PRIVATNA SOCIJALNA MREŽA ZA ZAPOSLENIKE TVRTKI

Vuković, Josip

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Primijenjeno računarstvo

JOSIP VUKOVIĆ

Z A V R Š N I R A D

Privatna socijalna mreža za zaposlenike tvrtki

Split, rujan 2023.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Primijenjeno računarstvo

Predmet: Programsko inženjerstvo

Z A V R Š N I R A D

Kandidat: Josip Vuković

Naslov rada: Privatna socijalna mreža za zaposlenike tvrtki

Mentor: mr.sc. Karmen Klarin, viši predavač

Split, rujan 2023.

Sadržaj

SAŽETAK....1
SUMMARY....2
1 UVOD....3
2 TEHNOLOGIJE....4
2.1 Frontend....4
2.1.1 React....4
2.1.2 React Router....5
2.1.3 Redux Toolkit....6
2.1.4 Redux Persist....8
2.1.5 Formik i yup....10
2.2 Backend....11
2.2.1 Node.js....13
2.2.2 Express.js....13
2.2.3 Mongoose....14
2.2.4 JSON Web Token (JWT)....15
2.2.5 Multer....16
2.2.6 MongoDB....16
2.3 UML Dijagrami....18
2.3.1 Use case dijagram....18
2.3.2 Dijagram klasa....18
2.4 Agilni način rada....19
3 DETALJAN OPIS WEB APLIKACIJE....21
3.1 Stranica za registraciju....21
3.2 Stranica za prijavu....22

3.3 Naslovna stranica 23
3.4 Navigacijska traka 24
3.5 Korisnička kartica 24
3.6 Lista pratitelja 25
3.7 Kartica objava 26
3.8 Kartica za kreiranje objave 27
3.9 Profilna stranica 29
3.10 Mobilni prikaz 29
3.11 Tema aplikacije 33
3.12 Korisnik administrator 33
4 IMPLEMENTACIJA 36
4.1 *Backend* kôd 36
4.1.1 Model korisnika 36
4.1.2 Model objave 38
4.1.3 Verifikacija tokena 39
4.1.4 Metoda prijave 40
4.1.5 Metoda registracije 41
4.1.6 Kreiranje objave 42
4.1.7 Lajk objave 43
4.1.8 Dodavanje komentara na objavu 44
4.1.9 Prati/otprati korisnika 45
4.1.10 Ažuriranje korisničke kartice 47
4.1.11 Brisanje korisnika 48
4.1.12 Rute 49
4.1.13 Spajanje s bazom podataka 50
4.1.14 Verzije korištenih biblioteka za *backend* 51

4.2 *Frontend* kôd 51
4.2.1 Početna stranica 51
4.2.2 Metoda prijave pozivom *backend* API-ja 52
4.2.3 Metoda registracije pozivom backend API-ja 54
4.2.4 Shema za prijavu i registraciju 55
4.2.5 Kalup profilne slike korisnika 55
4.2.6 Redux kôd 56
4.2.7 Rute web aplikacije 58
4.2.8 Učitavanje slike, videozapisa ili GIFa 58
4.2.9 Filter objava na osnovu izabranog tima 60
4.2.10 Metoda komentiranja pozivom *backend* API-ja 61
4.2.11 Metoda brisanja objava pozivom *backend* API-ja 62
4.2.12 Verzije korištenih biblioteka za *frontend* 63
5 ZAKLJUČAK 64
6 LITERATURA 65

SAŽETAK

Kroz ovaj rad su detaljno opisane tehnologije pomoću kojih je izrađena ova privatna socijalna mreža za zaposlenike tvrtki zvana *Techbook*. Svaka tehnologija je zasebno opisana te je objašnjena njena uloga i odabir. Pomoću UML dijagrama je prikazan kompletan način rada sustava od korisničkog do administratorskog sučelja. Kroz rad je istaknut agilni način rada kao fleksibilan i iterativni pristup razvoju softvera koji naglašava suradnju, prilagodljivost i usmjerenost na korisnika. Komponente web aplikacije su slikovno prikazane i objašnjene na jednostavan način. Istaknute su funkcionalnosti web aplikacije i njihov put od implementacije do testiranja i finalnog oblika. Detaljan opis kôda omogućava razumijevanje implementacije određenih komponenti bez predznanja korištenja *frontend* i *backend* tehnologija.

Ključne riječi: *backend, frontend, MongoDB*, socijalna mreža, web aplikacija

SUMMARY

Private social network for company employees

This paper describes in detail the technologies used to create this private social network for company employees called Techbook. Each technology is described separately, and its role and selection are explained. Using the UML diagram, the complete mode of operation of the system, from the user interface to the administrator interface, is shown. The paper highlights the agile way of working as a flexible and iterative approach to software development that emphasizes collaboration, adaptability, and user orientation. The components of the web application are illustrated and explained in a simple way. The functionalities of web applications and their path from implementation to testing and final form are highlighted. A detailed description of the code makes it possible to understand the implementation of certain components without prior knowledge of the use of front-end and back-end technology.

Keywords: backend, frontend, MongoD, social media, web application

1 UVOD

Techbook je naziv za privatnu socijalnu mrežu za zaposlenike tvrtki koja omogućuje zaposlenicima da kreiraju, lajkaju i komentiraju objave iz poslovnog života te ih podijele unutar svog tima ili na razini cijele organizacije. Web aplikacija omogućuje korisnicima izradu korisničkog računa. Svaki korisnik pri izradi računa upisuje svoje ime i prezime, lokaciju, trenutnu poziciju unutra tvrtke, tim, email adresu i lozinku. Web aplikacija je uvijek dostupna te je ideja da korisnici imaju pristup stranici samo preko svojih poslovnih uređaja kao što su laptopi i pametni uređaji. Osim za web, aplikacija je optimizirana i za pametne uređaje te prati sve moderne standarde dizajna. Kroz sam rad korištene su najmodernije tehnologije i načini izrade web aplikacija. Tehnologije su podijeljene na *backend* i *frontend* kategorije. *Frontend* je sve ono što korisnik vidi i s čim vrši interakciju, dok je *backend* ono što se događa u pozadini, što korisnik ne vidi. Svaka informacija koju web aplikacija koristi spremljena je u bazu podataka te je velika pažnja posvećena sigurnosti korisnika. Velik naglasak je stavljen na modularnost kôda jer svaka tvrtka ima svoj način rada i posebne zahtjeve.

Cilj ove web aplikacije je povezivanje zaposlenika na poslovnom i privatnom planu što dovodi do veće učinkovitosti i povezanosti u svakodnevnom radu. Kroz drugo poglavlje opisane su sve korištene *backend* i *frontend* tehnologije, prikazani su UML dijagrami te je opisan agilni način rada. U trećom poglavlju prikazan je detaljan opis web aplikacije pomoću slika koje detaljno prikazuju izgled i funkcionalnosti. Kroz posljednje četvrto poglavlje prikazana je i objašnjena implementacija svih funkcionalnosti kroz kôd same web aplikacije.

2 TEHNOLOGIJE

2.1 Frontend

Frontend se odnosi na razvoj web stranice ili aplikacije na strani klijenta, s fokusom na korisničko sučelje i korisničko iskustvo. Uključuje dizajniranje i implementaciju vidljivih komponenti s kojima korisnici komuniciraju. *Frontend* programeri koriste razne jezike, tehnologije i alate za stvaranje strukture, izgleda i ponašanja implementiranih elemenata. Od njih se očekuje vizualno privlačno, osjetljivo (*eng. responsive)* i intuitivno sučelje. Razvoj sučelja igra ključnu ulogu u pružanju privlačnog i korisničkog digitalnog iskustva.

2.1.1 React

React je naširoko korištena *JavaScript* biblioteka koja razvojnim programerima omogućuje izradu korisničkih sučelja za web aplikacije. Koristi pristup temeljen na komponentama, omogućujući stvaranje komponenti korisničkog sučelja koje se mogu ponovno koristiti i održavati. Ove samostalne jedinice kôda objedinjuju određene dijelove korisničkog sučelja, omogućujući jednostavnu kompoziciju za formiranje kompliciranih struktura. Programeri opisuju željeni izgled korisničkog sučelja u bilo kojem trenutku, dok se *React* bavi zadatkom ažuriranja stvarnog korisničkog sučelja kako bi odgovaralo tom opisu. Jedna od najbitnijih značajki *Reacta* je implementacija virtualnog DOM-a (*Document Object Model*). DOM omogućuje *Reactu* da učinkovito ažurira samo dijelove korisničkog sučelja koji su promijenjeni, izbjegavajući potrebu za potpunim ponovnim prikazom. Ova optimizacija poboljšava performanse i odziv u aplikacijama. *React* koristi koncept "stanja" za upravljanje dinamičkim podacima unutar komponenti. Stanje predstavlja trenutno stanje komponente i može se mijenjati tijekom vremena. Kad god se stanje promijeni, *React* automatski ponovno prikazuje komponentu i ažurira korisničko sučelje u skladu s tim. Ovo omogućuje programerima stvaranje interaktivnih i dinamičnih korisničkih sučelja bez napora. Drugi temeljni aspekt *Reacta* je korištenje "rekvizita". Rekviziti olakšavaju jednosmjerni protok podataka od nadređenih pa do podređenih komponenti. To omogućuje stvaranje ponovno upotrebljivih i modularnih komponenti koje mogu prilagoditi svoje ponašanje na temelju rekvizita koje dobiju. *React* se može pohvaliti raznolikim ekosustavom

alata i biblioteka koje proširuju njegove mogućnosti. To uključuje *React Router* za usmjeravanje na strani klijenta i *Redux* za upravljanje stanjem aplikacije, uz mnoge druge alate.

2.1.2 React Router

React Router je popularna biblioteka koja se koristi za usmjeravanje u *React* aplikacijama. Omogućuje stvaranje jednostraničnih aplikacija s više prikaza, od kojih svaki odgovara različitoj URL putanji. *React Routerom,* kao što je prikazano na ispisu 1, se dinamički ažurira sadržaj aplikacije na temelju trenutnog URL-a, bez pokretanja potpunog osvježavanja stranice. *S*adrži skup komponenti koje se mogu koristiti za definiranje ruta u aplikaciji.

Neke od ključnih komponenta su:

- <*BrowserRouter*>: Koristi kao korijen konfiguracije usmjeravanja. Sluša promjene u URL-u i iscrtava odgovarajuće komponente na temelju definiranih ruta.
- <*Route*>: Koristi se za definiranje pojedinačnih ruta. Iscrtava komponentu kada trenutni URL odgovara njenom putu. Možete navesti putanju kao niz ili koristiti napredne uzorke podudaranja pomoću regularnih izraza.
- <*Switch*>: Koristi se za grupiranje više ruta zajedno. Iscrtava samo prvi <*Route*> ili <*Redirect*> koji odgovara trenutnom URL-u. Ovo je korisno kada se želite osigurati prikazivanje samo jedne rute u isto vrijeme.
- <*Link*>: Koristi se za stvaranje poveznica unutar aplikacije. Iscrtava oznaku sidra (<*a*>) s navedenim URL-om, omogućujući korisnicima navigaciju između različitih pogleda bez ponovnog učitavanja cijele stranice.
- <*Redirect*>: Koristi se za preusmjeravanje na drugu rutu. Iscrtava novi URL i ažurira povijest preglednika, što ga čini korisnim za rješavanje slučajeva poput provjere autentičnosti ili neovlaštenog pristupa.

Korištenjem ovih komponenti definiraju se rute aplikacije i pridružene komponente. Također pruža dodatne značajke kao što su parametri rute, ugniježđene rute i programsku navigaciju, dajući potpunu kontrolu nad načinom na koji aplikacija rukuje rutama.

```
return (
    <div className="app">
      <BrowserRouter>
        <ThemeProvider theme={theme}>
          <CssBaseline />
          <Routes>
            <Route path="/" element={<LoginPage />} />
            <Route
              path="/home"
              element={isAuthorize ? <HomePage /> : <Navigate to="/"
/>}
            />
            <Route
              path="/profile/:userId"
              element={isAuthorize ? <ProfilePage /> : <Navigate
to="/" />}
            />
          </Routes>
        </ThemeProvider>
      </BrowserRouter>
    </div>
  );
}
```

Ispis 1: Implementacija React Routera

2.1.3 Redux Toolkit

Redux Toolkit je biblioteka čiji je cilj pojednostaviti rad s *Reduxom*, popularnom bibliotekom za upravljanje stanjem za *JavaScript* aplikacije, posebno one izrađene s *Reactom*. Omogućuje skup uslužnih programa i apstrakcija za pojednostavljenje razvojnog iskustva i smanjenje standardnog kôda pri korištenju *Reduxa* kao što je prikazano na ispisu 2 i ispisu 3.

Ključne značajke Redux Toolkita uključuju:

- *Redux Store* Konfiguracija: funkcija *configureStore()* postavlja *Redux Store* s zadanim postavkama, kombiniranjem reduktora, dodavanjem međuprograma (kao što je *Redux Thunk* za asinkrone akcije) i integracijom s *Redux DevTools Extensionom*.

- Pojednostavljena sintaksa reduktora: funkcija *createSlice()* pojednostavljuje stvaranje reduktora korištenjem objekta s reduktorskim funkcijama umjesto naredbi *switch*. To reduktore čini sažetijima i čitljivijima.
- Nepromjenjivost i *Immer* integracija: *Redux Toolkit* se integrira s *Immerom*, omogućujući intuitivnija ažuriranja stanja dopuštajući reduktorima da izravno mijenjaju stanje dok generiraju nepromjenjivu kopiju iza scene.
- Integracija *Redux DevTools Extension*: *Redux Toolkit* neprimjetno se integrira s *Redux DevTools Extensionom*, moćnim alatom za otklanjanje pogrešaka, što olakšava pregled i otklanjanje pogrešaka u stanju aplikacije.
- Rukovanje asinkronim radnjama: *createAsyncThunk() Redux Toolkita* pojednostavljuje rukovanje asinkronim radnjama slanjem višestrukih radnji tijekom različitih faza asinkrone operacije, kao što su čekanje, uspjeh i neuspjeh.
- Upravljanje stanjem entiteta: *CreateEntityAdapter() Redux Toolkita* pojednostavljuje upravljanje normaliziranim stanjem entiteta, pružajući pomoćne funkcije za uobičajene operacije na zbirkama entiteta.

Korištenjem *Redux Toolkita*, programeri mogu značajno smanjiti standardni kôd, slijediti najbolje prakse i uživati u intuitivnijem i učinkovitijem API-ju za rad s *Reduxom*. To rezultira preglednijim aplikacijama koje se lakše održavaju.

