise.all([
          createMessage(userId, chatId, message,
createdDate),
          getChatParticipants(userId, chatId),
        ]);
        participants.forEach((p) => {
          io.to(p.user_id).emit(
            "message",
            chatId,
            {
              sender_id: userId,
              message,
              created,
            },
            myData,
            unreadCount
          );
        });
        if (ack) ack(true);
      } catch (err) {
        console.log(err);
        if (ack) ack(false);
      }
    };
    socket.on("message", messageListener);

    ...

    socket.on("disconnect", async () => {
      await updateStatus(userId, false);
```

```
    console.log(`Client disconnected - ${userId}`);
    socket.removeAllListeners();
    socket.leave(userId);
  });
});
```

Kod 3.4 Početak, pratitelj i zatvaranje WebSocket konekcije

4. Klijent

4.1. Korištene tehnologije

Za razvoj klijenta korištene su tehnologije JavaScript, TypeScript, WebSocket i Socket.IO koje su opisane u prijašnjem poglavlju, te React i TailwindCSS koje će biti opisane u nastavku.

4.1.1. React

React je JavaScript biblioteka otvorenog koda. Namijenjena je za izgradnju korisničkih sučelja za jednostrane (*engl. single-page*) aplikacije [9]. Glavna prednost Reacta je u njegovoj komponentnoj arhitekturi koja omogućuje razdvajanje korisničkog sučelja na manje, ponovno upotrebljive i izolirane komponente. U Reactu, temeljna jedinica sučelja je komponenta. Komponente su modularni, samodostatni dijelovi korisničkog sučelja koji se mogu kombinirati kako bi se izgradile složenije strukture i aplikacije. Svaka komponenta može posjedovati dva ključna koncepta, a to su stanje (*eng. state*) i svojstva (*eng. props*). Stanje predstavlja unutarnje, lokalne atribute komponente koji se mogu dinamički mijenjati tijekom njenog životnog ciklusa. Promjene u stanju komponente izravno utječu na njezin prikaz, čime se omogućuje dinamičko ažuriranje korisničkog sučelja kao odgovor na korisničke akcije ili druge događaje. Svojstva su vanjski podaci koje komponenta prima od svojih roditeljskih komponenti. Svojstva su nepromjenjiva unutar komponente, što znači da ih komponenta koristi kao ulazne parametre bez mogućnosti njihovog mijenjanja. Ovaj pristup omogućuje predvidljivo ponašanje komponenti i olakšava njihovu ponovnu upotrebu i testiranje. Jedna od ključnih inovacija Reacta je koncept virtualnog DOM-a (*eng. Document Object Model*). Virtualni DOM je reprezentacija stvarnog DOM-a u memoriji. Kada se stanje komponente promijeni, React ažurira virtualni DOM i uspoređuje ga s prethodnim stanjem. Na temelju te usporedbe, React generira minimalan broj promjena koje su potrebne za ažuriranje stvarnog DOM-a. Ovaj pristup značajno poboljšava performanse aplikacije jer minimizira broj skupih operacija na stvarnom DOM-u. React koristi JSX (*eng. JavaScript XML*),

sintaktičku ekstenziju za JavaScript koja omogućuje pisanje HTML-a unutar JavaScript koda. JSX poboljšava čitljivost i održavanje koda.

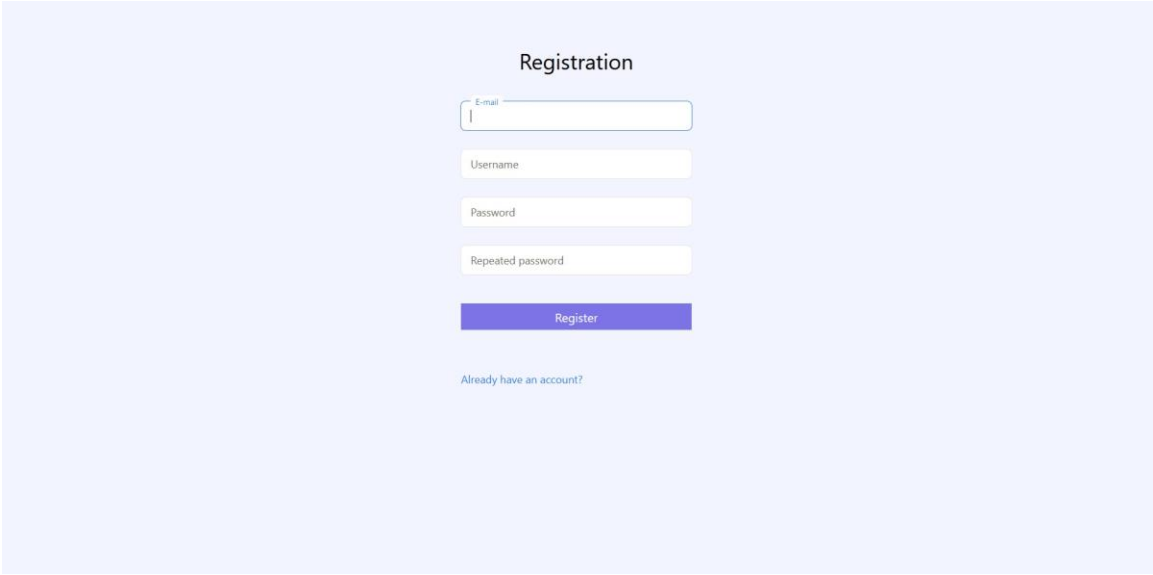
4.1.2. Tailwind CSS

Tailwind CSS (*eng. Cascading Style Sheets*) je programski okvir za izradu stilova u CSS-u [10]. Umjesto da pruža gotove komponente i stilove, TailwindCSS nudi niz malih, ponovno upotrebljivih CSS klasa (*eng. utility classes*) koje se mogu kombinirati za postizanje željenog izgleda i funkcionalnosti. Ovaj pristup omogućava brzu izgradnju prilagođenih dizajna bez potrebe za pisanjem standardnog CSS-a. Osim što omogućuje bržu i lakšu prilagodbu, pomaže u održavanju konzistentnosti dizajna i smanjenju broja stilskih pravila. Isto tako, dolazi s alatima za optimizaciju i prilagodbu, uključujući mogućnost definiranja vlastitih stilova, čime se dodatno poboljšava fleksibilnost i prilagodba dizajna.

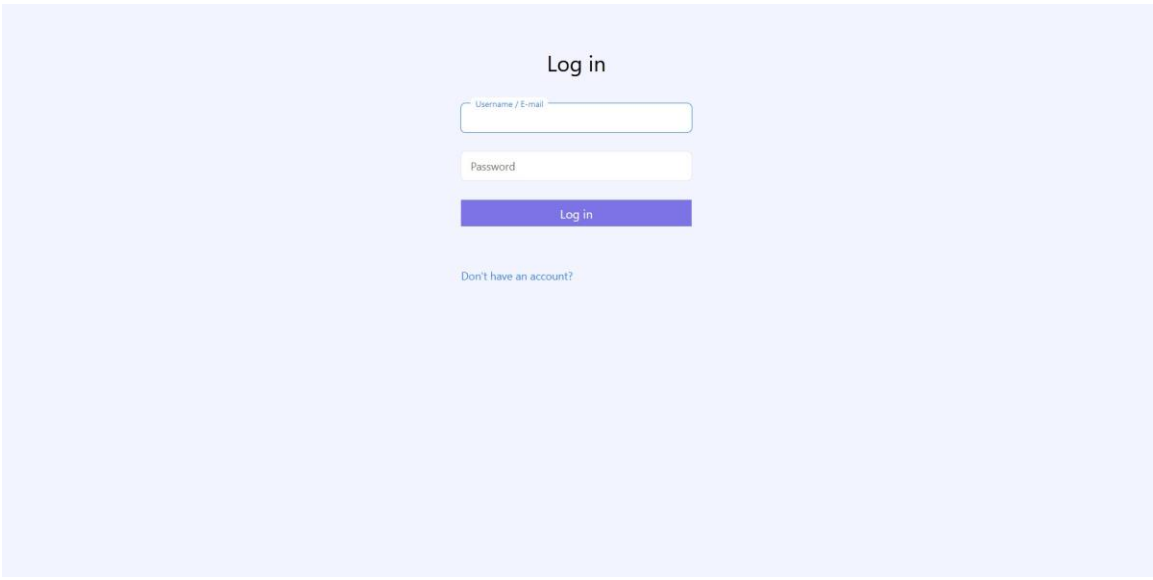
4.2. Korisničke upute

4.2.1. Prijava i registracija

Prije korištenja same aplikacije, potrebno je napraviti korisnički račun, te prijaviti se u aplikaciju. Za izradu korisničkog računa (Slika 4.1), potrebno je upisati e-mail, korisničko ime, lozinku i ponovljenu lozinku. U slučaju da korisnik već ima korisnički račun, klikom na poveznicu „Already have an account?“ prikazat će mu se forma za prijavu u aplikaciju (Slika 4.2). Kako bi se korisnik uspješno prijavio, treba upisati svoj e-mail ili korisničko ime, te pripadajuću lozinku.



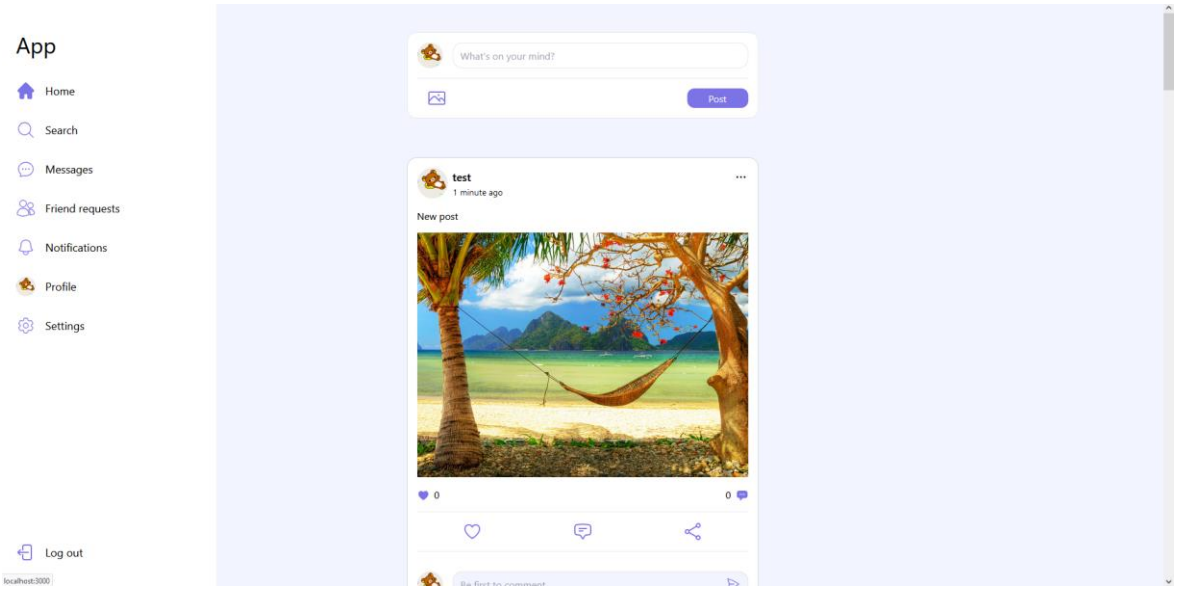
Slika 4.1 Registracija



Slika 4.2 Prijava

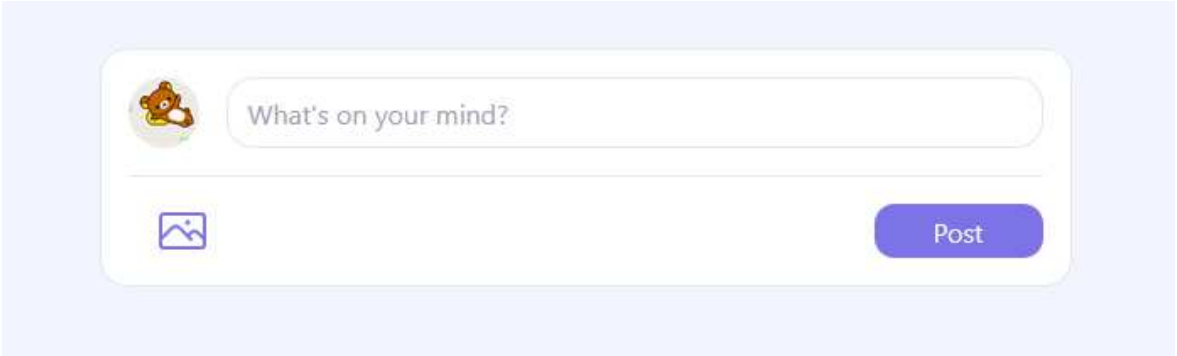
4.2.2. Početna stranica

Nakon uspješne prijave, korisnik će biti preusmjeren na početnu stranicu. Na početnoj stranici (Slika 4.3) s lijeve strane nalazi se navigacijska traka, a u sredini se nalaze najnovije objave korisnika i njegovih prijatelja, te na samom vrhu u sredini, nalazi se forma za kreiranje nove objave.



Slika 4.3 Početna stranica i navigacijska traka

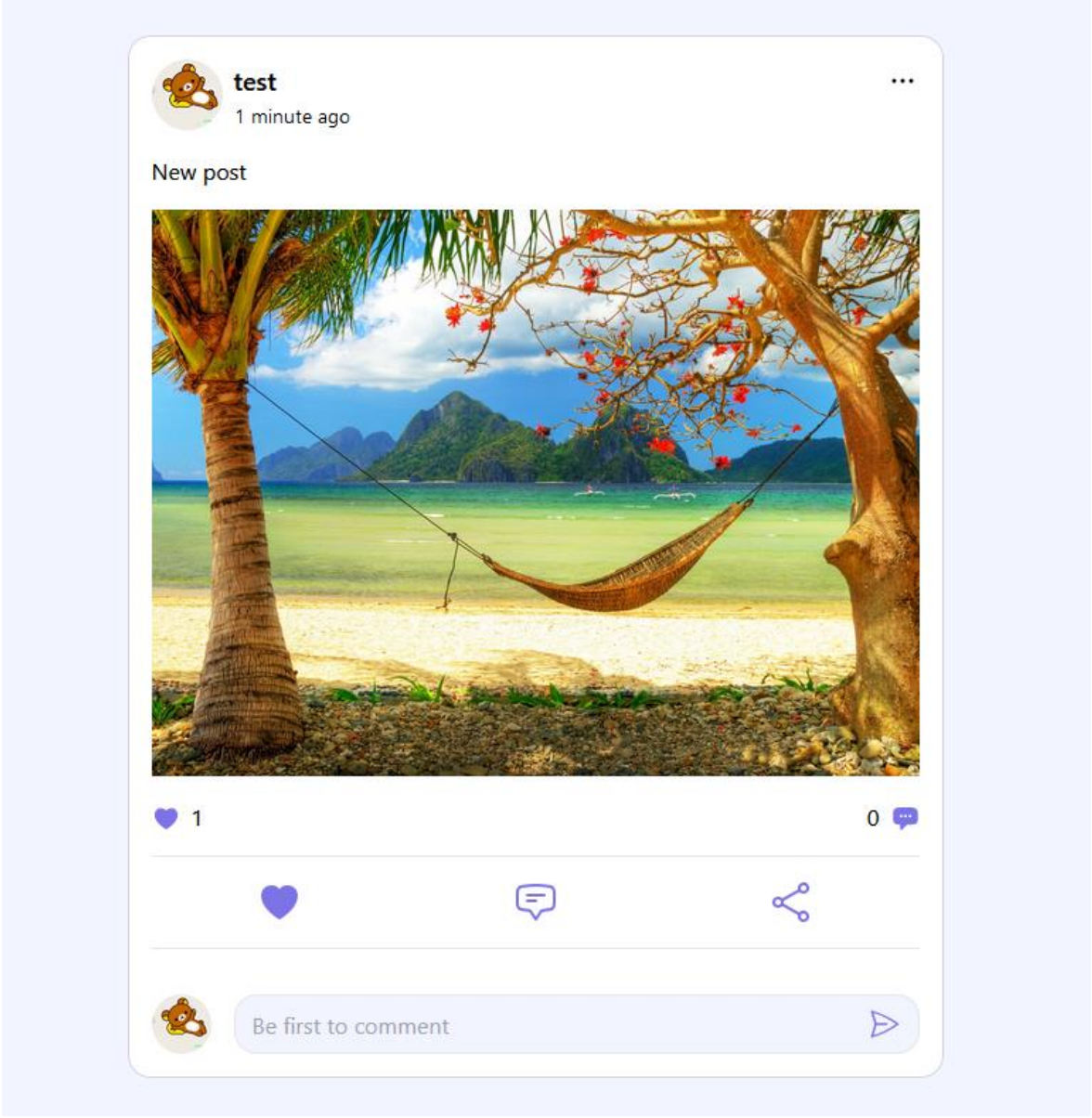
Forma za kreiranje nove objave (Slika 4.4) na vrhu sadrži profilnu sliku korisnika i polje za upis opisa objave, te na dnu sadrži gumb za učitavanje slika objave i gumb za objavu.