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
  mode: "light",
  user: {
    friends: [],
  },
  token: null,
  isAdmin: false,
  team: "",
  posts: [],
};
```

Ispis 2: Implementacija Redux početnog stanja

```
initialState,
  reducers: {
    setMode: (state) => {
      state.mode = state.mode === "light" ? "dark" :
"light";
    },
    setLogin: (state, action) => {
      state.user = action.payload.user;
      state.token = action.payload.token;
      state.isAdmin = action.payload.user.isAdmin;
    },
    setLogout: (state) => {
      state.user = null;
      state.token = null;
      state.isAdmin = false;
    },
    setFollowers: (state, action) => {
      if (state.user) {
        state.user.friends = action.payload.friends;
      } else {
        console.error("user friends non-existent :(");
      }
    },
    setPosts: (state, action) => {
      state.posts = action.payload.posts;
    },
    setPost: (state, action) => {
      const updatedPosts = state.posts.map((post) => {
        if (post._id === action.payload.post._id) return
action.payload.post;
        return post;
      });
      state.posts = updatedPosts;
    },
  },
```

Ispis 3: Implementacija Redux metoda stanja

2.1.4 Redux Persist

Redux Persist je biblioteka koja omogućuje održavanje stanja *Redux Storea*, osiguravajući da ono ostane netaknuto čak i kada korisnik osvježi stranicu ili ponovno pokrene aplikaciju. Omogućuje prikladan način za spremanje i učitavanje stanja Redux *Store-a* u mehanizam za pohranu, kao što je lokalna pohrana ili *AsyncStorage* u *React Native*. Integriranjem *Redux Persista* u *Redux* aplikaciju, može se neprimjetno održavati

korisnička sesija, sačuvati postavke ili omogućiti izvanmrežna funkcionalnost. Pojednostavljuje proces održavanja stanja, poboljšavajući cjelokupno korisničko iskustvo.

Ključne komponente i koncepti prikazani na ispisu 4 u *Redux Persistu* uključuju:

- *Perzistor*: *Perzistor* upravlja procesom postojanosti. Stvoren je omotavanjem *Redux Storea* s *Redux Persist* funkcijom *persistStore(). Persistor* upravlja sinkronizacijom između stanja *Redux Storea* i konfiguriranog mehanizma za pohranu.
- Pohrana: *Redux Persist* podržava različite mehanizme pohrane, uključujući lokalnu pohranu, pohranu sesije ili *AsyncStorage*. Može konfigurirati mehanizam za pohranjivanje pomoću kojeg se određuje gdje se stanje pohranjuje. Zadana postavka je *localStorage* za web aplikacije.
- Transformacije: *Redux Persist* omogućuje prilagođavanje serijalizacije i deserijalizacije podataka putem transformacija. Transformacije omogućuju izmjene podataka koji se zadržavaju, kao što je šifriranje osjetljivih informacija ili izostavljanje određenih polja.
- *Persist Gate*: Komponenta koja odgađa iscrtavanje aplikacije dok se trajno stanje ne dohvati i vrati. To osigurava da se aplikacija ne prikazuje s praznim stanjem dok se podaci učitavaju.

Korištenjem *Redux Persista* poboljšava se korisničko iskustvo održavajući stanje aplikacije, čineći je otpornom na ponovno učitavanje preglednika ili ponovno pokretanje aplikacije. Uklanja potrebu za ponavljajućom inicijalizacijom ili dohvaćanjem podataka, pružajući dosljedno i besprijekorno korisničko iskustvo. Međutim, važno je uzeti u obzir ograničenja pohrane, implikacije performansi i sigurnosna razmatranja kada se koristi *Redux Persist*.

```
const persistConfig ={ key: "root", storage, version: 1 };
const persistedReducer = persistReducer(persistConfig,
authReducer);
const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: {
```

```
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST,
PURGE, REGISTER],
      },
    }),
});
```

Ispis 4: Implementacija Redux Persista

2.1.5 Formik i yup

Formik i *Yup* dvije su popularne biblioteke u *JavaScript* ekosustavu koje se obično koriste zajedno za upravljanje obrascima i provjeru valjanosti obrazaca u *React* aplikacijama.

Formik pruža jednostavan i intuitivan API za rukovanje operacijama povezanim s obrascima i besprijekorno se integrira s *React* komponentama.

Pojednostavljuje razvoj obrazaca pružajući sljedeće značajke:

- Upravljanje stanjem obrasca: *Formik* upravlja stanjem unosa obrasca, vrijednostima, pogreškama i statusom podnošenja obrasca, smanjujući standardni kôd potreban za ručno rukovanje ovim aspektima.
- Provjera valjanosti obrazaca: *Formik* pruža jednostavan način provjere valjanosti unosa obrasca i prikaza poruka o pogrešci provjere valjanosti. Podržava i sinkronu i asinkronu provjeru, omogućujući definiciju pravila i uvjete provjere valjanosti za pojedinačna polja obrasca.
- Predaja obrasca: *Formik* usmjerava proces podnošenja obrasca pružajući rukovatelj *onSubmit* koji automatski obrađuje podnošenje obrasca, uključujući provjeru valjanosti, serijalizaciju podataka obrasca i integraciju API-ja.
- Upravljanje poljima: *Formik* pojednostavljuje upravljanje poljima obrasca pružanjem komponenti kao što su <*Field*> i <*FastField*>. Ove komponente obrađuju različite aspekte kao što su rukovanje ulaznim vrijednostima, *onChange* događajima i prikazivanjem poruka o greškama.
- *Form Context*: *Formik* koristi *Reactov Context* API za pružanje centraliziranog rješenja za upravljanje stanjem. Osigurava da su funkcije

stanja obrasca, provjere valjanosti i podnošenja lako dostupne u različitim komponentama obrasca.

S druge strane, *Yup* je *JavaScript* biblioteka koja se koristi za provjeru valjanosti obrazaca. Pruža jednostavan i izražajan API za definiranje shema provjere valjanosti i izvođenje provjera valjanosti podataka obrazaca.

Yup nudi sljedeće značajke:

- Definicija sheme: Omogućuje definiranje sheme provjere valjanosti pomoću API-ja koji se može ulančati. Određuju se pravila kao što su obavezna polja, vrste podataka, minimalne i maksimalne duljine, regularni izrazi i više.
- Validacija: Nakon što je shema provjere valjanosti definirana, *Yup* nudi metode za provjeru valjanosti ulaznih podataka u odnosu na shemu. Izvodi provjeru sinkrono i asinkrono, što omogućuje jednostavno rukovanje složenim scenarijima.
- Poruke o pogreškama: Omogućuje definiranje prilagođenih poruka o pogreškama za svako pravilo provjere valjanosti, pružajući fleksibilnost u prikazivanju poruka o pogrešci prilagođenih korisniku na temelju rezultata provjere valjanosti.
- Validacija oblika objekta: Osim provjere oblika, *Yup* može potvrditi oblik složenih objekata. Ovo je osobito korisno pri radu s ugniježđenim podatkovnim strukturama.

Formik i *Yup* nadopunjuju se besprijekornom integracijom. *Formik* omogućuje upravljanje stanjem obrasca i mogućnosti podnošenja, dok *Yup* upravlja provjerom valjanosti unosa obrazaca. Ova kombinacija pojednostavljuje proces izgradnje robusnih i validiranih obrazaca u *React* aplikacijama.

2.2 Backend

Izraz *backend* odnosi se na poslužiteljsku stranu web aplikacije ili softverskog sustava. Obuhvaća komponente, procese i logiku koji djeluju iza kulisa za rukovanje pohranom i obradom podataka te komunikacijom s klijentima.

Pozadina se obično sastoji od sljedećih ključnih elemenata:

- Poslužitelj: Može biti računalo ili mreža računala. Pokreće ga softver za rukovanje zahtjevima klijenata koji daje odgovore u obliku podataka. Upravlja komunikacijom između klijenta i pozadinskih komponenti.
- Logika aplikacije: Uključuje kôd i algoritme koji implementiraju poslovnu logiku aplikacije. Obrađuje zahtjeve, izvodi izračune, primjenjuje poslovna pravila i komunicira s bazama podataka i drugim uslugama kako bi generirao željeni rezultat.
- Baza podataka: Pozadinski sustavi se oslanjaju na bazu podataka za pohranjivanje i dohvaćanje podataka. Baza podataka pruža strukturiran način organiziranja i upravljanja velikim količinama informacija. Uobičajene vrste baza podataka koje se koriste u *backend* razvoju uključuju relacijske baze podataka (kao što su *MySQL* i *PostgreSQL*) i *NoSQL* baze podataka (kao što su *MongoDB* i *Redis*).
- API (*Application Programming Interfaces*): Omogućuje komunikaciju i razmjenu podataka između različitih softverskih sustava. Razvoj pozadine uključuje stvaranje API-ja koji izlažu krajnje točke putem kojih klijenti mogu slati zahtjeve i primati odgovore. API-ji definiraju protokole, formate podataka (npr. JSON, XML) i mehanizme provjere autentičnosti za interakciju s pozadinom.
- Sigurnost: *Backend* programeri odgovorni su za provedbu sigurnosnih mjera za zaštitu aplikacije i njenih podataka. To uključuje mehanizme provjere autentičnosti korisnika i autorizacije, enkripciju podataka, provjeru valjanosti unosa i zaštitu od uobičajenih ranjivosti kao što su *cross-site scripting* (XSS) i SQL *injection*.
- Predmemorija i optimizacija izvedbe: *Backend* programeri koriste različite tehnike za poboljšanje performansi i odziva aplikacije. Mehanizmi predmemorije, kao što je korištenje predmemorije u radnoj memoriji ili mreža za isporuku sadržaja (CDN), mogu smanjiti opterećenje baze podataka i poboljšati vrijeme odziva. Pozadinski programeri također optimiziraju kôd, upite baze podataka i mrežnu komunikaciju kako bi poboljšali učinkovitost.

- Integracija s uslugama trećih strana: Mnoge se aplikacije oslanjaju na vanjske usluge za dodatnu funkcionalnost. Pozadinski programeri integriraju te usluge u aplikaciju interakcijom sa svojim API-jima, bilo da se radi o pristupnicima plaćanja, uslugama e-pošte, uslugama geolokacije ili platformama društvenih medija.

Razvoj pozadine uključuje korištenje programskih jezika, okvira i alata prikladnih za zadatke na strani poslužitelja. Često korištene pozadinske tehnologije uključuju Node.js (*JavaScript runtime*), *Python* (s okvirima kao što su *Django* ili *Flask*), *Ruby* (s *Ruby on Rails), Java* (s okvirima kao što je *Spring Boot*) i PHP (s okvirima kao što su *Laravel ili Symfony*) .

Ukratko, *backend* upravlja operacijama aplikacije na strani poslužitelja, uključujući obradu zahtjeva, upravljanje podacima, osiguranje sigurnosti i integraciju s drugim uslugama. Služi kao temelj koji pokreće funkcionalnost i mogućnosti obrade podataka web ili softverske aplikacije.

2.2.1 Node.js

Node.js moćno je i svestrano okruženje za izvršavanje *JavaScripta* koje programerima omogućuje izradu skalabilnih aplikacija na strani poslužitelja. Iskorištava asinkronu arhitekturu vođenu događajima, što ga čini vrlo učinkovitim za rukovanje istodobnim vezama i obradu velikih količina zahtjeva. Sa svojim opsežnim ekosustavom i upraviteljem paketa (NPM), programeri imaju pristup širokom nizu biblioteka i modula za ubrzavanje razvoja. *Node.js* omogućuje ponovnu upotrebu kôda između *frontenda* i *backenda*, što ga čini idealnim za izradu *end-to-end JavaScript* aplikacija. Njegova kompatibilnost s više platformi, aktivna zajednica i sposobnost rukovanja aplikacijama i mikroservisima u stvarnom vremenu pridonijeli su njegovoj popularnosti među programerima diljem svijeta. *Node.js* mijenja igru za razvoj *JavaScripta* na strani poslužitelja.

2.2.2 Express.js

Express.js brz je, minimalistički i fleksibilan okvir web aplikacije za *Node.js*. Pojednostavljuje proces izgradnje robusnih web poslužitelja i API-ja pružanjem skupa

intuitivnih metoda i međuprograma. S fokusom na jednostavnost i modularnost, *Express.js* omogućuje programerima stvaranje skalabilnih i prilagodljivih aplikacija. Upravlja usmjeravanjem i rukovanjem zahtjeva/odgovora te upravlja posrednim softverom, što ga čini idealnim izborom za izgradnju *RESTful* API-ja i web aplikacija. *Express.js* nudi bogat ekosustav s brojnim proširenjima i međuprogramskim paketima, omogućujući programerima jednostavno dodavanje funkcionalnosti. Stekao je popularnost zbog svoje jednostavnosti i lakoće korištenja, što ga čini uobičajenim okvirom za web razvoj *Node.js-a*.

2.2.3 Mongoose

Mongoose je biblioteka za modeliranje objektnih podataka (ODM) za *Node.js* i *MongoDB*. Omogućuje izravan način interakcije s *MongoDB* bazama podataka, kao što je prikazani na ispisu 5, omogućujući razvojnim programerima definiranje shema podataka, izvođenje CRUD operacija i provođenje provjere valjanosti podataka. *Mongoose* pojednostavljuje rad s *MongoDB-om* pružajući apstrakciju više razine, upravljajući vezama i nudeći značajke kao što su provjera valjanosti temeljena na shemi, srednji softver i izgradnja upita. Besprijekorno se integrira s *Node.js-om* i pruža moćne uslužne programe za rukovanje složenim odnosima podataka, indeksiranje i postavljanje upita. *Mongoose* se naširoko koristi u *Node.js* aplikacijama kako bi se pojednostavile *MongoDB* interakcije i učinile operacije baze podataka intuitivnijim i učinkovitijim.

```
const PORT = process.env.PORT || 5001;
mongoose
  .connect(process.env.MONGO_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => {
    app.listen(PORT, () => console.log(`Server Port:
${PORT}`));
  })
  .catch((error) => console.log(`${error} did not
connect`));
```

Ispis 5: Povezivanje s bazom podataka korištenjem Mongoosea

2.2.4 JSON Web Token (JWT)

JWT ili *JSON web token*, kompaktan je i samostalan način za siguran prijenos informacija o autentifikaciji i autorizaciji između strana. Sastoji se od tri dijela: zaglavlja, korisnog sadržaja i potpisa. Zaglavlje sadrži vrstu tokena i algoritam koji se koristi za provjeru potpisa. Token sadrži zahtjeve kao što su identitet korisnika i vrijeme isteka. Potpis se stvara kombinacijom zaglavlja, nosivosti i tajnog ključa, čime se osigurava integritet i autentičnost tokena. JWT tokeni se obično koriste za autentifikaciju bez stanja u web aplikacijama, omogućujući poslužitelju da provjeri informacije o tokenu bez potrebe za sesijama na strani poslužitelja. Ispis 6 i ispis 7 prikazuju implementaciju JWT tokena kroz kôd dok ispis 8 prikazuje verifikaciju tog tokena.

```
const salt = await bcrypt.genSalt();
const passwordHash = await bcrypt.hash(password, salt);
```

Ispis 6: JTW salt i hash metode

```
const token = jwt.sign({ id: user._id },
process.env.JWT_SECRET);
const userWithoutPassword = { ...user.toObject() };
delete userWithoutPassword.password;
```

Ispis 7: JWT sign metoda

```
import jwt from "jsonwebtoken";

export const verifyJwtToken = async (req, res, next) => {
  try {
    let token = req.header("Authorization");

    if (!token) {
      return res.status(403).send("Access to account
denied");
    }

    if (token.startsWith("Bearer ")) {
      token = token.slice(7, token.length).trimLeft();
    }
```

```
    const verified = jwt.verify(token,
process.env.JWT_SECRET);
    req.user = verified;
    next();
  } catch (err) {
  }
};
```

Ispis 8: Provjera ispravnosti JWT tokena

2.2.5 Multer

Multer je međuprogram za rukovanje *multipart/form* podacima. Prvenstveno se koristi za učitavanje datoteka u *Node.js* aplikacijama. Pojednostavljuje proces učitavanja datoteka pružanjem praktičnog API-ja i dobrom integracijom s *Express.js-om*. *Multer* programerima omogućuje odabir odredišne mape, pravila imenovanja datoteka i ograničenja veličine datoteka. Automatski analizira i sprema učitane datoteke na određeno mjesto. *Multer* podržava učitavanje jedne datoteke, kao i više datoteka u jednom zahtjevu. Također pruža provjeru vrste datoteke i druge prilagodljive mogućnosti. Programeri mogu učinkovito rukovati učitavanjem datoteka u svojim aplikacijama, što ga čini popularnim izborom.

2.2.6 MongoDB

MongoDB je fleksibilna i skalabilna *NoSQL* baza podataka koja pohranjuje podatke u dokumente slične JSON-u. Omogućuje dizajn bez shema, što pruža dinamično i agilno modeliranje podataka. Uz značajke poput automatskog dijeljenja i replikacije, *MongoDB* omogućuje horizontalnu skalabilnost i visoku dostupnost. Podržava moćan upitni jezik s opsežnim mogućnostima indeksiranja, što pruža učinkovito pronalaženje podataka što je i prikazano na ispisu 9 i ispisu 10. *MongoDB* se široko koristi u raznim aplikacijama, uključujući web i mobilni razvoj, sustave za upravljanje sadržajem i analitiku u stvarnom vremenu. Dobro se integrira s popularnim programskim jezicima i okvirima, što olakšava rad s njim. Sveukupno, *MongoDB* nudi programerima skalabilno, visokoučinkovito i jednostavno rješenje baze podataka za rukovanje raznolikim i evoluirajućim zahtjevima podataka.

```
_id: ObjectId('64e5d95c40eb7a13f55a5984')
userId: "64e5d90d40eb7a13f55a5964"
firstName: "Ena"
lastName: "Enic"
location: "Split"
description: "Code for today"
picturePath: "code.jpg"
userPicturePath: "matheus-ferrero-W7b3eDUb_2I-unsplash.jpg"
▾ likes: Object
    64e5d4e140eb7a13f55a5802: true
    64e606976c13b6a25a08740f: true
▾ comments: Array
  ▾ 0: Object
      firstName: "Ana"
      lastName: "Antic"
      comment: "Code review please :)"
      _id: ObjectId('64e5edb76c13b6a25a087348')
  ▾ 1: Object
      firstName: "Luka"
      lastName: "Vukovic"
      comment: "Nelose!"
      _id: ObjectId('64e606b86c13b6a25a08742a')
  team: "Team Alpha"
  createdAt: 2023-08-23T10:03:08.348+00:00
  updatedAt: 2023-08-23T13:16:40.257+00:00
  __v: 2
```


Ispis 9: Podatak objave u bazi podataka