Slika 4.4 Forma za kreiranje nove objave

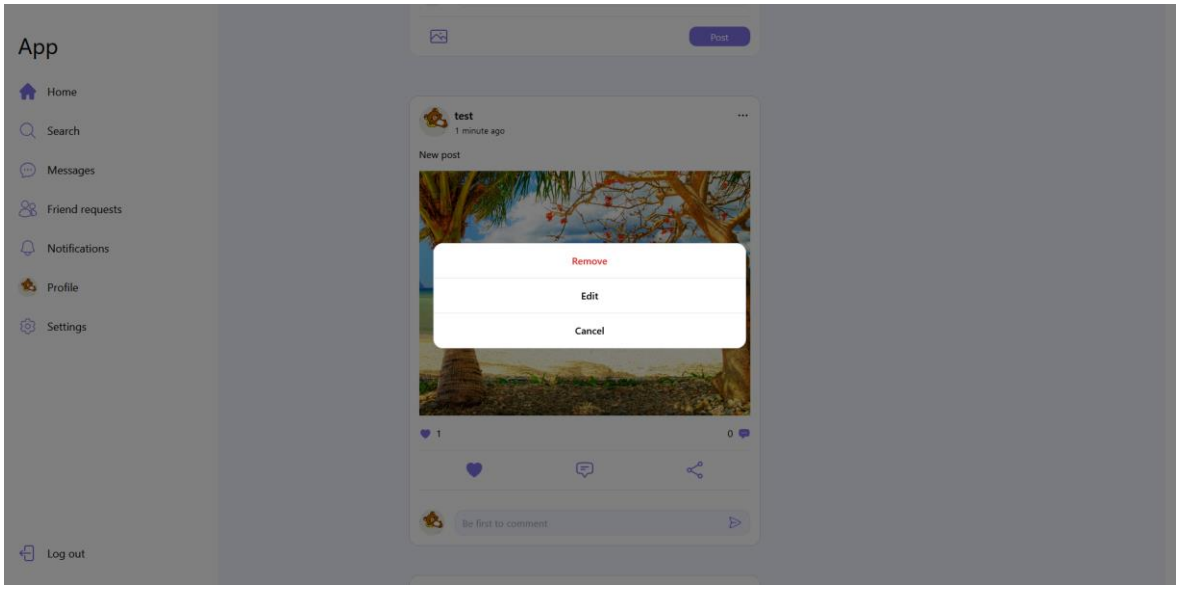
Na slici 4.5 nalazi se prikaz objave. Na vrhu objave s lijeve strane nalazi se profilna slika korisnika, njegovo korisničko ime i vrijeme kad je objavljena. S desne strane na vrhu objave, nalazi se gumb u obliku tri točke koji otvara izbornik za upravljanje objavom. Ispod vremena objave, nalaze se opis i slike objave (ako ih objava ima), ispod slike s lijeve strane nalazi se brojač oznaka sviđanja, te s desne strane brojač komentara. Ispod brojača nalaze se 3 gumba, s lijeva na desno, to su gumb za oznaku sviđanja označen praznom ikonicom srca ako objava nije označena sa sviđanjem ili punom ikonicom srca ako je objava označena sa sviđanjem, gumb za otvaranje pregleda komentara označen oblačićem s dvije crte, te gumb za otvaranje izbornika za dijeljenje objave označen ikonicom s tri

povezana kruga. Na samom dnu objave s lijeva na desno nalaze se slika profila korisnika, polje za upis komentara i gumb za objavu komentara označen ikonicom papirnatog aviona.



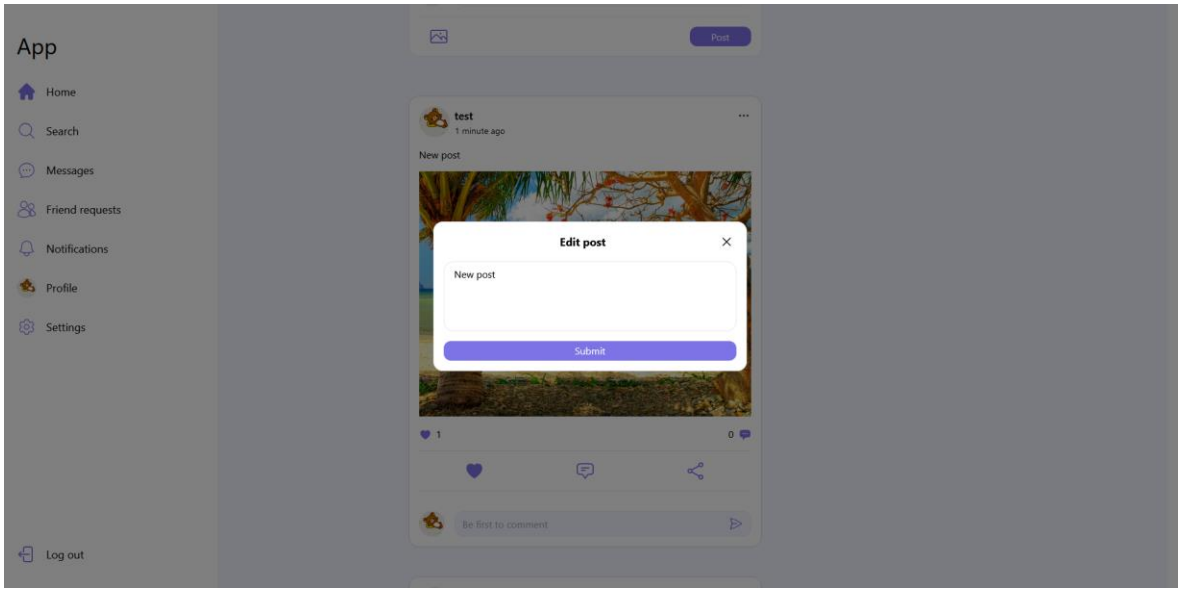
Slika 4.5 Prikaz objave

Klikom na gumb za upravljanje objavom (Slika 4.5) otvara se izbornik za upravljanje objavom (Slika 4.6), klikom na prvi gumb označen crvenim tekstom „Remove“ korisnik može obrisati objavu, klikom na drugi gumb označen tekstom „Edit“ može otvoriti izbornik za uređivanje teksta, te klikom na zadnji gumb označen tekstom „Cancel“ može zatvoriti meni.



Slika 4.6 Izbornik za upravljanje objavom

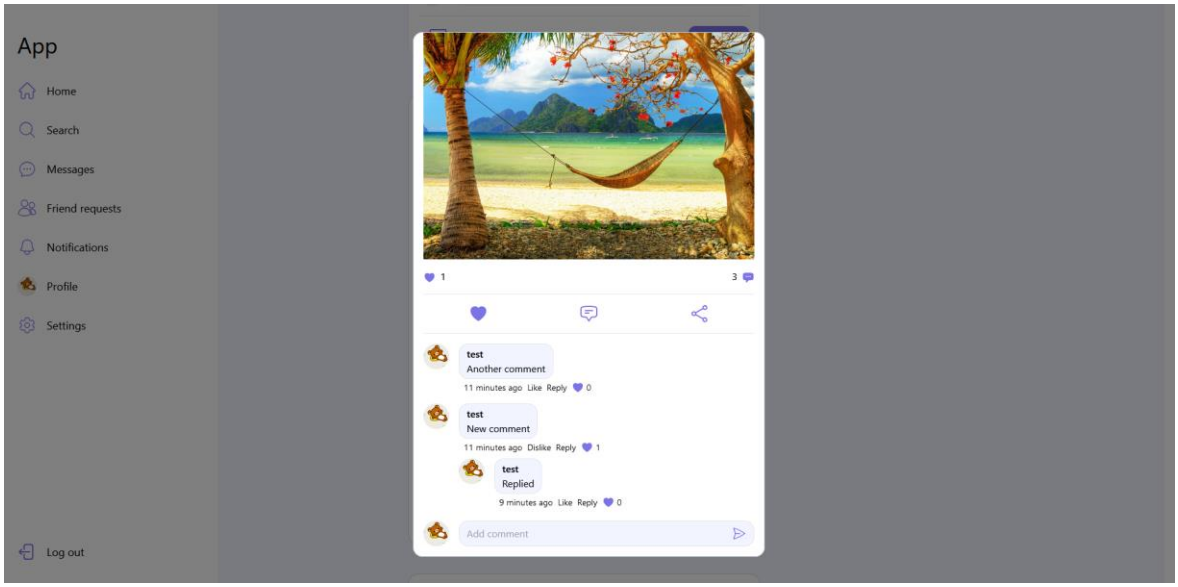
Klikom na gumb označen tekstom „Edit“ (Slika 4.6), otvara se izbornik za uređivanje teksta (Slika 4.7). U izborniku na vrhu nalazi se s desne strane gumb za zatvaranje izbornika označen ikonicom znaka „x“, u sredini se nalazi polje za upis opisa objave, te na samom dnu gumb za potvrdu promjene označen ljubičastom bojom i tekstom „Submit“.