```
_id: ObjectId('64e5d90d40eb7a13f55a5964')
firstName: "Ena"
lastName: "Enic"
email: "ena.enic@gmail.com"
password: "$2b$10$pbSeYC./yjtPSnwRXBzMp.5Pyh1m9ZH3i50jegUQ00yhXqnT8.Os6"
picturePath: "matheus-ferrero-W7b3eDUb_2I-unsplash.jpg"
▾ friends: Array
    0: "64e473c42c20cc6fa1074f4d"
    1: "64e5d4e140eb7a13f55a5802"
location: "Split"
occupation: "Software Developer"
impressions: 0
twitterLink: ""
linkedinLink: ""
isAdmin: false
team: "Team Alpha"
createdAt: 2023-08-23T10:01:49.990+00:00
updatedAt: 2023-08-23T10:03:22.779+00:00
__v: 4
```

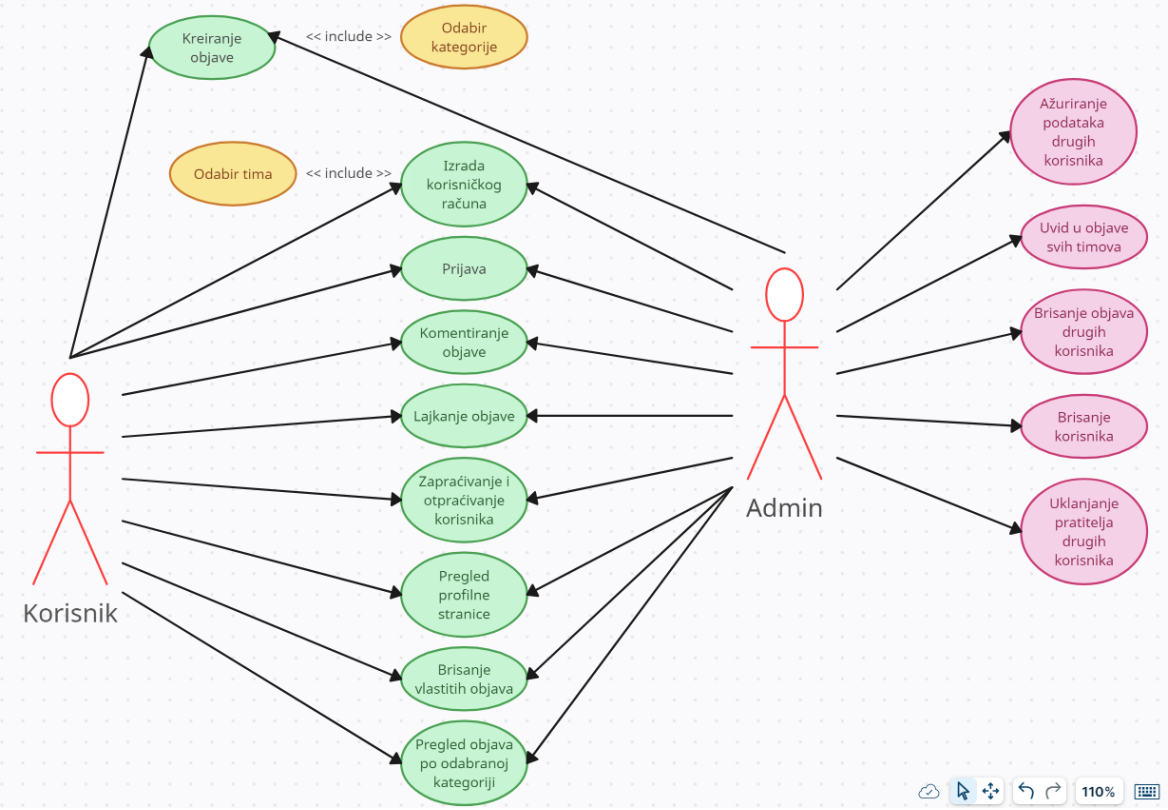

Ispis 10: Podatak korisnika u bazi podataka

2.3 UML Dijagrami

UML dijagrami su vizualni prikazi koji se koriste u softverskom inženjerstvu za opisivanje, vizualizaciju i komuniciranje različitih aspekata dizajna i ponašanja sustava. UML dijagrami nude standardizirani način za prikazivanje koncepata kao što su klase, objekti, odnosi, interakcije i više. Oni pomažu programerima i dizajnerima da razumiju arhitekturu i funkcionalnost softverskog sustava.

2.3.1 Use case dijagram

Na slici 1 prikazan je *use case* dijagram web aplikacije *Techbook* koji prikazuje kompletno funkcioniranje cijelog sustava.

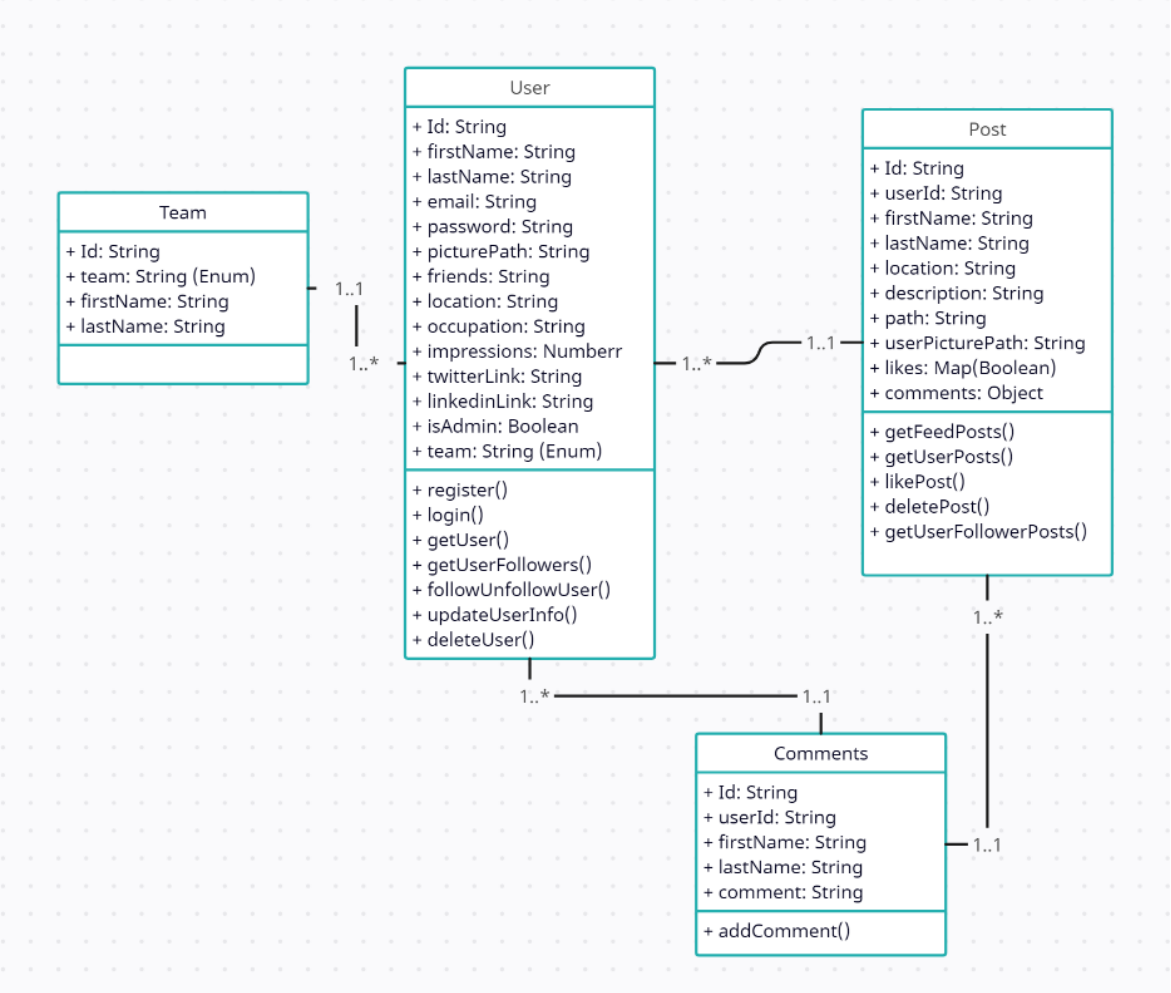


Slika 1: Use case dijagram sustava

2.3.2 Dijagram klasa

Na slici 2 prikazan je dijagram klasa sustava web aplikacije *Techbook*. Prikazane su četiri klase. Klasa *User* se sastoji od podataka o korisniku dok se klasa *Post* sastoji od

podataka o objavi. Osim podataka tu su i imena metoda koje su definirane i koje koriste te podatke na određene načine. Svaki korisnik može imat jedan tim, a u timu može biti više korisnika. Svaki korisnik može napisati više komentara, ali jedan komentar može imati samo jednog kreatora. Svaka objava također može imati više komentara, ali svaki komentar može biti na samo jednoj objavi.



Slika 2: Dijagram klasa sustava

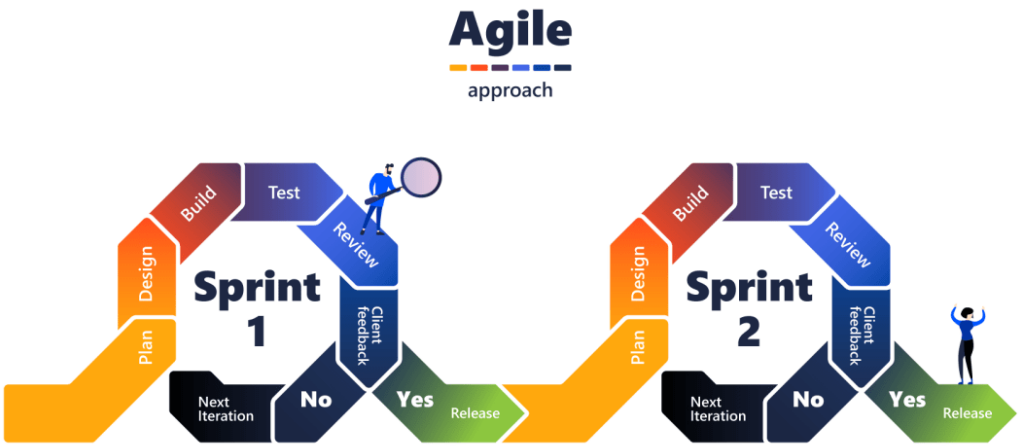
2.4 Agilni način rada

Web aplikacija *Techbook* napravljena je koristeći agilni način rada. Agilni način rada u programiranju odnosi se na fleksibilan i interaktivni pristup razvoju softvera koji naglašava suradnju, prilagodljivost i usmjerenost na korisnika. Uključuje rastavljanje procesa razvoja u manje cikluse koji se nazivaju "sprintovi" ili "iteracije", koji obično traju jedan do četiri tjedna. Tijekom svakog sprinta, kao što je prikazano na slici 3, međufunkcionalni tim

programera, testera, dizajnera i drugih dionika surađuje kako bi isporučio radni inkrement softvera. Agilno poslovanje njeguje otvorenu komunikaciju i jednakost među zaposlenima, bez tradicionalne hijerarhije. Temelji se na timskom radu i dobroj povezanosti, a potiče učenje kroz iskustvo, samoinicijativnost i osobni napredak. Velika je pozornost posvećena međusobnim odnosima i dobroj suradnji, podijeli znanja i ohrabrivanju.

Agilna kultura temelji se na 4 osnovne vrijednosti:

- pojedinci i njihove interakcije su uvijek važniji od procesa i alata
- proizvod koji radi je važniji od njegove dokumentacije
- odnos s klijentom i dobar dogovor je važniji od potpisivanja ugovora
- reagiranje na promjenu je bitnije od tog da se kruto slijedi plan



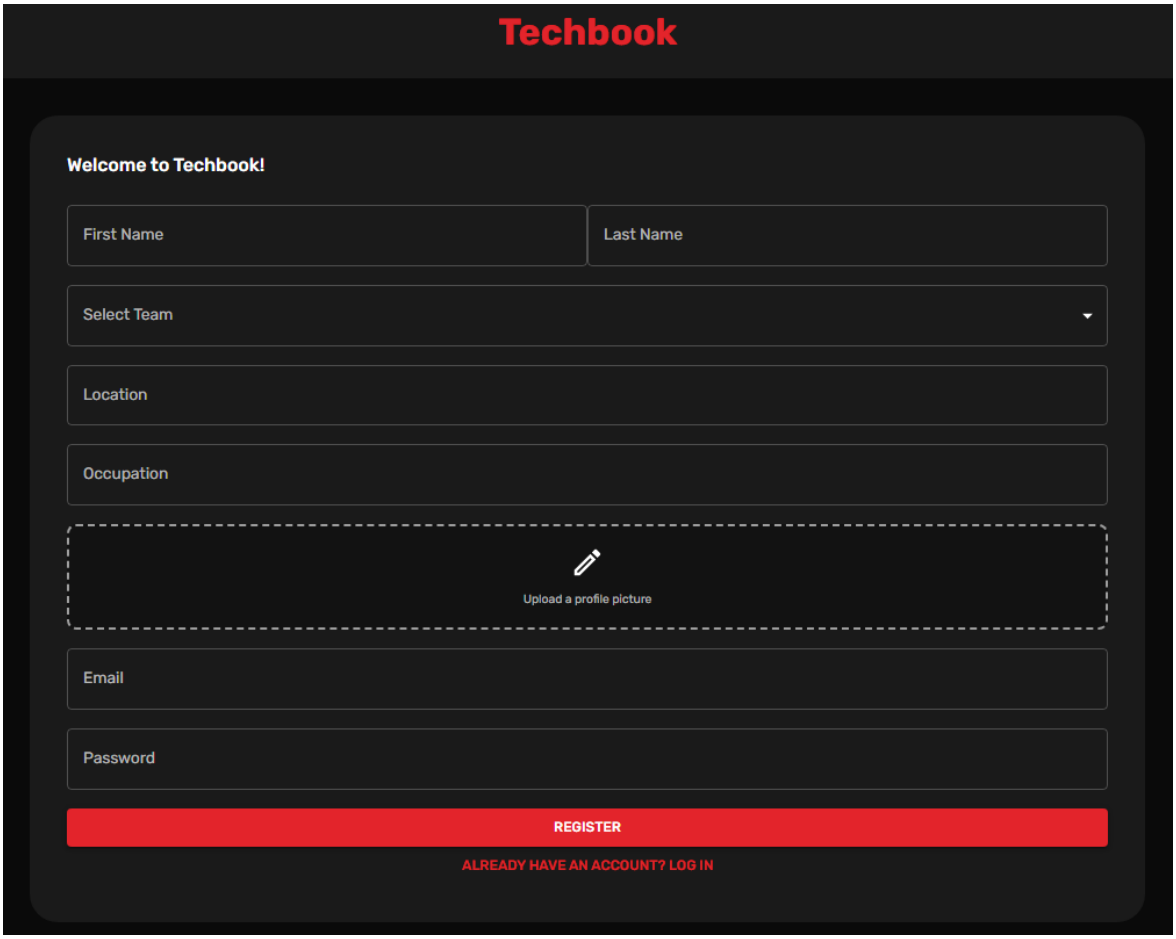
Slika 3: Graf agilnog načina rada

3 DETALJAN OPIS WEB APLIKACIJE

U ovom poglavlju predstavljen je detaljan opis web aplikacije te su istaknute osnovne komponente odnosno dijelovi aplikacije. Svaka od komponenti ima svoju svrhu i način primjene.

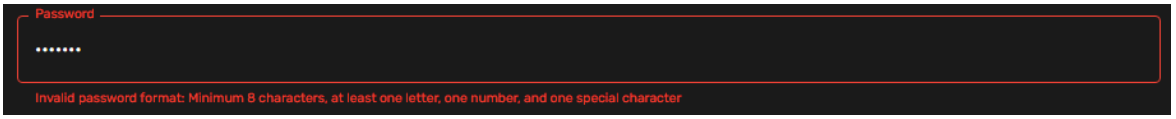
3.1 Stranica za registraciju

Svaki korisnik mora napraviti korisnički račun kako bi koristio aplikaciju. Potrebno je unijeti informacije kao što su ime i prezime korisnika, tim, lokacija i zaposlenje, email adresa i lozinka te na samom kraju profilnu sliku kao što je i prikazano na slici 4. Nakon toga klikom na „*Register*“ dugme kreira se korisnički račun ako su svi podaci uneseni po postavljenim kriterijima. Za ime i prezime korisnik ima ograničenje na dužinu od minimalno 2 slova, a maksimalno 50 slova. Email adresa mora sadržavati maksimalno 50 slova, mora biti unikatna, te imati konstrukciju email adrese kao što je i prikazano na slici 5. Lozinka mora imati minimalno 8 znakova što i vidimo na slici 6. Ona se prije upisa u bazu podataka hešira radi sigurnosti korisnika. Lokacija i zaposlenje su ograničeni na maksimalno 100 slova. Za profilnu sliku podržavaju se svi široko korišteni formati (.jpg, .jpeg, .png itd.). Sve ove informacije su obavezne za unos.



Slika 4: Stranica za registraciju

Slika 5: Poruka prilikom upisa netočnog formata email adrese

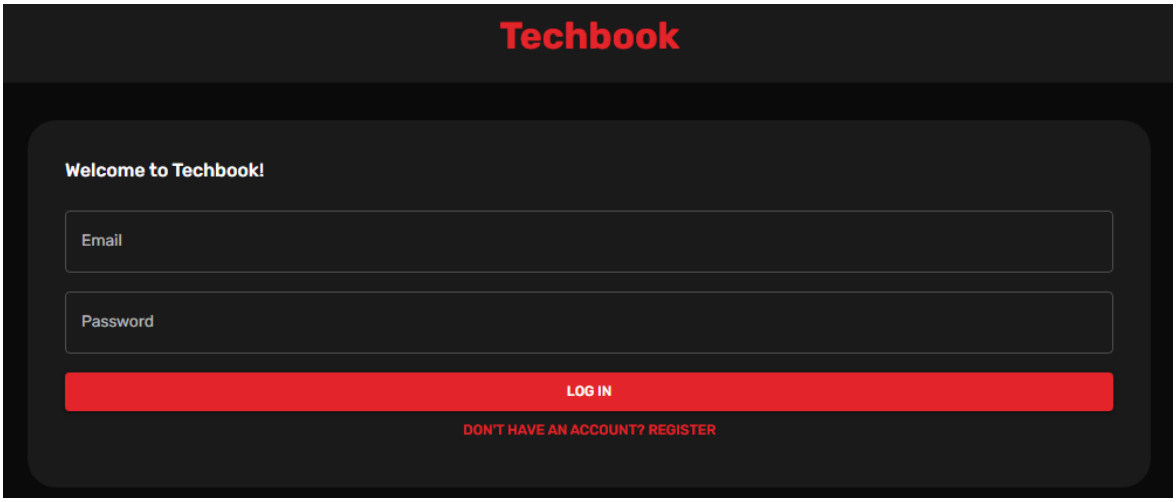


Slika 6: Poruka prilikom upisa netočnog formata lozinke

3.2 Stranica za prijavu

Nakon što korisnik napravi korisnički račun potrebno se prijaviti na stranicu. Korisnik mora unijeti svoju email adresu i lozinku što je i prikazano na slici 7. Ako korisnik unese krive informacije prikaže se poruka koja obavještava korisnika da je unesen kriva

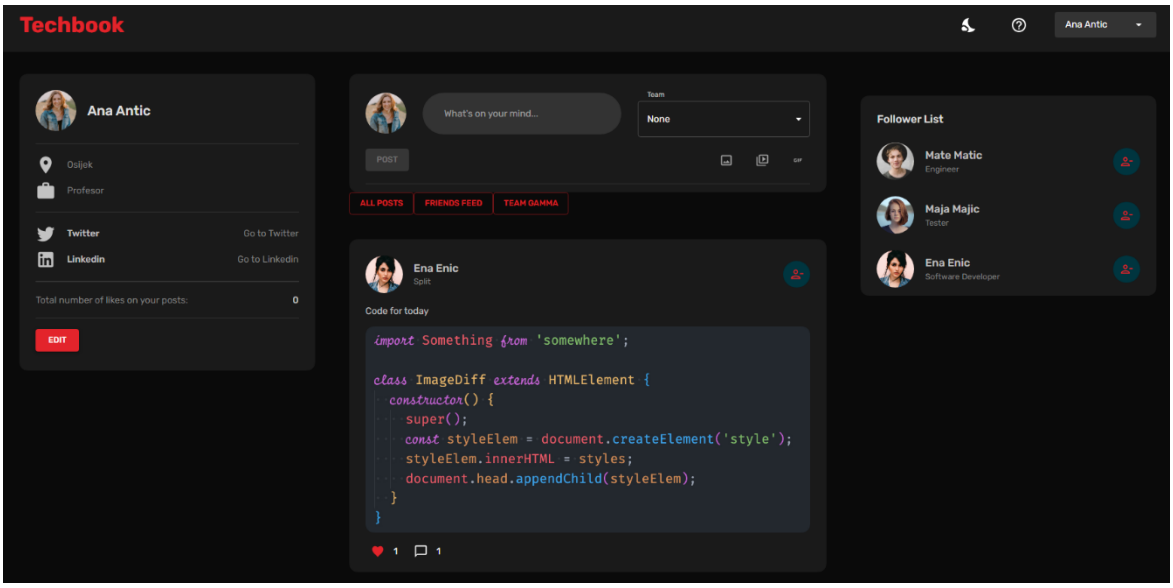
email adresa ili lozinka. Nakon uspješne autentifikacije, korisnika se pozicionira na naslovnu stranicu aplikacije.



Slika 7: Stranica za prijavu

3.3 Naslovna stranica

Naslovna stranica je prva stranica koju korisnik vidi kada obavi prijavu, tj. autentifikaciju. Stranica se sastoji od 5 komponenti (slika 8). Navigacijska traka, korisnička kartica, lista pratitelja, kartica objave i kartica za kreiranje objave.



Slika 8: Naslovna stranica

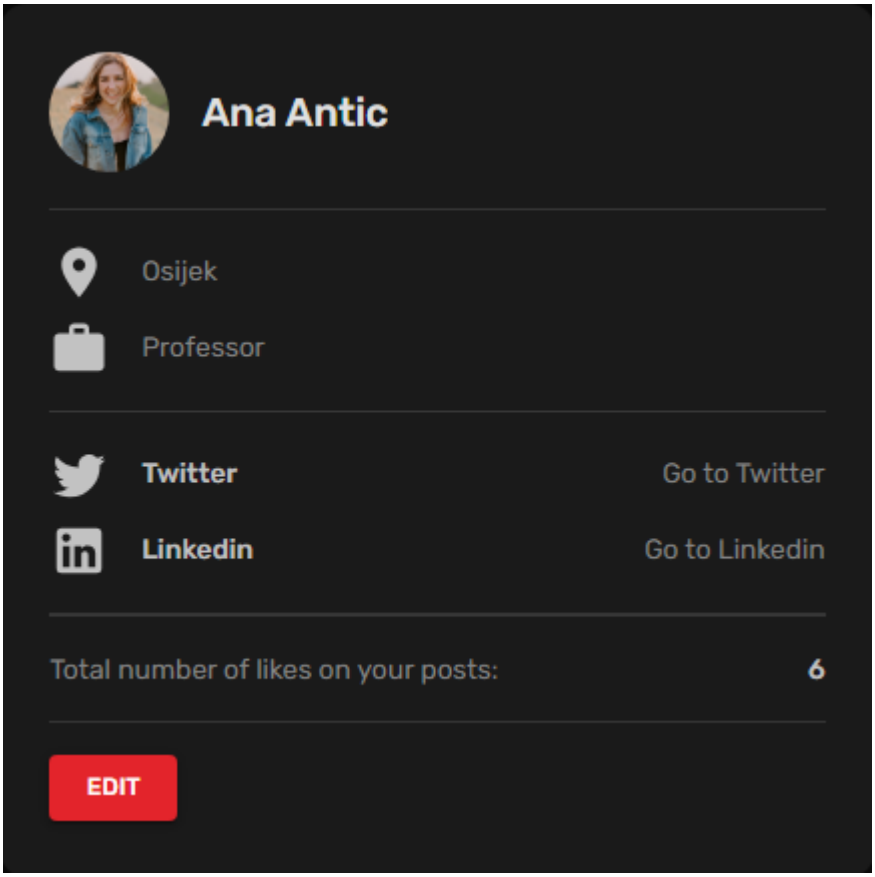
3.4 Navigacijska traka

Navigacijska traka je navigacijska komponenta koja se sastoji od više funkcija (slika 9). Klikabilni tekst „*Techbook*“ služi za osvježavanje stranice. Klikom na ikonu „sunce/mjesec“ korisnik prebacuje temu cijele web aplikacije iz svijetle u tamnu i obrnuto. Kada korisnik približi miš ikoni „?“, pojavljuje tekst koji objašnjava osnovne funkcionalnosti ove web aplikacije. Klikom na svoje ime i prezime otvara se padajuća lista u kojoj korisnik ima opciju „*Logout*“ kojom izlazi iz stranice i vraća se na stranicu za prijavu.

Slika 9: Navigacijska traka

3.5 Korisnička kartica

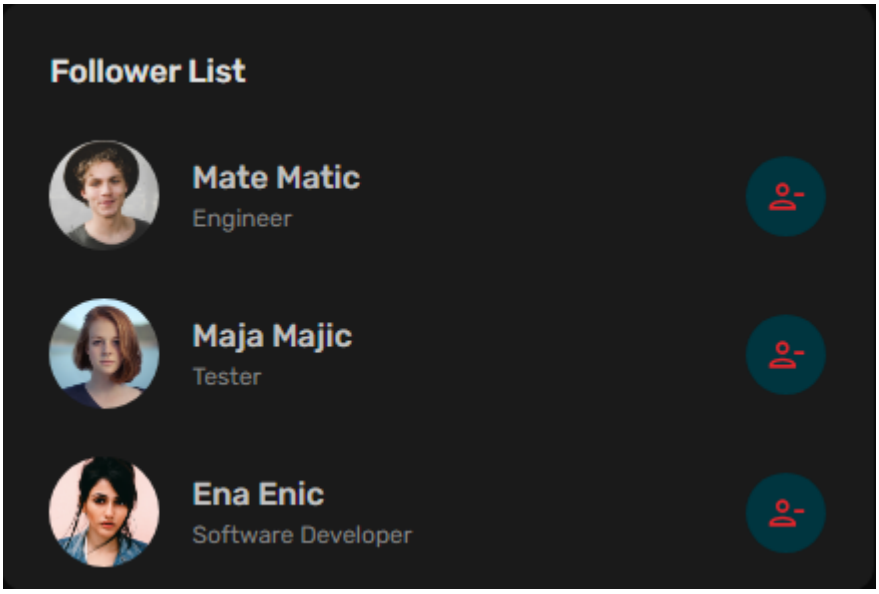
Korisnička kartica je komponenta koje se sastoji od imena i prezimena korisnika, njegove profilne slike, lokacije, trenutnog zanimanja, ukupnog broja lajkova na objavama, poveznice za *Twitter* i *Linkedin* račun te *Edit* dugmeta što je i prikazano na slici 10. Klikom na „*Edit*“ dugme korisnik ima mogućnost ažuriranja lokacije, trenutnog zanimanja i poveznice za *Twitter* i *Linkedin*. Svaki put kada netko lajka njegovu objavu korisniku se „*Total number of likes on your posts*“ poveća se za 1.



Slika 10: Korisnička kartica

3.6 Lista pratitelja

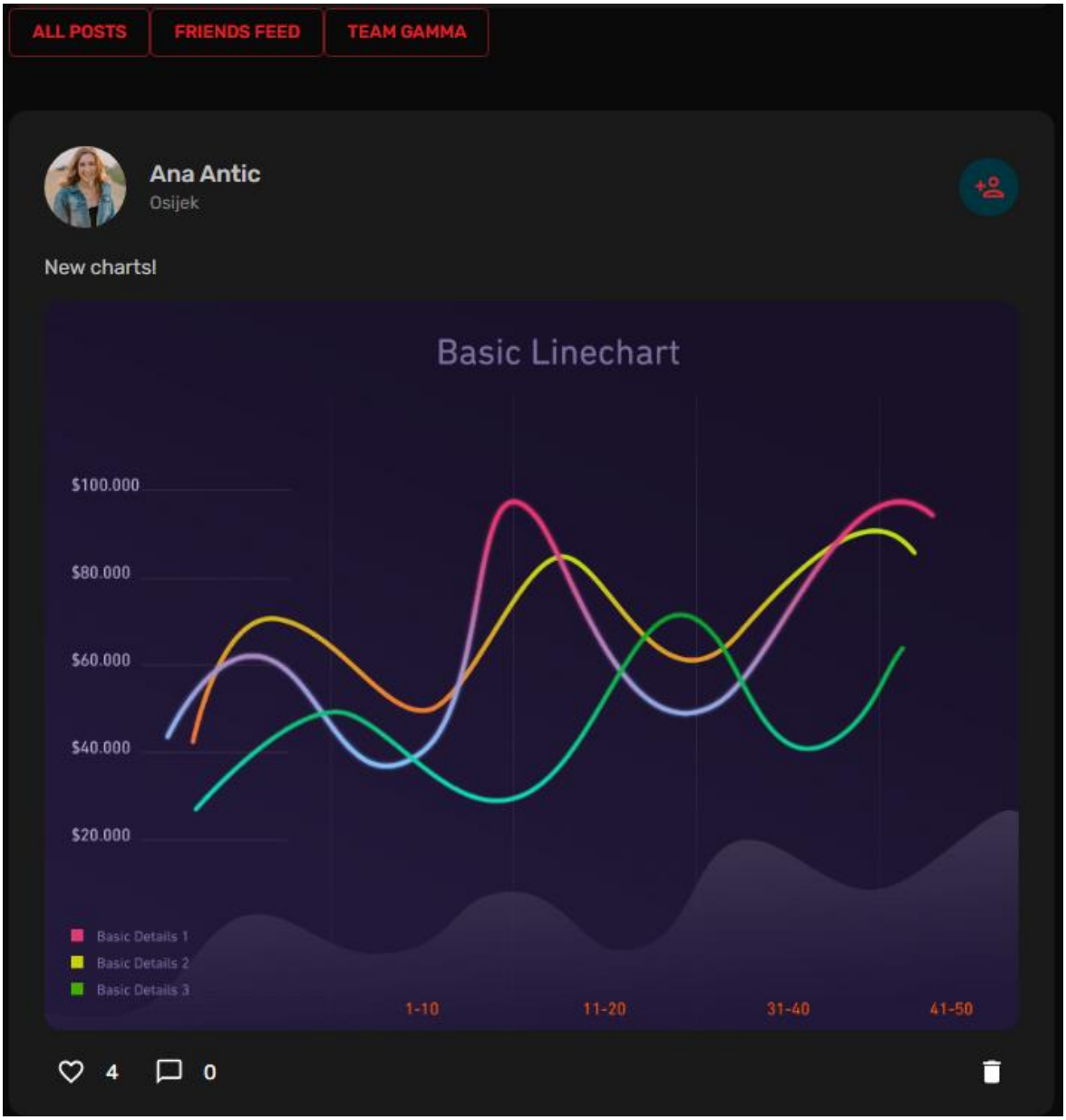