Slika 4.7 Izbornik za uređivanje opisa objave

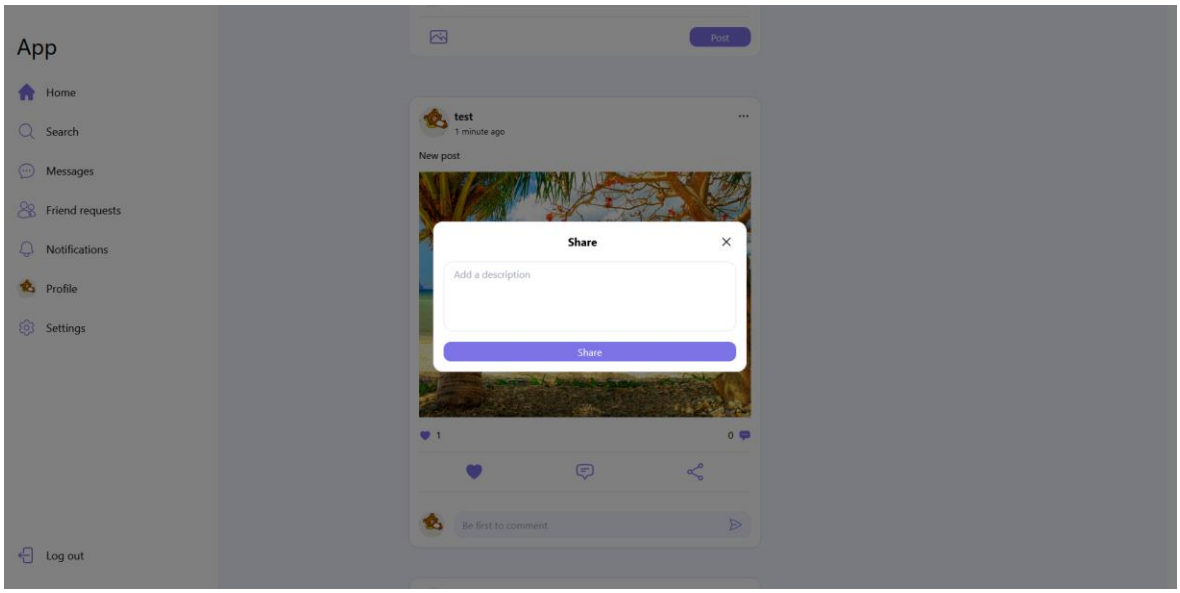
Klikom na gumb označen oblačićem s dvije crte (Slika 4.5), otvara se pregled komentara objave (Slika 4.8), gdje korisnik može vidjeti sve komentare na objavu. Komentar sadrži profilnu sliku korisnika koji je objavio komentar, njegovo korisničko ime,

tekst komentara, vrijeme objave, gumb za označavanje sviđanja označen tekstom „Like“ ako nije označen sa sviđanjem ili „Dislike“ ako je označen sa sviđanjem, gumb za odgovaranje na komentar označen tekstom „Reply“ i broj oznaka sviđanja.



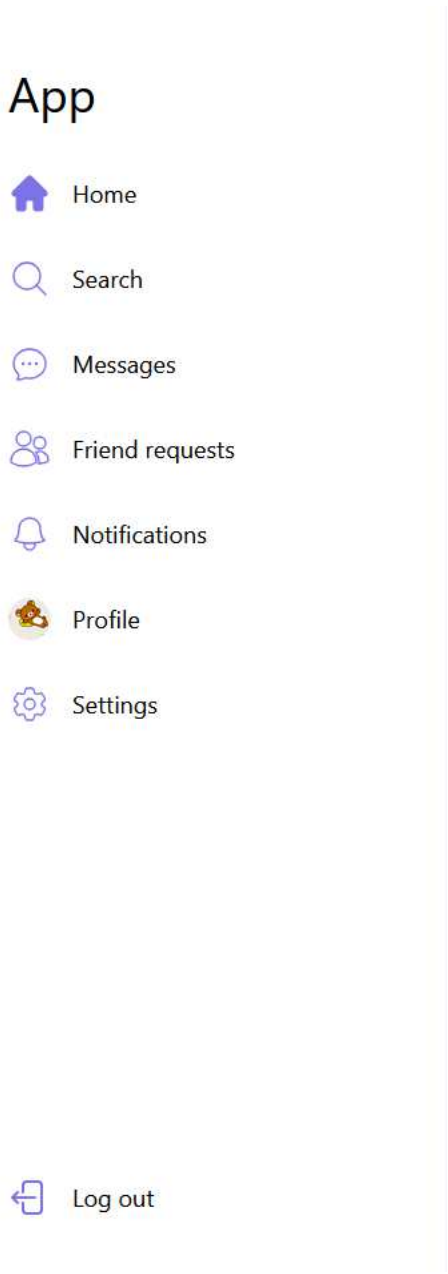
Slika 4.8 Pregled komentara objave

Klikom na gumb označen s tri povezana kruga (Slika 4.5) otvara se izbornik za dijeljenje objave (Slika 4.9), koji je identičan izborniku za uređivanje opisa objave (Slika 4.7), koji je opisan ranije.



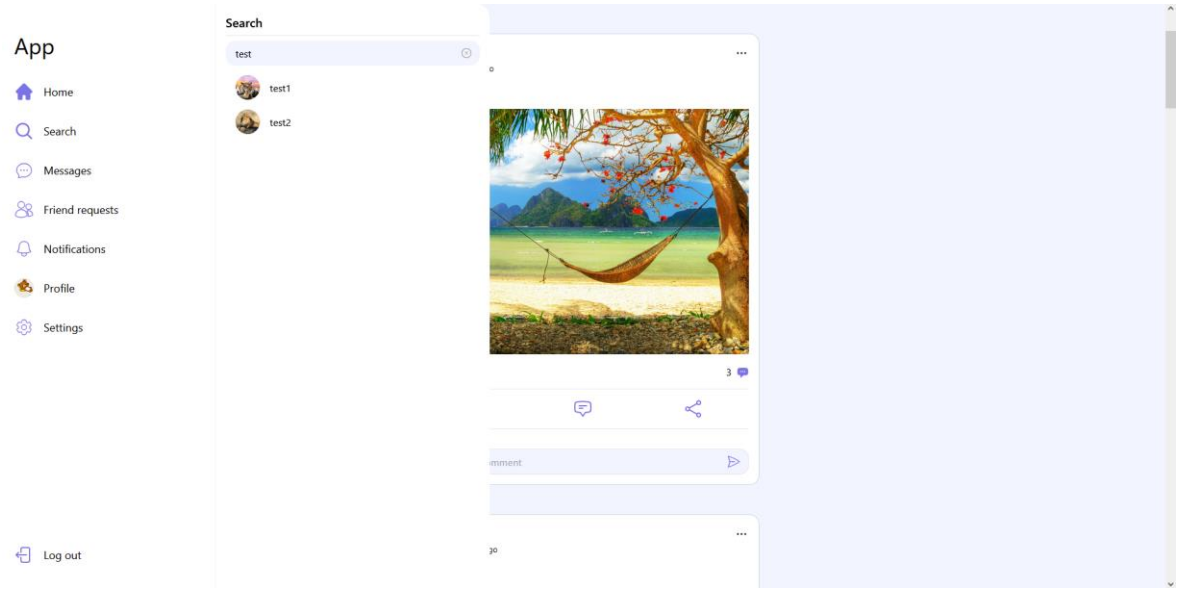
Slika 4.9 Izbornik za dijeljenje objave

Navigacijska traka (Slika 4.10) redom sadrži poveznicu na početnu stranicu označenu ikonicom kuće i tekstom „Home“, gumb koji otvara prozor za pretragu korisničkih profila označen ikonicom povećala i tekstom „Search“, poveznicu na razgovore korisnika sa svojim prijateljima označenu ikonicom oblačića poruke i tekstom „Messages“, gumb koji otvara prozor za pregledavanje i upravljanje zahtjevima za prijateljstvo označen ikonicom dva čovjeka i tekstom „Friend requests“, gumb koji otvara prozor za pregledavanje obavijesti označen ikonicom zvona i tekstom „Notifications“, poveznicu na profil korisnika označen profilnom slikom korisnika i tekstom „Profile“, poveznicu na postavke profila korisničkog računa označenom ikonicom zupčanika i tekstom „Settings“, te na samom dnu gumb za odjavu označen pravokutnikom i lijevom strelicom i tekstom „Log out“.