Korisnik u listi pratitelja (slika 11) vidi druge korisnike koje prati. Klikom na ikonu označenu sa „–„ može otpratiti korisnika kojeg odabere. Klikom na ime jednog od korisnika kojeg prati ulazi u njegov korisnički profil.



Slika 11: Lista pratitelja

3.7 Kartica objava

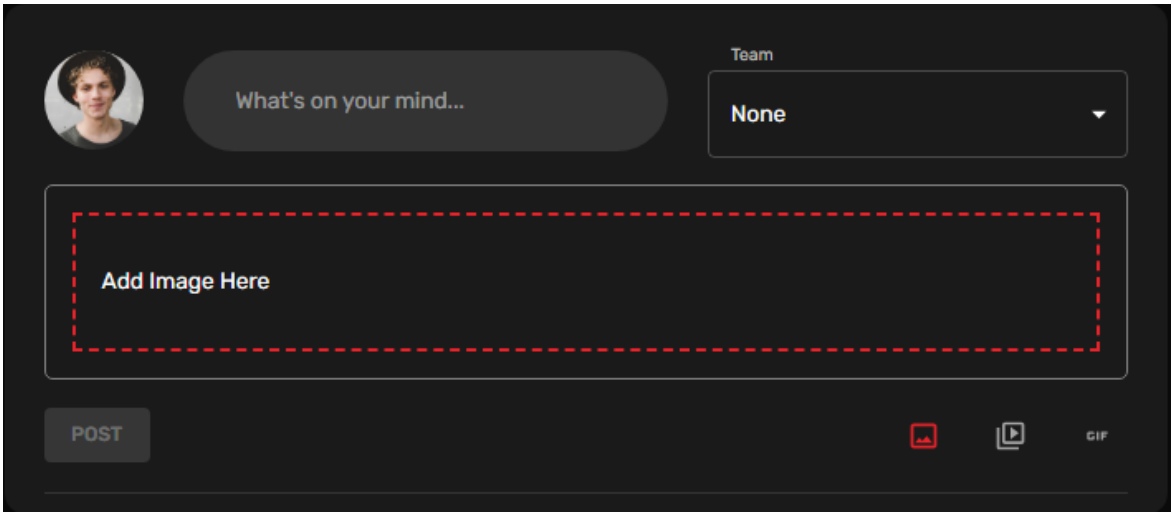
Kartica objava je komponenta koja se sastoji od svih objava koje su napravili korisnici web aplikacije. Objave mogu biti u obliku fotografije, GIF-a ili video formata. Svaku objavu može lajkati i komentirati bilo koji korisnik te zapratiti tog korisnika ako mu se sviđa njegov sadržaj. Također se preko objave može ući na profil korisnika koji je napravio tu objavu. Objave su podijeljene u tri kategorije što je i prikazano na slici 12. *All posts* prikazuju objave svih korisnika koje nisu objavljene unutar grupe tima. *Friends feed* prikazuje samo objave korisnika koje je korisnik zapratio, a koje također nisu objave unutar grupe tima. Pod *Team name* nalaze se isključivo objave iz timske grupe i njih mogu vidjeti samo članovi tog tima.



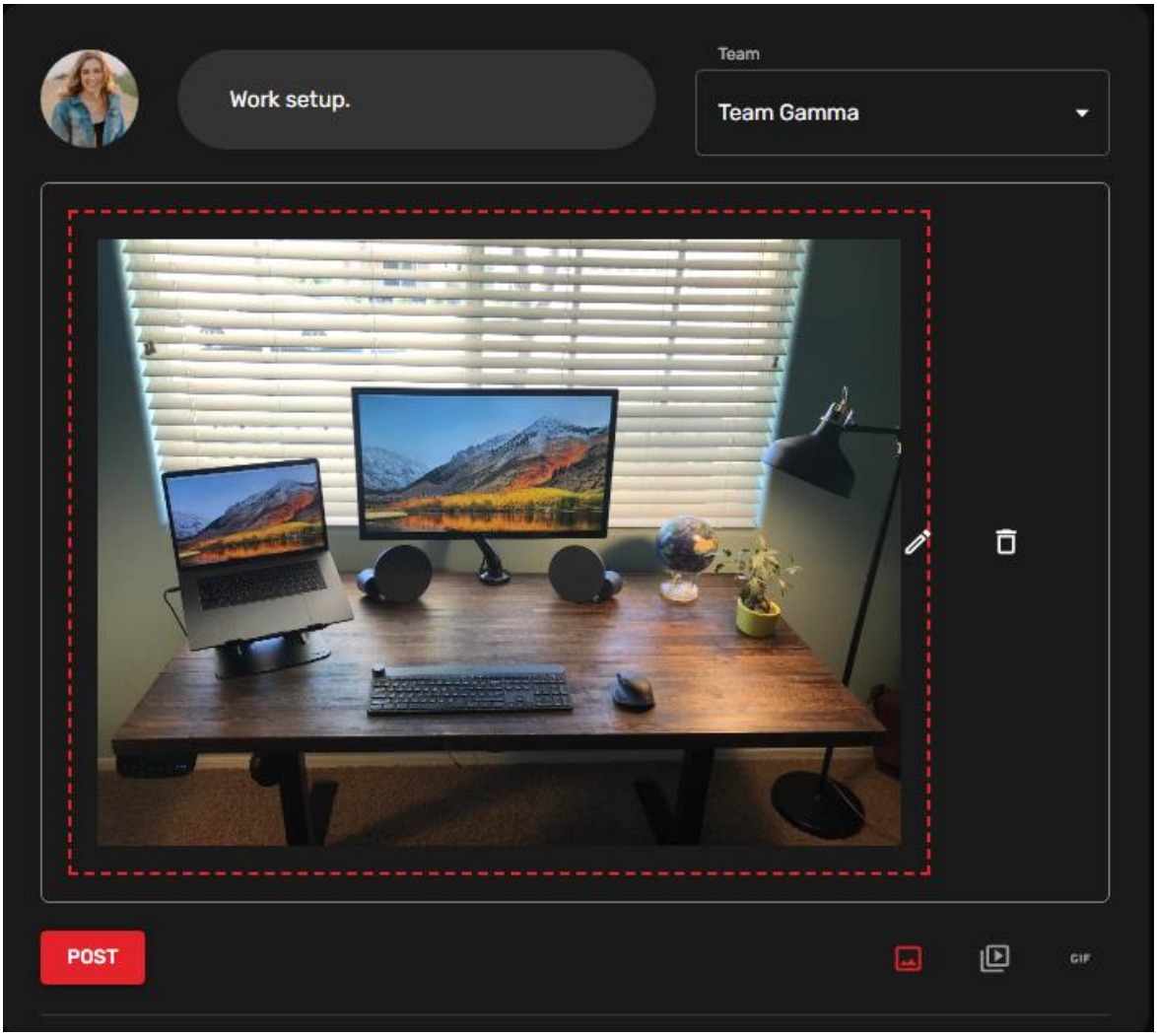
Slika 12: Kartica objava

3.8 Kartica za kreiranje objave

Kartica za kreiranje objave (slika 13 i slika 14) je komponenta koja se sastoji od trake za upis opisa objave, ikona za dodavanje fotografije, videozapisa ili GIF-a te dugme „*Post*“ kojim korisnik kreira novu objavu. Nakon klika na pojedinu ikonu otvara se prozor pomoću koje korisnik bira video, GIF ili fotografiju koju želi objaviti.



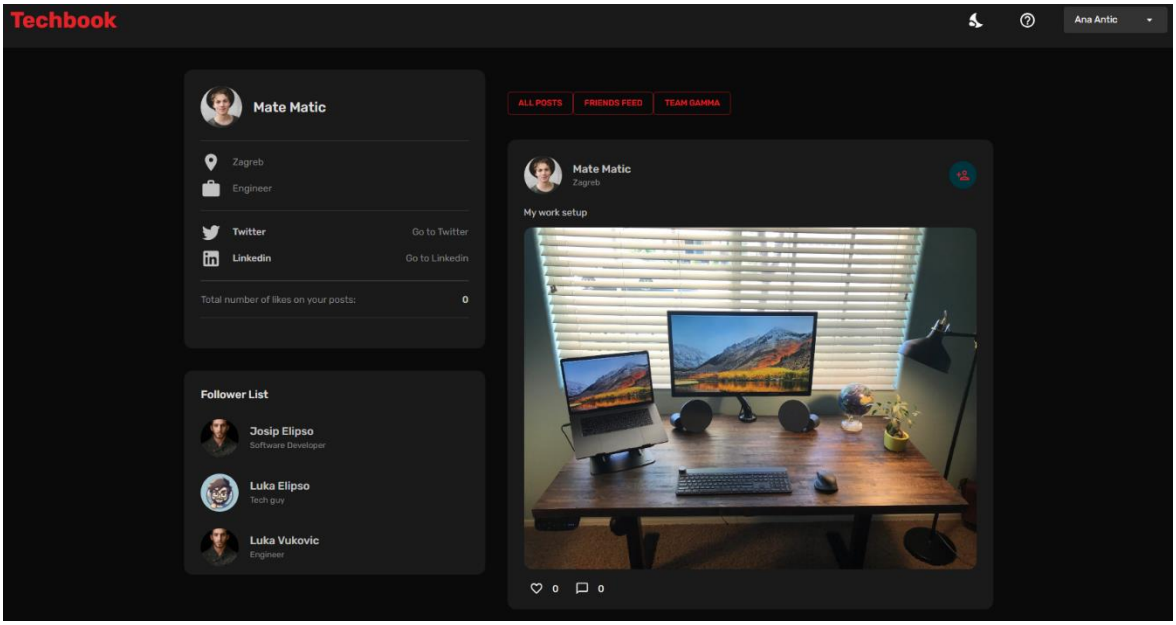
Slika 13: Kartica za kreiranje objave



Slika 14: Kartica za kreiranje objave nakon odabira slike i tima

3.9 Profilna stranica

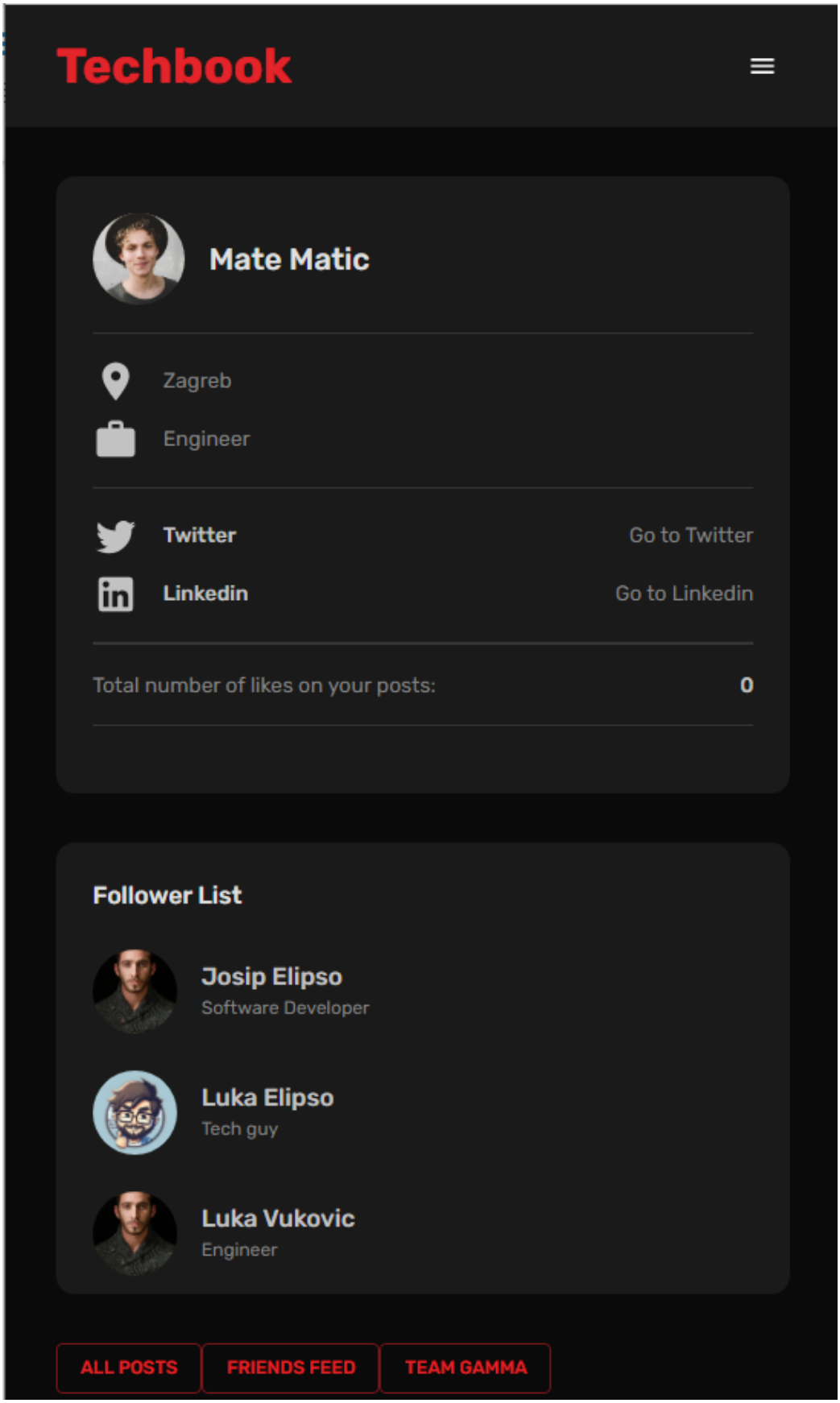
Ulaskom na profilnu stranicu korisnika (slika 15) imamo pristup svim njegovim javno dostupnim informacijama. Vidimo sve njegove objave koje nisu unutar tima, listu pratitelja i njegovu karticu korisnika preko koje možemo pristupiti korisnikovom *Twitteru* i *LinkedInu*.



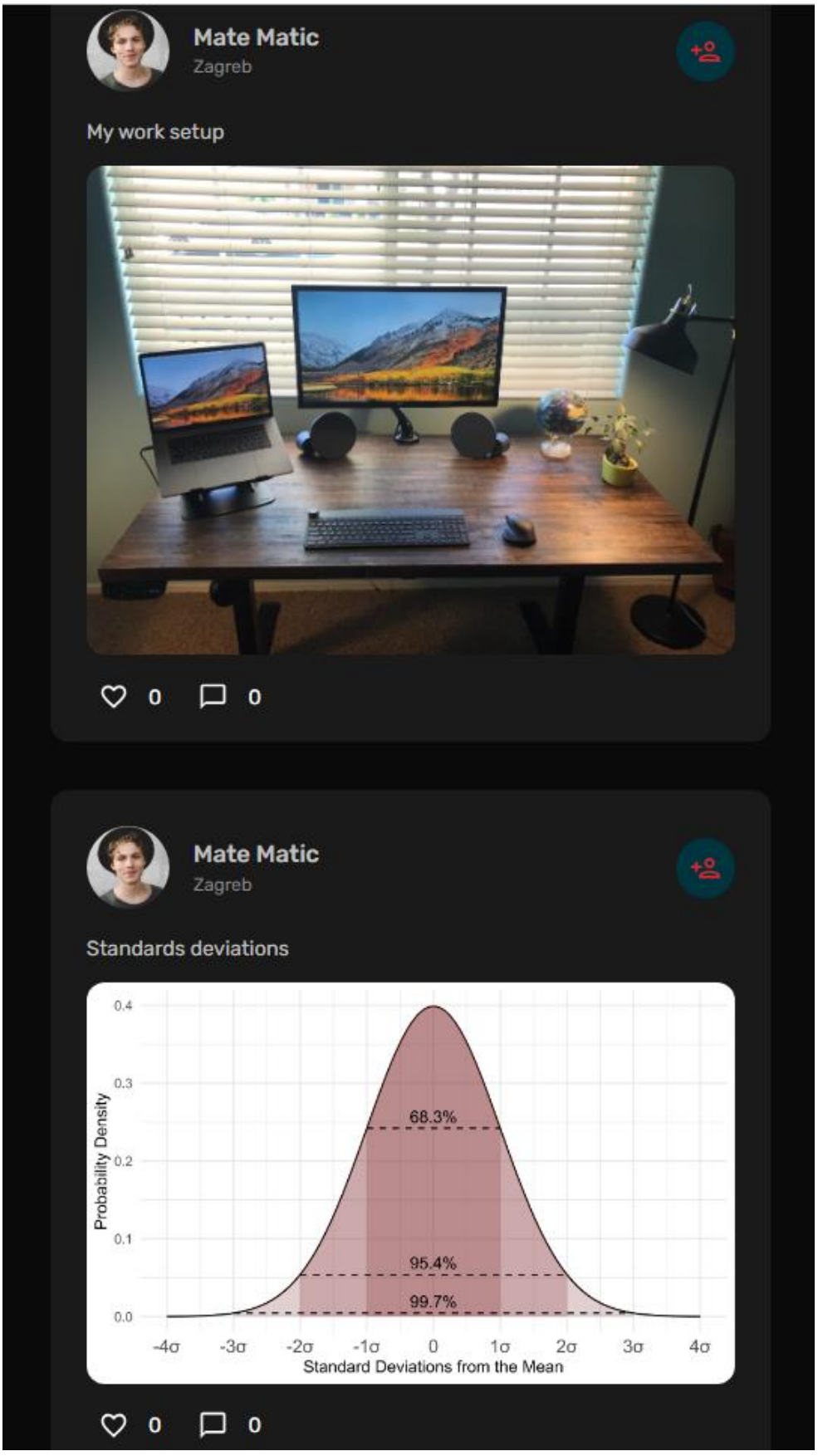
Slika 15: Profilna stranica korisnika

3.10 Mobilni prikaz

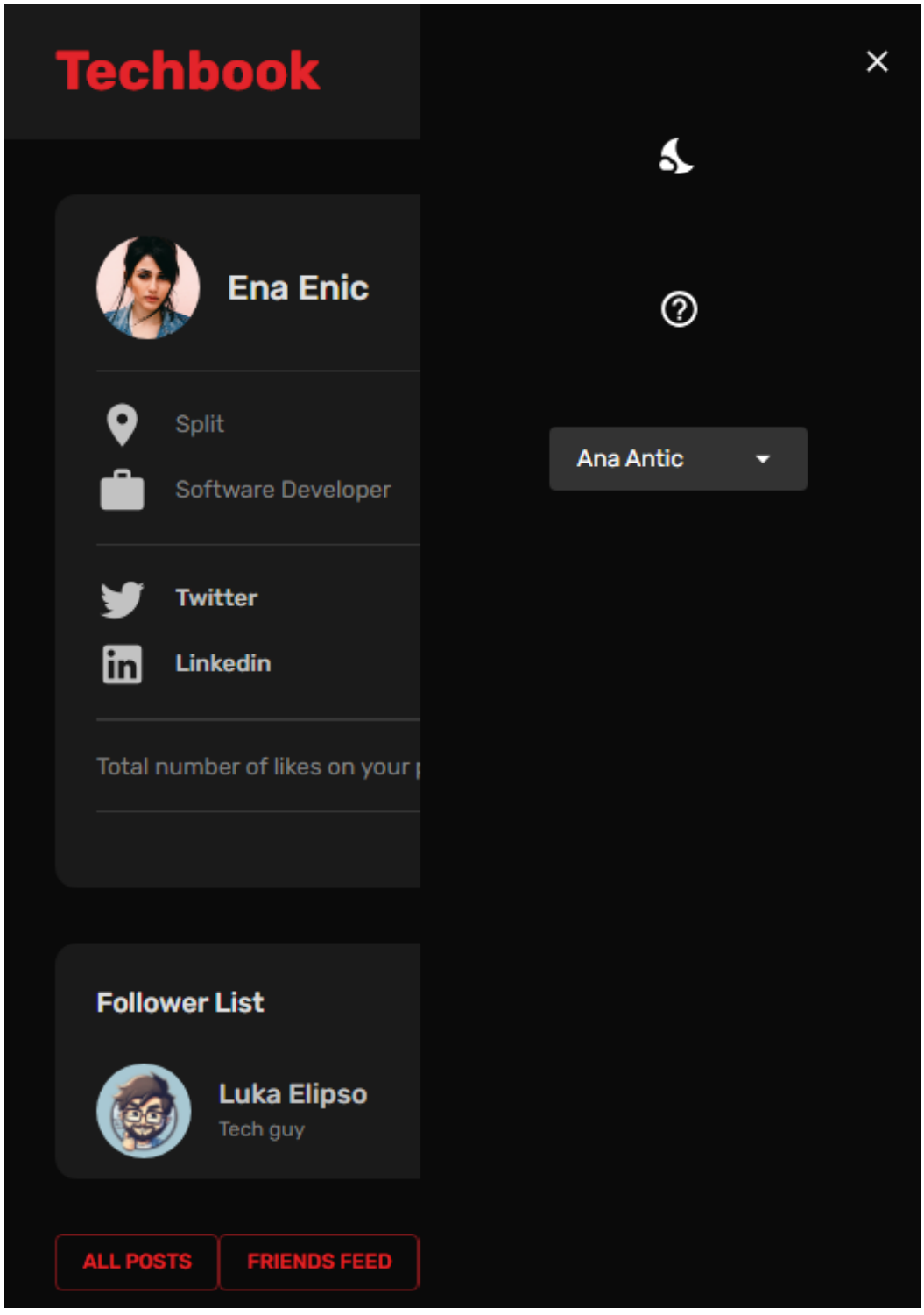
Danas većina korisnika pristupa ovakvim web aplikacijama preko pametnih telefona te je iz tog razloga ova web aplikacija optimizirana za mobilne uređaje. Cilj optimizacije bio je što jednostavniji dizajn koji u najvećoj mogućoj mjeri podsjeća na originalni web prikaz (slika 16, slika 17, slika 18). Pametni telefoni nude prenosivost, omogućujući korisnicima pristup web aplikaciji s bilo kojeg mjesta i povećavaju angažman korisnika.. Mobilna optimizacija potiče bolje performanse, što dovodi do bržeg vremena učitavanja, povećane stope korištenja stranice i povećanog zadovoljstva korisnika. Negativna strana je to što postoji širok raspon modela pametnih telefona i veličina zaslona, što dovodi do fragmentacije uređaja. Osiguravanje kompatibilnosti svih ovih uređaja može biti izazovno.



Slika 16: Mobilni prikaz 1



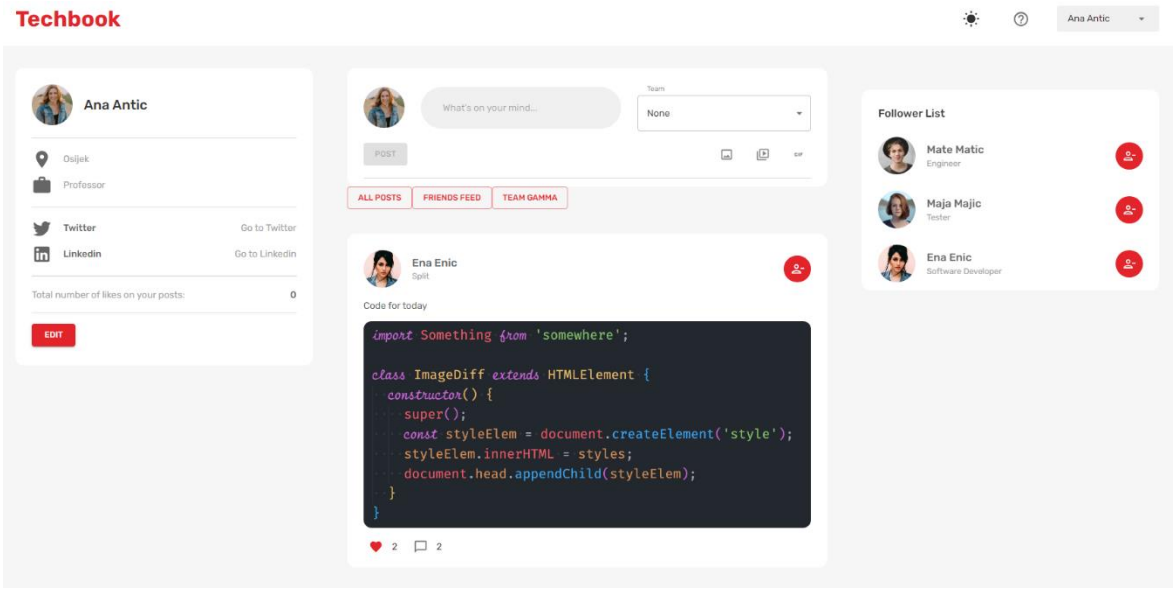
Slika 17: Mobilni prikaz 2



Slika 18: Mobilni prikaz 3

3.11 Tema aplikacije

Ova web aplikacija sastoji se od normalne svijetle teme (slika 19) i tamne teme koja je prikazana kroz cijeli rad. Zadnjih godina mnoge popularne web aplikacije nude svojim korisnicima tamnu temu. Osim što neki korisnici jednostavno uživaju u estetici tamnijeg dizajna, mnogi tamne teme koriste jer osjećaju da im je takav način rada ugodniji za oči. To se osobito primjećuje u okruženju gdje je osvjetljenje slabije, ili je potpuni mrak gdje je svjetlina ekrana u visokom kontrastu u odnosu na okolinu. Još jedan od razloga koji se navodi kao prednost korištenja tamnih tema na pametnim uređajima jest produženje trajanja baterije.

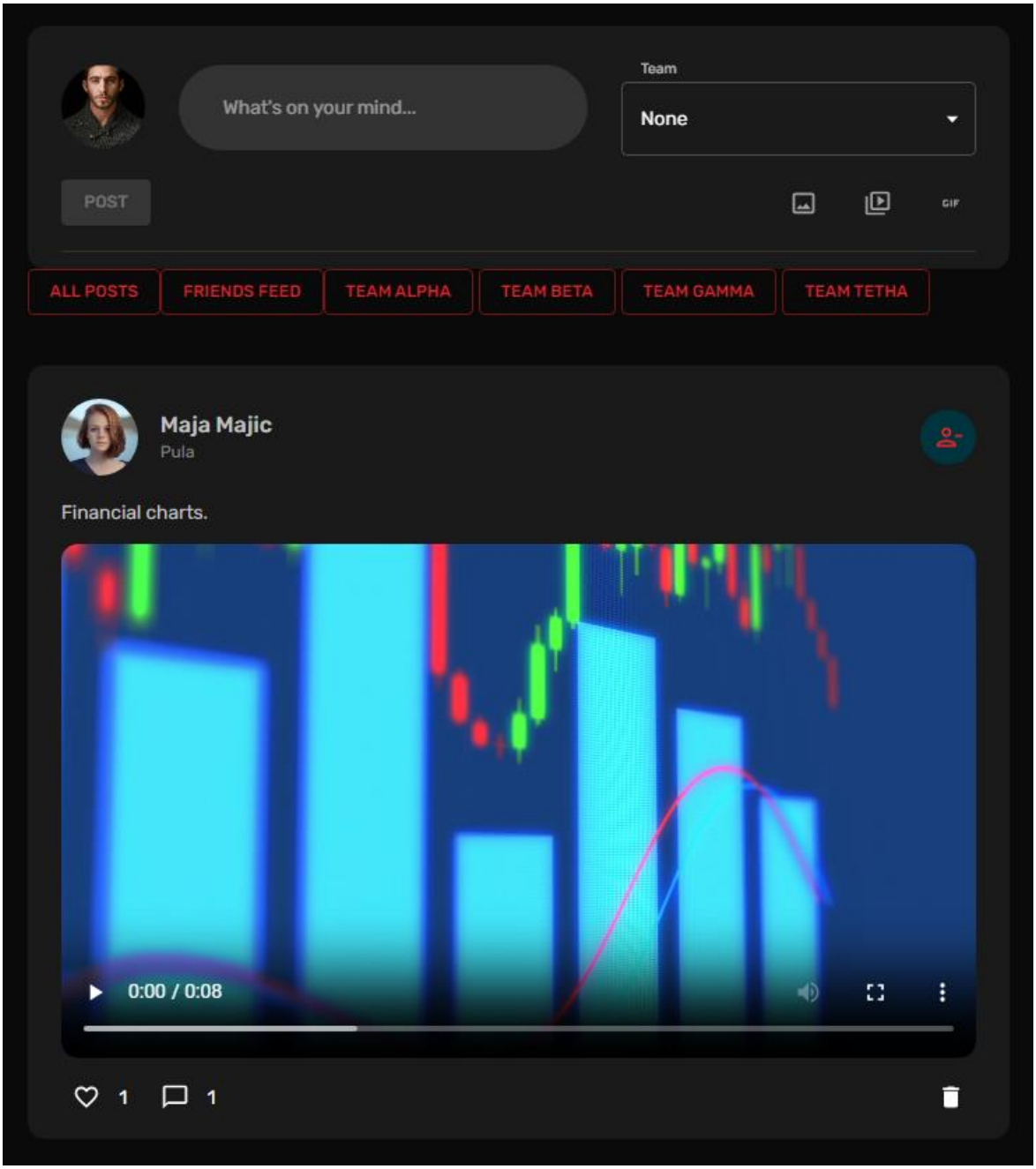


Slika 19: Svijetla tema

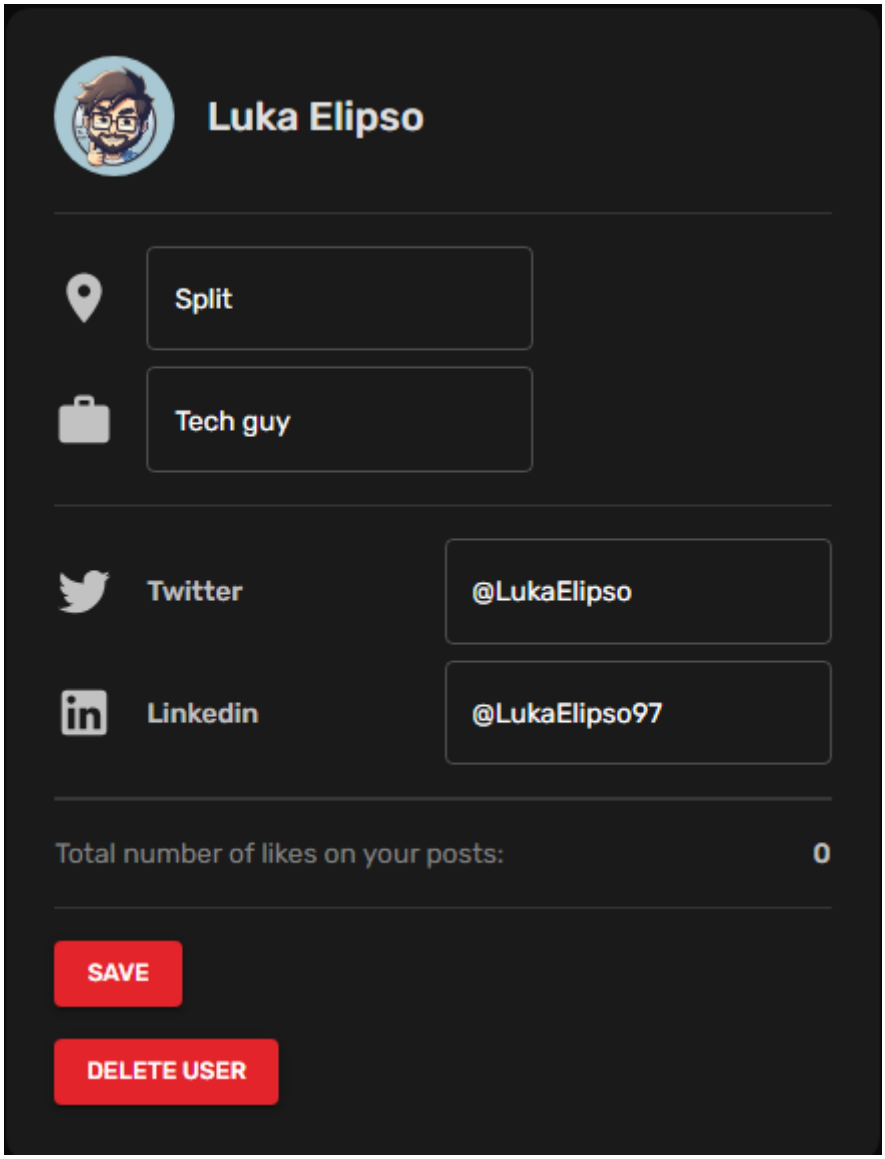
3.12 Korisnik administrator

Administrator je pojedinac ili skupina pojedinaca koji imaju posebne privilegije i kontrolu nad upravljanjem i funkcioniranjem web stranice ili online platforme. Uloga administratora obično je povezana s određenim odgovornostima i dopuštenjima koja im omogućuju nadgledanje različitih aspekata web stranice. Administrator ima prava brisati korisnike što je i prikazano na slici 20. Kada je korisnik izbrisan također se brišu sve njegove objave i komentari. Briše se iz liste pratitelja svih korisnika koji su ga zapratili. Osim brisanja korisnika, administrator može brisati objave i uređivati podatke o korisniku, kao i

mijenjati njegove pratitelje što se i vidi na slici 21. Također vidi objave svih timova u kompaniji, a ne samo objave onog tima kojem pripada.



Slika 20: Pogled sa strane administratora



Slika 21: Edit i Delete user opcije kod administratora

4 IMPLEMENTACIJA

Kroz ovo poglavlje bit će prikazani i objašnjeni bitni dijelovi kôda. Dvije glavne kategorije su *backend* kôd i *frontend* kôd.

4.1 *Backend* kôd

Backend kôd uključuje rute za korisničke profile, objave i autentifikaciju. Sve se sprema u bazama podataka, kao što je *MongoDB*, i koriste se *HTTP* metode za izvođenje radnji kao što su dohvaćanje korisničkih podataka, objavljivanje ažuriranja i brisanje korisnika.

4.1.1 Model korisnika

Kod izrade ove web aplikacije za bazu podataka je izabran *MongoDB*. Kod te baze podataka potrebno je napraviti *mongoose* shemu. U toj shemi, koja je prikazana u ispisu 11, definirane su varijable koje u ovom slučaju definiraju korisnika. Prve dvije varijable su ime i prezime tipa *String*. Moraju se sastojati od minimalno 2 znaka, a maksimalno 50 znakova te su obe varijable obavezne za unos. Zatim ide email koji je također tipa *String*, maksimalna dužina je 50 znakova, također je obavezan i mora biti unikatan. Ako se već postoji taj email u bazi podataka korisnik neće moći napraviti novi korisnički račun s tim emailom. U *frontendu* kod upisa podataka je postavljeno da email mora imati strukturu *text@host.domain*. Lozinka je iduća zanimljiva varijabla. Također je tipa *String*, obavezna je za unos i mora imati najmanje 8 znakova. U *frontendu* su postavljena dodatna ograničenja da se lozinka mora sastojati od bar jednog slova, jednog broja i jednog specijalnog znaka. Varijable *Twitter* i *LinkedIn* poveznice te varijabla *put do slike*, lokacija i zanimanje su sve tipa *String* te im je početna zadana vrijednost prazan *String*. Varijabla pratitelji je tipa *Array* što označava niz podataka i njena početna vrijednost je prazan niz. Ostale su još dvije varijable. *isAdmin* varijabla je tipa *Boolean* koja može primiti samo dvije mogućnosti a to su *true* ili *false* od čega je *false* postavljena kao početna vrijednost. Varijabla tim tipa *Enum* koja ima postavljenje 4 moguće vrijednosti koje predstavljaju imena timova koje korisnik može izabrati.

```
const UserSchema = new mongoose.Schema(
  {
    firstName: {
      type: String, required: true, min: 2, max: 50,
    },
    lastName: {
      type: String, required: true, min: 2, max: 50,
    },
    email: {
      type: String, required: true, max: 50, unique: true,
    },
    password: {
      type: String, required: true, min: 8,
    },
    picturePath: {
      type: String, default: "",
    },
    followers: {
      type: Array, default: [],
    },
    location: String,
    occupation: String,
    impressions: {
      type: Number, default: 0,
    },
    twitterLink: {
      type: String, default: "",
    },
    linkedinLink: {
      type: String, default: "",
    },
    isAdmin: {
      type: Boolean, default: false,
    },
    team: {
      type: String,
     enum: ["Team Alpha", "Team Beta", "Team Gamma", "Team
     Tetha"],
      required: true,
    },
  },
  { timestamps: true }
);
```

Ispis 11: Model korisnika

4.1.2 Model objave

Model objave je *mongoose* shema koja definira objavu. Objava se sastoji od 11 varijabli što se i vidi na ispisu 12. Prve 3 varijable su korisnik ID, ime i prezime koje su tipa *String* i koje pri kreiranju objave spremaju informacije o korisniku koji je kreirao objavu kako bi ju se moglo povezati sa korisnikom. Osim podataka korisnika sprema se putanja do slike, videozapisa ili *GIF-a* u *String* obliku i opis same objave. Količina lajkova se sprema u mapu tipa *Boolean*. Ključ je ID od korisnika koji je lajkao objavu, a vrijednost je „*true*“ ili „*false*“. Komentari su objekt koji se sastoji od 3 *String* varijable. To su ime i prezime korisnika koji je komentirao objavu i sam njegov komentar. Na samom kraju tu je tim tipa *Enum* kojem je početna zadana vrijednost „None“, a može imati i 4 druge opcije.