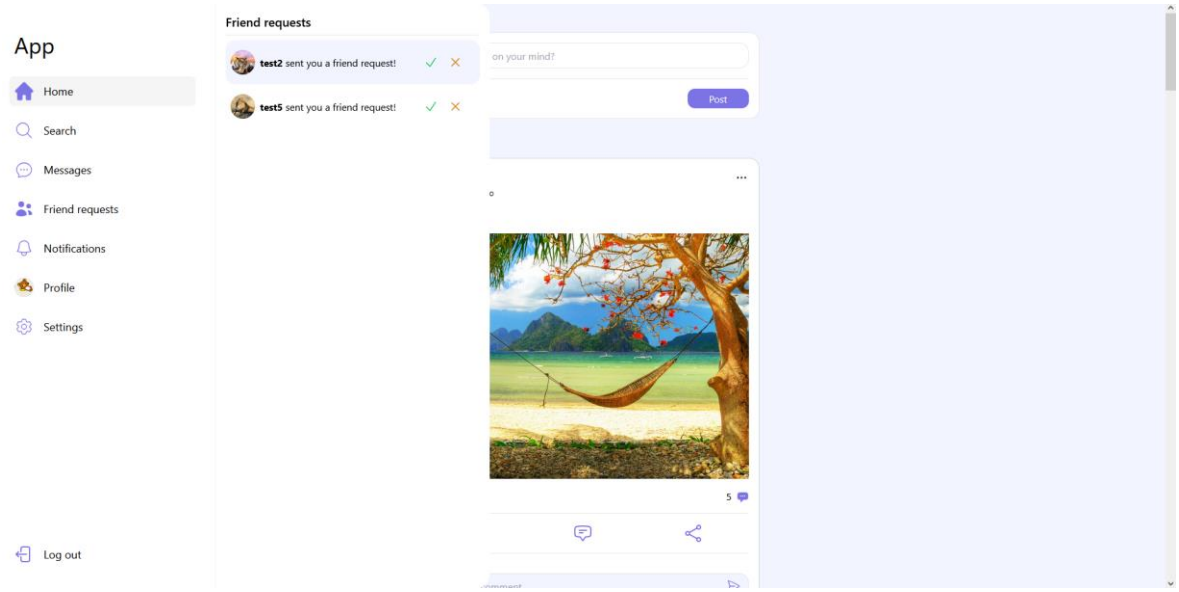
Slika 4.10 Navigacijska traka

Klikom na gumb označen ikonicom povećala i tekstom „Search“ (Slika 4.10) otvara se prozor za pretragu korisničkih profila (Slika 4.11), u kojem se nalazi polje za upis korisničkog imena po kojemu se pretražuju korisnički profili. Ako takvi postoje, prikaže se lista pronađenih korisničkih profila, te se za svaki prikaže poveznica na pojedinačni korisnički profil u obliku profilne slike korisnika i njegovog korisničkog imena.



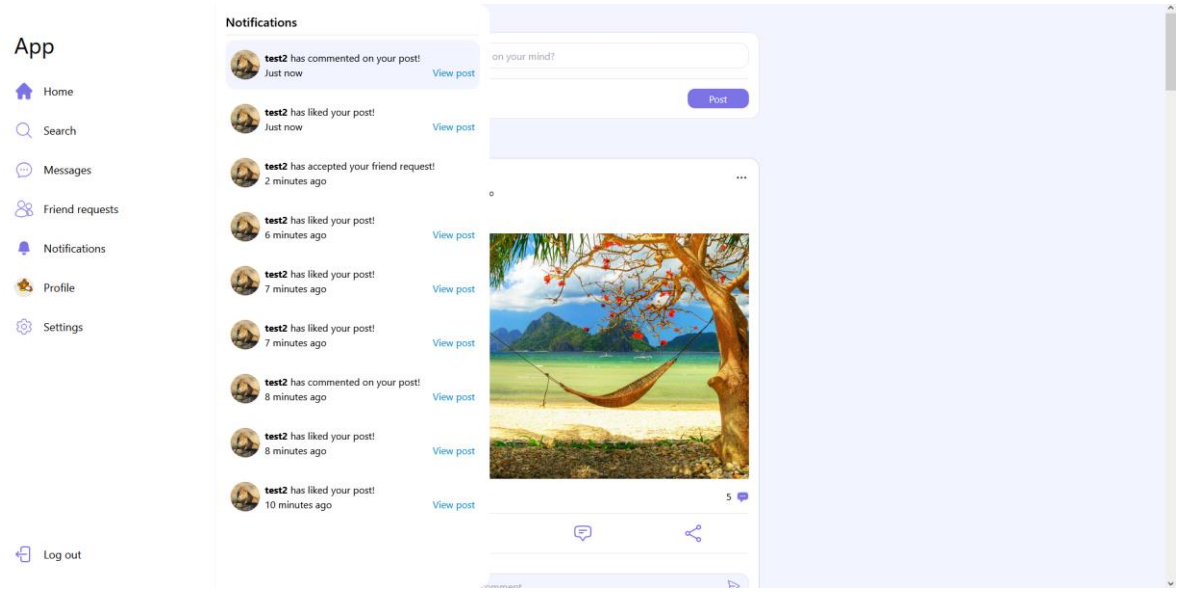
Slika 4.11 Prozor za pretragu korisničkih profila

Klikom na gumb označen ikonicom dva čovjeka i tekstom „Friend requests“ (Slika 4.10) otvara se prozor za pregledavanje i upravljanje zahtjevima za prijateljstvo (Slika 4.12), u kojem se nalaze zahtjevi za prijateljstvo, koji sadrže profilnu sliku korisnika koji je uputio zahtjev za prijateljstvo, njegovo korisničko ime koje služi kao poveznica na njegov korisnički profil, gumb za prihvaćanje zahtjeva označen zelenom ikonicom kvačice i gumb za odbijanje zahtjeva označen crvenom ikonicom znaka „x“. Ako je zahtjev za prijateljstvo pročitan, imat će bijelu pozadinu, a ako je nepročitan imat će ljubičastu pozadinu.



Slika 4.12 Prozor za pregledavanje i upravljanje zahtjevima za prijateljstvo

Klikom na gumb označen ikonicom zvona i tekstom „Notifications“ (Slika 4.10) otvara se prozor za pregledavanje obavijesti (Slika 4.13), u kojem se nalaze obavijesti korisniku od strane njegovih prijatelja. Obavijest ima plavu pozadinu ako je nepročitana, a ako je pročitana pozadina je bijela. U obavijesti se nalazi profilna slika korisnika koji je prouzrokovao slanje obavijesti i njegovo korisničko ime, tekst obavijesti, vrijeme obavijesti i u slučaju da je obavijest povezana s objavom, sadrži poveznicu označenu tekstom „View post“, koja vodi na objavu.