```
const postSchema = mongoose.Schema(
  {
    userId: {
      type: String,
      required: true,
    },
    firstName: {
      type: String,
      required: true,
    },
    lastName: {
      type: String,
      required: true,
    },
    location: String,
    description: String,
    picturePath: String,
    videoPath: String,
    userPicturePath: String,
    likes: {
      type: Map,
      of: Boolean,
    },
    comments: [
      {
        firstName: String,
        lastName: String,
        comment: String,
      }
    ],
```

```
    team: {
      type: String,
      enum: ["None", "Team Alpha", "Team Beta", "Team
Gamma", "Team Tetha"],
      default: "None",
    },
  },
  { timestamps: true }
);
```

Ispis 12: Model objave

4.1.3 Verifikacija tokena

Međuprogramski kôd definira funkciju pod nazivom *verifyToken* koja se koristi za autentifikaciju i provjeru *JSON web tokena* (*JWT*) poslanih u zaglavlju zahtjeva čija je implementacija prikazana na ispisu 13. Prvo provjerava prisutnost tokena, a ako ga ne pronađe, šalje odgovor „*403 Access denied*“. Ako token postoji, ekstrahira ga i dekodira uklanjanjem prefiksa „*Bearer*“. Zatim koristi biblioteku *jsonwebtoken* za provjeru autentičnosti tokena pomoću tajnog ključa. Ako je provjera uspješna, međuprogram prilaže dekodirane korisničke informacije i predaje kontrolu sljedećem međuprogramu ili rukovatelju rutom. Ako se dogodi greška ona ću biti ispisana u konzoli preglednika.

```
try {
    let token = req.header("Authorization");

    if (!token) {
      return res.status(403).send("Access to account
denied");
    }

    if (token.startsWith("Bearer ")) {
      token = token.slice(7, token.length).trimLeft();
    }
    const verified = jwt.verify(token,
process.env.JWT_SECRET);
    req.user = verified;
    next();
  } catch (err) {
  }
```

Ispis 13: Kôd za verifikaciju tokena

4.1.4 Metoda prijave

Ispis 14 definira funkciju pod nazivom *login* koja upravlja provjerom autentičnosti prijave korisnika u web aplikaciji. Prvi korak je da korisnik upiše svoj email i lozinku i stisne dugme „prijava“. Nakon toga šalje se zahtjev u bazu koji traži postoji li korisnik s upisanom email adresom u bazi podataka. Ako ne postoji, vrati poruku, ako postoji ide se na idući korak koji je provjera lozinke. Pomoću metode *bcrypt.compare* uspoređuje se upisana lozinka s heširanom lozinkom spremljenom u bazi podataka. Ako lozinka nije dobra, kôd vraća poruku, a ako je lozinka dobra, kreira se token pomoću kojeg se korisnik prijavljuje u web aplikaciju i koji sadrži određene informacije koje će kasnije biti potrebne.