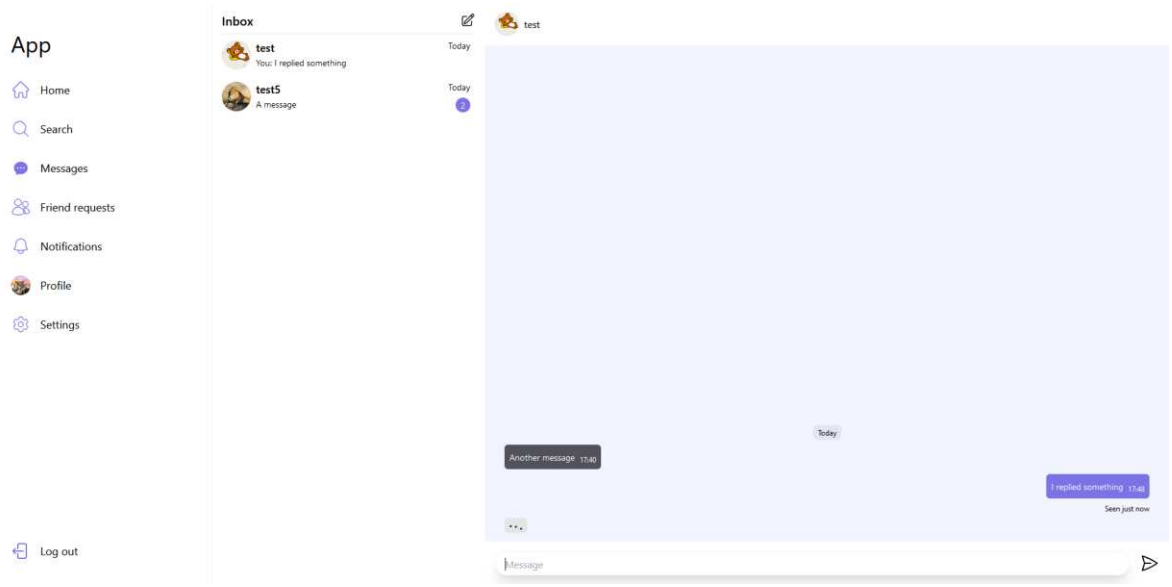


Slika 4.13 Prozor za pregledavanje obavijesti

4.2.3. Razgovori i poruke

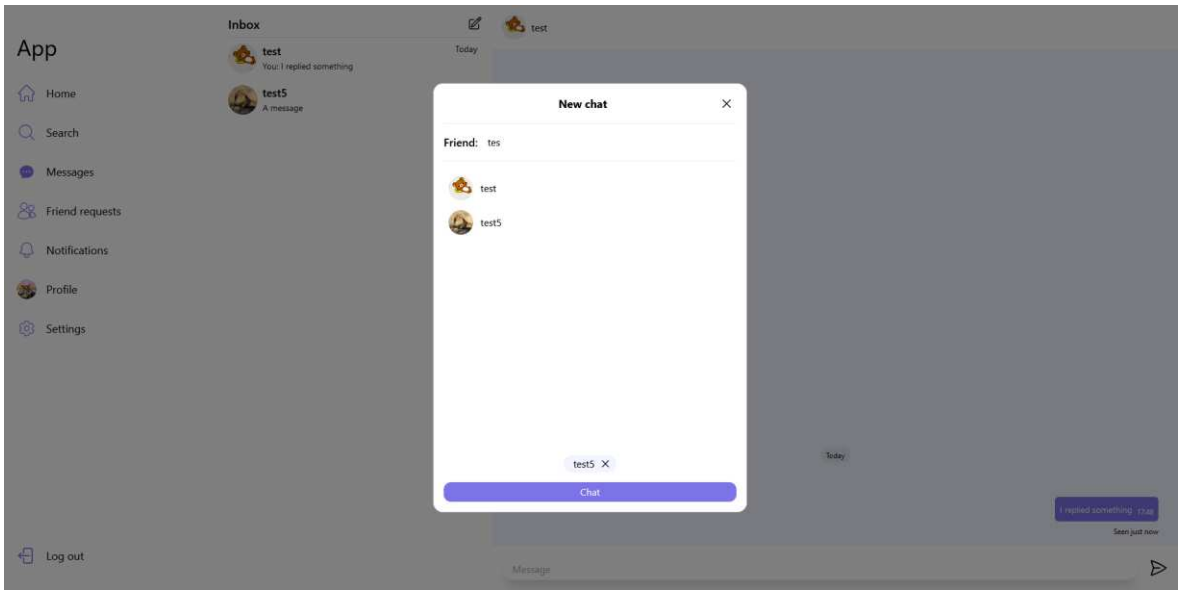
Na stranici razgovora (Slika 4.14) na lijevoj strani nalaze se prijašnji razgovori korisnika sa svojim prijateljima, sortirani od najnovijeg prema najstarijem. Svaki razgovor ima prikazano korisničko ime i profilnu sliku prijatelja, zadnju poruku u razgovoru, dan kad je zadnja poruka poslana i broj nepročitanih poruka. Iznad prikazanih razgovora nalazi se gumb za otvaranje prozora za pronalazak prijatelja i otvaranje razgovora označen ikonicom olovke. Klikom na razgovor, otvorit će se detaljan prikaz s desne strane. U detaljnom prikazu razgovora na samom vrhu se nalazi korisničko ime i profilna slika prijatelja, potom u sredini se nalazi prikaz međusobnih poruka, te na samom dnu se nalazi polje za upis teksta poruke i gumb za slanje poruke označen ikonicom papirnatog aviona. U prikazu poruka, poruke su grupirane po danu kad su poslane, a same poruke sadrže tekst

i vrijeme kad je poruka poslana. Na samom dnu prikaza poruka nalazi se vrijeme kad je zadnja poruka pročitana, te tri točkice koje se prikazuju kad drugi korisnik tipka poruku.



Slika 4.14 Stranica s razgovorima

Klikom na gumb označen ikonicom olovke (Slika 4.14) otvara se prozor za pronalazak prijatelja i otvaranje razgovora (Slika 4.15). U tom prozoru, na samom vrhu s desne strane nalazi se gumb za zatvaranje prozora označen ikonicom znaka „x“, u sredini nalazi se polje za upis korisničkog imena prijatelja kojeg želimo pronaći, te ispod tog polja se prikazuju korisnička imena i profilne slike pronađenih prijatelja. Klikom na pronađenog prijatelja, na dnu se prikaže njegovo korisničko ime, što označava da će se klikom na gumb označen tekstom „Chat“ na samom dnu, otvoriti razgovor s tim prijateljem.

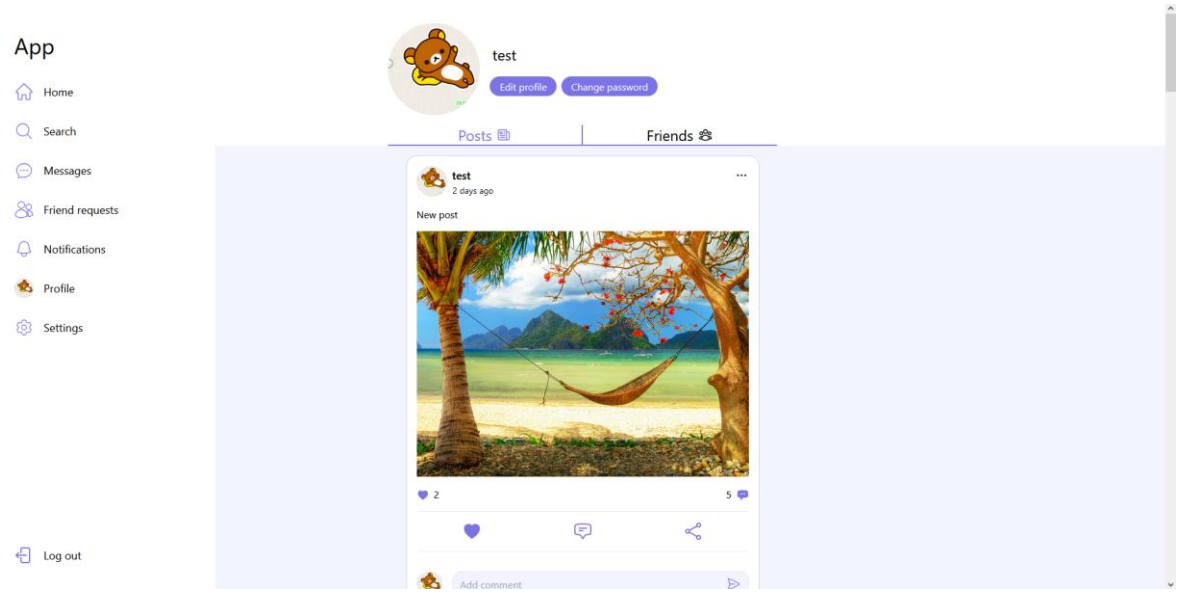


Slika 4.15 Prozor za pronalazak prijatelja i otvaranje razgovora

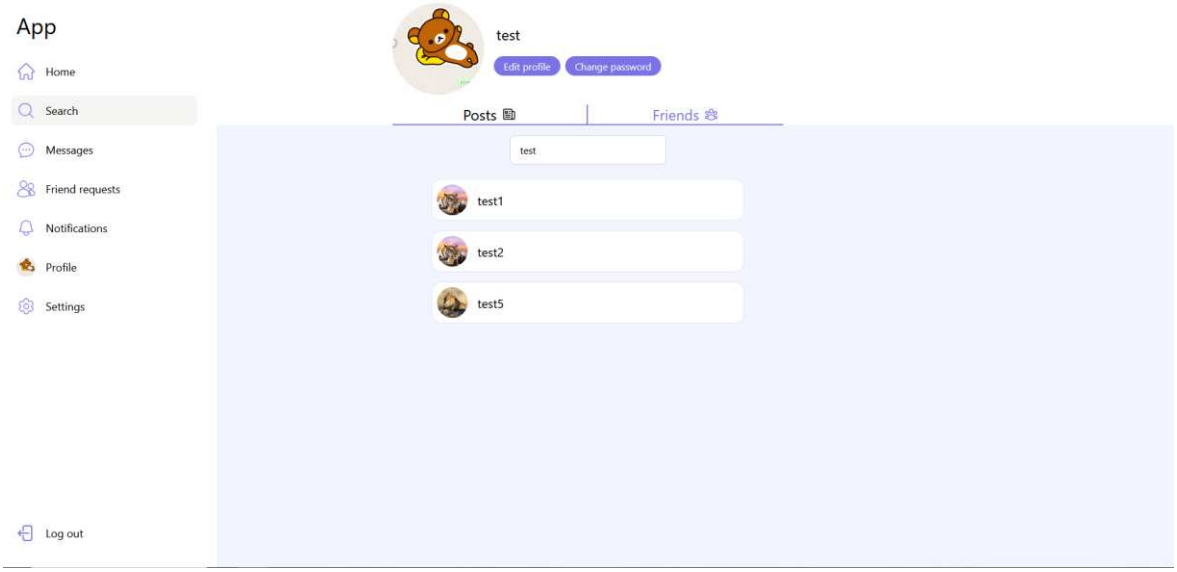
4.2.4. Korisnički profil

Na stranici korisničkog profila korisnik može pregledati svoje objave, pregledati svoje prijatelje, promijeniti sliku profila, promijeniti korisničke podatke i promijeniti lozinku. Na slici 4.16 na samom vrhu prikazana je slika profila korisnika, koja je gumb koji otvara prozor za upravljanje profilnom slikom, pored slike profila nalazi se korisničko ime, te ispod njega poveznica na stranicu za uređivanje profilnih podataka označenu tekstom „Edit Profile“ i poveznica na stranicu za promjenu korisničke lozinke označenu tekstom „Change Password“. U slučaju da to nije korisnikov profil, ne prikazuju se poveznice za upravljanje korisničkim podacima i lozinkom, nego se prikazuje status prijateljstva (Slika 4.18). Nadalje, nalaze se poveznica za prikaz objava korisnika označena tekstom „Posts“ i ikonicom novina i poveznica za prikaz prijatelja korisnika (Slika 4.17.)

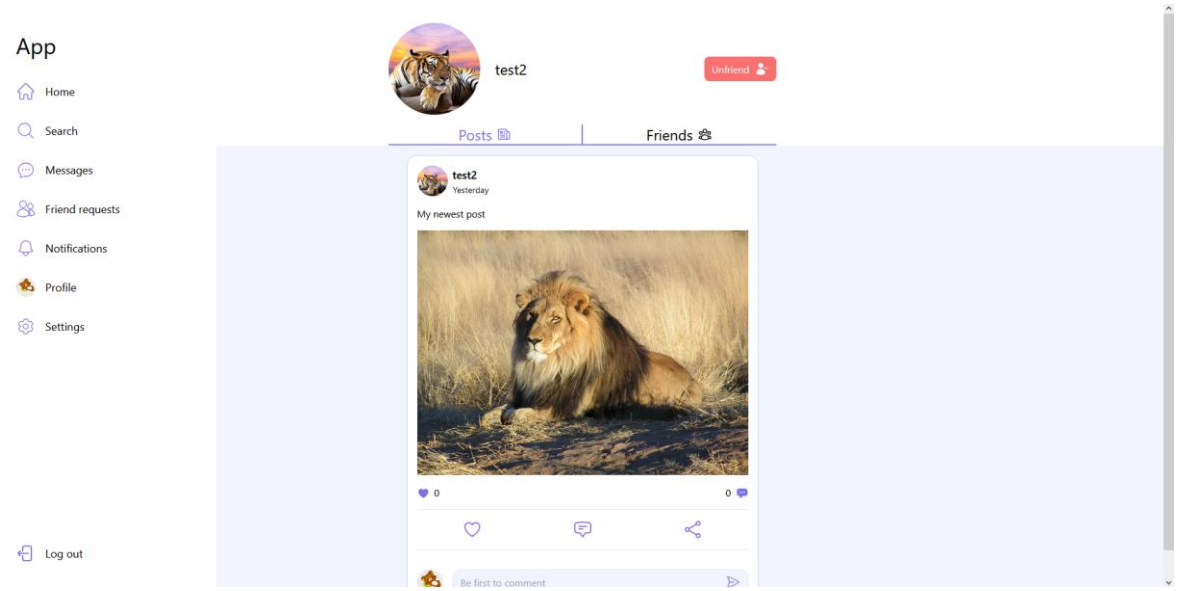
označena tekstom „Friends“ i ikonicom tri čovjeka. Ispod tih poveznica nalazi se prikaz objava ili prijatelja ovisno na kojoj se stranici nalazimo.



Slika 4.16 Stranica korisničkog profila s prikazom objava

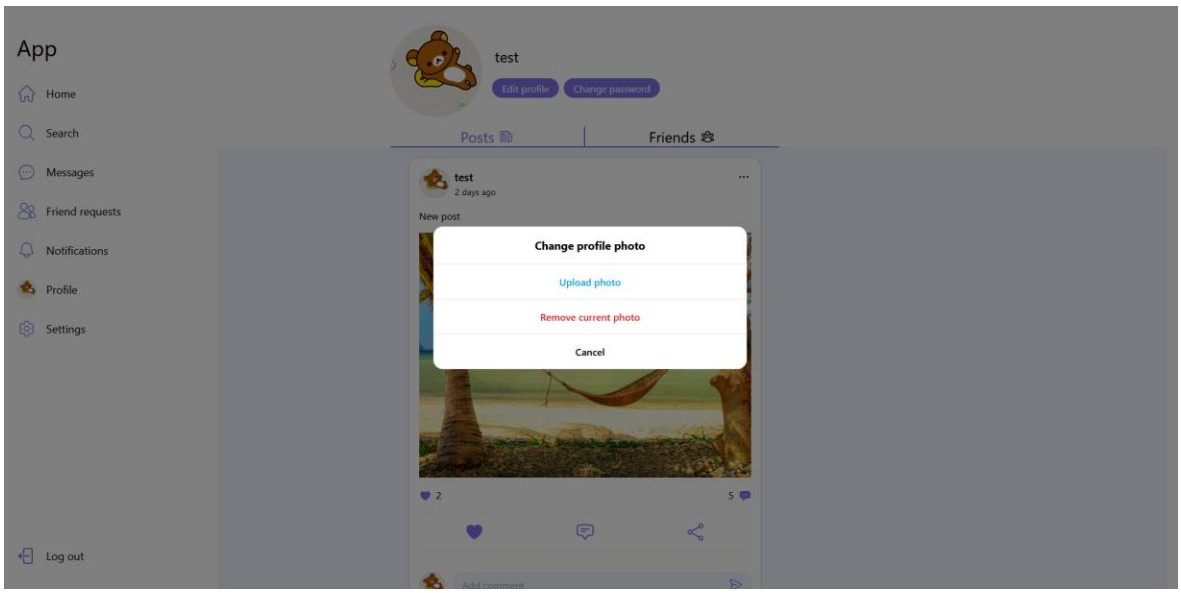


Slika 4.17 Stranica korisničkog profila s prikazom prijatelja



Slika 4.18 Stranica korisničkog profila prijatelja

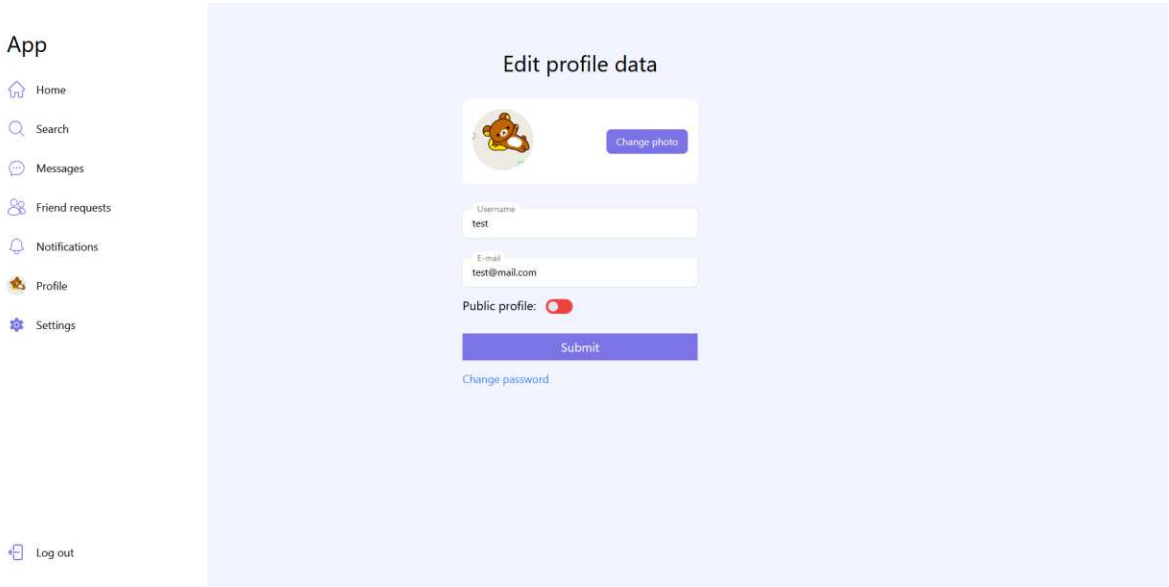
Klikom na gumb slike profila korisnika (Slika 4.16) otvara se prozor za upravljanje slikom profila (Slika 4.19). U tom prozoru, nalaze se tri gumba. Prvi gumb označen plavim tekstom „Upload photo“ otvara prozor za pretraživanje datoteke za učitavanje slike profila. Drugi gumb označen crvenim tekstom „Remove current photo“ briše postojeću sliku profila. Treći gumb označen tekstom „Cancel“ zatvara prozor.



Slika 4.19 Prozor za upravljanje slikom profila

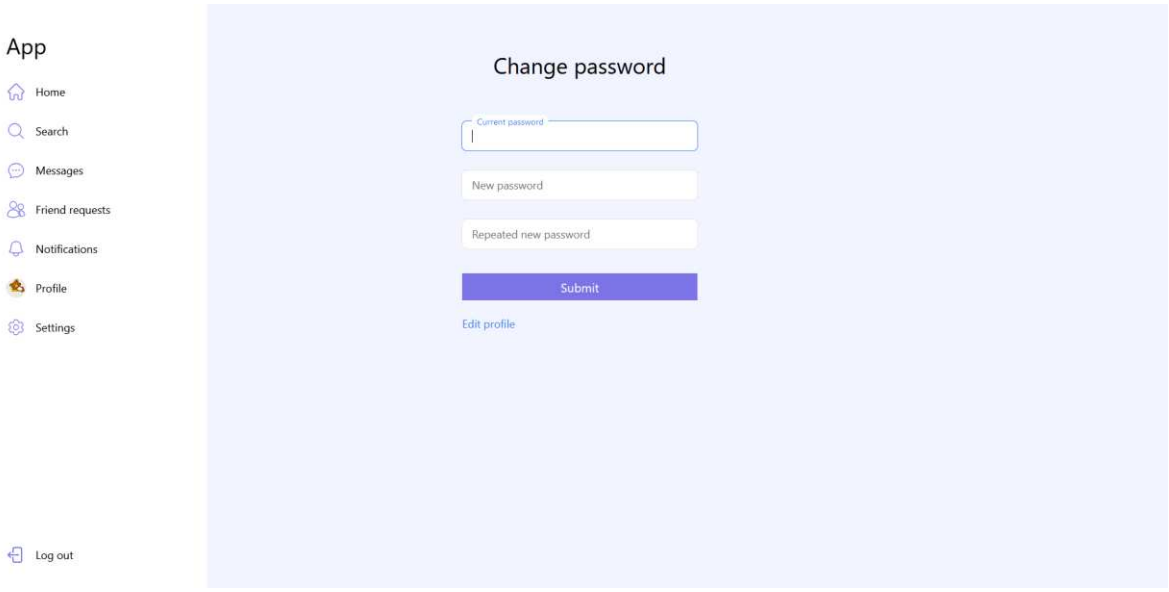
Klikom na poveznicu označenu tekstom „Edit profile“ (Slika 4.16) otvara se stranica za uređivanje profilnih podataka (Slika 4.20). Na stranici na vrhu se nalazi gumb koji otvara prozor za upravljanje slikom profila označen tekstom „Change photo“. Potom

slijedi polje za upis korisničkog imena i e-maila, gumb koji postavlja vidljivost korisničkog profila, koji može biti javan ili privatan, gumb koji potvrđuje promjene označen tekstom „Submit“ i na kraju se nalazi poveznica na stranicu za promjenu lozinke označena tekstom „Change password“.



Slika 4.20 Stranica za uređivanje profilnih podataka

Klikom na poveznicu označenu tekstom „Change password“ (Slika 4.16 ili Slika 4.20) otvara se stranica za promjenu korisničke lozinke (Slika 4.21). Na toj stranici, nalaze se tri polja, gumb i poveznica. Prvo polje služi za upis trenutne lozinke, potom slijedi polje za upis nove lozinke, a zadnje polje služi za unos ponovljene nove lozinke. Gumb označen tekstom „Submit“ služi za potvrdu promjene lozinke, te se na samom dnu nalazi poveznica na stranicu za uređivanje profilnih podataka (Slika 4.20) označena tekstom „Edit profile“.



Slika 4.21 Stranica za promjenu korisničke lozinke

Zaključak

Cilj ovog rada bio je razvoj web aplikacije društvene mreže. Korištenjem modernih tehnologija, razvijena je web aplikacija koja se sastoji od baze podataka, poslužitelja i klijenta. Implementirane su funkcionalnosti poput stvaranja korisničkog računa, objavljivanje, sviđanje i komentiranje sadržaja, dodavanje prijatelja, pretraživanje drugih korisnika, slanje poruka prijateljima i pregledavanje profila drugih korisnika.

U prvom dijelu razvoja je modelirana i implementirana baza podataka koristeći MongoDB nerelacijsku bazu podataka. Potom u drugom dijelu implementiran je poslužitelj koristeći tehnologije Node.js, Express.js, WebSocket i Prisma ORM koja služi za komunikaciju s bazom podataka. Poseban naglasak stavljen je na WebSocket protokol, koji omogućava implementaciju istovremene komunikacije između korisnika. Zadnje je razvijena klijentska aplikacija pomoću React.js i TailwindCSS programskih okvira.

Moguće nadogradnje ove aplikacije su dodatne funkcionalnosti, poput grupnih razgovora, audio i video poziva, oglasa, algoritama za personalizirani sadržaj, prijenos uživo, te optimizacija performansi za veći broj korisnika.

Kod razvoja ovog rada, javljali su se problemi u obliku proširivanja i restrukturiranja modela baze podataka zbog implementiranja novih funkcionalnosti, te odlučivanja koje funkcionalnosti će se implementirati putem WebSocket protokola, a koje putem HTTP protokola. Oba problema riješena su istraživanjem i analiziranjem, kako bi razvijen program bio što razumljiviji, lakši za održavanje i omogućavao lakšu nadogradnju.

Literatura

[1] MongoDB. Poveznica: https://www.mongodb.com/; pristupljeno 20.8.2024.

[2] PrismaORM. Poveznica: https://www.prisma.io/; pristupljeno 20.8.2024.

[3] JavaScript. Poveznica: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript; pristupljeno 20.8.2024.

[4] TypeScript. Poveznica: https://www.typescriptlang.org/; pristupljeno 20.8.2024.

[5] Node.js. Poveznica: https://nodejs.org/en; pristupljeno 20.8.2024.

[6] Express.js. Poveznica: https://expressjs.com/; pristupljeno 20.8.2024.

[7] WebSocket. Poveznica: https://en.wikipedia.org/wiki/WebSocket; pristupljeno 20.8.2024.

[8] Socket.IO. Poveznica: https://socket.io/; pristupljeno 20.8.2024.

[9] React.js. Poveznica: https://react.dev/; pristupljeno 20.8.2024.

[10] TailwindCSS. Poveznica: https://tailwindcss.com/; pristupljeno 20.8.2024.

[11] Social media. Poveznica: https://en.wikipedia.org/wiki/Social_media; pristupljeno 20.8.2024.

Sažetak

Razvoj web aplikacije društvene mreže

Ovaj rad opisuje razvoj društvene mreže koja omogućava korisnicima stvaranje korisničkog računa, objavljivanje, sviđanje i komentiranje sadržaja, dodavanje prijatelja, pretraživanje drugih korisnika, slanje poruka prijateljima i pregledavanje profila drugih korisnika. U radu su opisane korištene tehnologije, te sama izvedba baze podataka, poslužitelja i klijenta. Poseban naglasak stavlja se na istovremenu komunikaciju, koja je od velike važnosti, jer pridonosi dinamičnijoj aplikaciji i većoj angažiranosti korisnika u samoj aplikaciji. Prvo je osmišljena, modelirana i kreirana nerelacijska baza podataka koristeći MongoDB. Potom je napravljena arhitektura poslužitelja, po kojoj je i isprogramiran koristeći Node.js okruženje, Express.js programski okvir, Prisma ORM alat i WebSocket protokol. Na samom kraju razvijena je klijentska aplikacija koristeći programske okvire React.js i TailwindCSS.

Ključne riječi

web aplikacija, društvena mreža, WebSocket, Node.js, Express.js, Prisma ORM, MongoDB, React.js, TailwindCSS

Summary

The development of social network web application

This thesis describes the development of a social network that enables users to create user account, post, like, and comment on content, add friends, search for other users, send messages to friends, and view other users' profiles. The thesis describes the technologies used, as well as the implementation of the database, server, and client. Special emphasis is placed on real-time communication, which is of great importance as it contributes to a more dynamic application and greater user engagement. The process began with designing, modeling, and creating a non-relational database using MongoDB. Then, the server architecture was developed and programmed using the Node.js environment, Express.js framework, Prisma ORM tool, and WebSocket protocol. Finally, the client application was developed using the React.js and TailwindCSS frameworks.

Keywords

web application, social network, WebSocket, Node.js, Express.js, Prisma ORM, MongoDB, React.js, TailwindCSS