```
export const login = async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: "Invalid email or
password." });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(401).json({ message: "Invalid email or
password." });
    }
    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);
    const userWithoutPassword = { ...user.toObject() };
    delete userWithoutPassword.password;

    res.status(200).json({ token, user: userWithoutPassword });
  } catch (err) {
    console.error(err);
    if (err.name === 'MongoError') {
      return res.status(500).json({ message: "Database error." });
    } else {
      return res.status(500).json({ message: "Internal server error."
});
    }
  }
};
```

Ispis 14: Metoda prijave

4.1.5 Metoda registracije

Ispis 15 definira metodu pod nazivom *register*, koja upravlja registracijom korisnika u web aplikaciji. Prvi korak je uzimanje svih informacija koje korisnik upiše kada popunjava obrazac za registraciju. Zatim se pomoću metoda *bcrypt.genSalt* i *bcrypto.hash* lozinka hešira. Heširanje je kada se od normalne lozinke recimo „testLozinka123“ napravi *hash* te lozinke koja izgleda ovako: *$2b$10$OfHsyYez56dZGZlxiKnwquFx9o7pFZ0MZXAamdjdQGIN.OM6iqg.m*

Da bi se od tog niza znakova dobila nazad originalna lozinka potrebno je znati tu lozinku te je kao u metodi za prijavu potrebno usporediti tu lozinku sa heširanom lozinkom. Nakon toga kreira se novi korisnik sa svim unesenim informacijama koje se upisuju u bazu podataka. U bazi svaki korisnik ima svoj unikatan ID. Ako je sve u redu dobije se poruka 201, ako dođe do greške dobije se poruka 500.

```
export const register = async (req, res) => {
  try {
    const {
      firstName, lastName, email, password, picturePath, friends,
location, occupation, impressions, team,
    } = req.body;
    const salt = await bcrypt.genSalt();
    const passwordHash = await bcrypt.hash(password, salt);
    const newUser = new User({
      firstName, lastName, email, password: passwordHash,
picturePath, friends, location, occupation, impressions, team,
    });
    const savedUser = await newUser.save();
    res.status(201).json(savedUser);
  } catch (err) {
    console.error(err); // Log the error
    if (err.name === 'MongoError' && err.code === 11000) {
      return res.status(400).json({ message: "Email is already
registered." });
    } else {
      return res.status(500).json({ message: "Internal server error."
});
    }
  }
};
```

Ispis 15: Metoda registracije

4.1.6 Kreiranje objave

Ispis 16 definira funkciju pod nazivom *createPost*. Ova funkcija je asinkrona i upravlja stvaranjem novog posta u web aplikaciji. Kôd izvlači svojstva kao što su korisnikov ID, opis, lokacija slike/videozapisa/GIFa i tim iz tijela zahtjeva pomoću destrukturiranja. Očekuje se da će ova svojstva biti poslana u zahtjevu kada se kreira novi post. Funkcija provjerava postoji li korisnik koji ima navedeni *ID* u bazi podataka pozivanjem *User.findById(userId)*. Ako korisnik ne postoji, šalje odgovor "*400 Bad request*" s JSON objektom koji sadrži poruku o pogrešci: "Korisnik nije pronađen." Ako korisnik postoji novokreirana objava sprema se u bazu pomoću metode spremanja. Ključna riječ *await* koristi se za čekanje završetka operacije baze podataka. Nakon što je nova objava spremljena, kôd dohvaća sve ažurirane objave iz baze podataka koristeći *Post.find()*. Ključna riječ *await* ponovno se koristi za čekanje završetka upita. Ako je sve uspješno, funkcija šalje odgovor "*201 Created*" s JSON objektom koji sadrži ažurirani popis objava. Ako se dogodi bilo kakva pogreška tijekom izvođenja funkcije (kao što su upiti u bazu podataka, stvaranje objave ili spremanje), ona hvata pogrešku. Zatim šalje odgovor "*500 Internal server error*" s JSON objektom koji sadrži opću poruku o pogrešci: "Došlo je do pogreške prilikom stvaranja objave."

```
export const createPost = async (req, res) => {
  try {
    const { userId, description, picturePath, team } = req.body;

    const user = await User.findById(userId);
    if (!user) {
      return res.status(400).json({ message: "User not found." });
    }

    const newPost = new Post({
      userId,
      firstName: user.firstName,
      lastName: user.lastName,
      location: user.location,
      userPicturePath: user.picturePath,
      description, picturePath, likes: {}, comments: [], team,
    });

    await newPost.save();
```

```
    const posts = await Post.find();
    res.status(201).json(posts);
  } catch (err) {
    console.error(err);
    if (err.name === 'ValidationError') {
      return res.status(400).json({ message: "Validation error.
Please check your input." });
    } else {
      return res.status(500).json({ message: "Internal server error."
});
    }
  }
};
```

Ispis 16: Metoda kreiranja objave

4.1.7 Lajk objave

Ispis 17 definira funkciju pod nazivom *likePost*. Ova funkcija upravlja procesom lajka ili uklanjanja lajka objave u web aplikaciji. Kôd izdvaja *ID* parametar iz URL-a (*req.params*) i Korisnik ID iz tijela zahtjeva (*req.body*). Ove su vrijednosti potrebne za identifikaciju objave i korisnika koji je izvršio radnju lajka. Metoda dohvaća objavu s navedenim ID-om koristeći *Post.findById(id)*. Ako objava ne postoji, šalje odgovor "404 Nije pronađeno“ s određenom porukom o pogrešci: "Objava nije pronađena." Zatim se provjerava je li korisnik s navedenim korisničkim ID-om već lajkao objavu. To se utvrđuje provjerom postoji li korisnikov ID u mapi sviđanja objave. Ako je korisniku već lajkao objavu, kôd uklanja korisnikov ID s mape lajkova. Ako korisniku nije lajkana objava, kôd dodaje korisnikov ID na mapu lajkova. Nakon promjene statusa lajka, kôd ažurira polje lajka objave pomoću *Post.findByIdAndUpdate*. Opcija { *new: true* } osigurava vraćanje ažurirane objave kao odgovora. U slučaju greške ispisuje se poruka.

```
export const likePost = async (req, res) => {
  try {
    const { id } = req.params;
    const { userId } = req.body;

    const post = await Post.findById(id);
    if (!post) {
      return res.status(404).json({ message: "Post not found." });
    }
```

```
    const isLiked = post.likes.get(userId);

    if (isLiked) {
      post.likes.delete(userId);
    } else {
      post.likes.set(userId, true);
    }

    const updatedPost = await Post.findByIdAndUpdate(
      id,
      { likes: post.likes },
      { new: true }
    );
    res.status(200).json(updatedPost);
  } catch (err) {
    console.error(err);
    if (err.name === 'CastError' && err.kind === 'ObjectId') {
      return res.status(400).json({ message: "Invalid post ID." });
    } else {
      return res.status(500).json({ message: "Internal server error."
});
    }
  }
};
```

Ispis 17: Metoda lajka objave

4.1.8 Dodavanje komentara na objavu

Ispis 18 definira funkciju pod nazivom *addComment*. Ova funkcija upravlja postupkom dodavanja komentara objavi u web aplikaciji. Kôd izdvaja ID parametar iz URL-a (*req.params*) i svojstva kao što su komentar, ime i prezime iz tijela zahtjeva (*req.body*). Te su vrijednosti potrebne za identifikaciju objave i pružanje informacija o komentaru. Metoda dohvaća objavu s navedenim ID-om koristeći *Post.findById(id)*. Ako objava ne postoji, šalje odgovor "*404 Not found*" s određenom porukom o pogrešci: "Objava nije pronađena." Novi objekt komentara stvara se pomoću ekstrahiranog komentara, imena i prezimena. Komentar se dodaje u niz komentara posta metodom *push*, a ažurirana objava, koja sada sadrži novi komentar, sprema se pomoću metode *post.save()*. Ako dođe do greške ispisuje se prikladna poruka.

```
export const addComment = async (req, res) => {
  try {
    const { id } = req.params;
    const { comment, firstName, lastName } = req.body;

    const post = await Post.findById(id);
    if (!post) {
      return res.status(404).json({ message: "Post not found." });
    }

    const newComment = {
      comment, firstName, lastName,
    };

    post.comments.push(newComment);
    const updatedPost = await post.save();

    res.status(200).json(updatedPost);
  } catch (err) {
    console.error(err); // Log the error
    if (err.name === 'CastError' && err.kind === 'ObjectId') {
      return res.status(400).json({ message: "Invalid post ID." });
    } else {
      return res.status(500).json({ message: "Internal server error."
});
    }
  }
};
```

Ispis 18: Metoda dodavanja komentara na objavu

4.1.9 Prati/otprati korisnika

Ispis 19 definira funkciju pod nazivom *followUnfollowUsers*. Ova funkcija upravlja procesom dopuštanja autentificiranom korisniku da prati ili prestane pratiti druge korisnike u web aplikaciji. Metoda dohvaća autentificiranog korisnika i pratitelja pomoću navedenih ID-ova. Ako korisnik ili pratitelj ne postoje, šalje odgovor "*404 Not found*" s određenom porukom o pogrešci: "Korisnik ili pratitelj nisu pronađeni." Ovisno o tome je li korisnik već na korisnikovom popisu pratitelja, ili uklanja ID pratitelja iz polja pratitelja korisnika ili dodaje ID pratitelja u niz pratitelja korisnika. Funkcija sprema ažurirane popise pratitelja koristeći metodu spremanja. Nakon ažuriranja popisa pratitelja, dohvaća ažurirani popis i formatira podatke za svakog pratitelja. To uključuje ID, ime, prezime, zanimanje, lokaciju i putanju profilne slike.

```
export const followUnfollowUser = async (req, res) => {
  try {
    const { id: authenticatedUserId, followerId } = req.params;

    const user = await User.findById(authenticatedUserId);
    const follower = await User.findById(followerId);
    if (!user || !follower) {
      return res.status(404).json({ message: "User or follower not
found." });
    }

    if (authenticatedUserId !== user._id.toString()) {
      return res.status(403).json({ message: "Access denied. Cannot
add/remove followers for other users." });
    }
    if (user.followers.includes(followerId)) {
      user.followers = user.followers.filter((id) => id !==
followerId);
      follower.following = follower.following.filter((id) => id !==
authenticatedUserId);
    } else {
      user.followers.push(followerId);
      follower.following.push(authenticatedUserId);
    }
    await user.save();
    await follower.save();

    const followers = await User.find({ _id: { $in: user.followers }
});
    const formattedFollowers = followers.map(({ _id, firstName,
lastName, occupation, location, picturePath }) => {
      return { _id, firstName, lastName, occupation, location,
picturePath };
    });
    res.status(200).json(formattedFollowers);
  } catch (err) {
    console.error(err);
    if (err.name === 'CastError' && err.kind === 'ObjectId') {
      return res.status(400).json({ message: "Invalid user ID or
follower ID." });
    } else {
      return res.status(500).json({ message: "Internal server error."
});
    }
  }
};
```

Ispis 19: Metoda dodavanja i uklanjanja pratitelja

4.1.10 Ažuriranje korisničke kartice

Ispis 20 definira funkciju pod nazivom *updateUserInfo*. Ova metoda upravlja procesom ažuriranja polja lokacije, zanimanja, poveznice na *Twitter* i veze na *LinkedIn* korisnički profil. Metoda provjerava podudara li se ID ciljanog korisnika s ID-om provjerenog korisnika. Ako se ne podudaraju, šalje odgovor "*403 Forbidden*" s određenom porukom o pogrešci: "Pristup odbijen. Nije moguće ažurirati korisnički profil za druge korisnike." Prije pokušaja ažuriranja korisničkog profila, dohvaća korisnika s navedenim ID-om. Ako korisnik ne postoji, šalje odgovor "*404 Not found*" s određenom porukom o pogrešci: "Korisnik nije pronađen." Funkcija potvrđuje i ažurira korisnička polja koja se mogu ažurirati. Definira niz pod nazivom *allowedFields* koji uključuje "lokaciju", "zanimanje", "*twitterLink*" i "*linkedinLink*". Iterira kroz ovaj niz i provjerava jesu li polja prisutna u tijelu zahtjeva i jesu li dopuštena za ažuriranje. Ako je tako, ažurira ta polja za korisnika. Nakon ažuriranja korisničkog profila, funkcija sprema promjene pomoću metode spremanja na objektu korisnika.

```
export const updateUserInfo = async (req, res) => {
  try {
    const { id } = req.params;
    const authenticatedUserId = req.user._id;
    const updatedUser = req.body;

    if (id !== authenticatedUserId) {
      return res.status(403).json({ message: "Access denied. Cannot
update user profile for other users." });
    }

    const allowedFields = ["location", "occupation", "twitterLink",
"linkedinLink"];
    const user = await User.findById(id);
    if (!user) {
      return res.status(404).json({ message: "User not found." });
    }

    allowedFields.forEach((field) => {
      if (updatedUser[field] !== undefined) {
        user[field] = updatedUser[field];
      }
    });

    await user.save();
```

```
    res.status(200).json(user);
  } catch (err) {
    console.error(err);
    if (err.name === 'CastError' && err.kind === 'ObjectId') {
      return res.status(400).json({ message: "Invalid user ID." });
    } else {
      return res.status(500).json({ message: "Internal server error."
});
    }
  }
};
```

Ispis 20: Metoda ažuriranja korisničke kartice

4.1.11 Brisanje korisnika

Ispis 21 definira funkciju pod nazivom *deleteUser*. Ova funkcija upravlja postupkom brisanja korisničkog računa i povezanih podataka. Metoda dohvaća korisnika s navedenim ID-om koristeći *User.findById*. Ako korisnik ne postoji, šalje odgovor "*404 Not found*" s određenom porukom o pogrešci: "Korisnik nije pronađen." Funkcija provjerava podudara li se *authenticatedUserId* s ID-om korisnika kojeg treba izbrisati. Ako odgovaraju, šalje odgovor "*403 Forbidden*" s porukom o pogrešci: "Pristup odbijen. Nije moguće izbrisati vaš račun." Funkcija briše sve objave povezane s korisnikom koji se briše. Koristi *Post.deleteMany* za brisanje postova gdje ID odgovara ID-u korisnika. Također se ažurira popis pratitelja drugih korisnika kako bi uklonio ID korisnika koji se briše. Koristi *User.updateMany* s operatorom *$pull* za uklanjanje ID-a iz niza pratitelja drugih korisnika. Na samom kraju koristi se *User.deleteOne* za brisanje samog korisničkog računa na temelju navedenog ID-a.

```
export const deleteUser = async (req, res) => {
  try {
    const { id } = req.params;
    const authenticatedUserId = req.user._id;
    const { authenticatedUserId: requestAuthenticatedUserId } =
req.body;
    const userToDelete = await User.findById(id);

    if (!userToDelete) {
      return res.status(404).json({ message: "User not found." });
    }
```

```
    if (authenticatedUserId !== requestAuthenticatedUserId) {
      return res.status(403).json({ message: "Access denied. You can
only delete your own account." });
    }
    if (authenticatedUserId === userToDelete._id.toString()) {
      return res.status(403).json({ message: "Access denied. Cannot
delete your own account." });
    }

    await Post.deleteMany({ userId: id });

    await User.updateMany(
      { _id: { $in: userToDelete.friends } },
      { $pull: { friends: id } }
    );

    await User.deleteOne({ _id: id });

    res.status(200).json({ message: "User deleted successfully." });
  } catch (err) {
    console.error(err); // Log the error
    if (err.name === 'CastError' && err.kind === 'ObjectId') {
      return res.status(400).json({ message: "Invalid user ID." });
    } else {
      return res.status(500).json({ message: "Internal server error."
});
    }
  }
};
```

Ispis 21: Metoda brisanja korisnika

4.1.12 Rute

Uvozi se ekspresna biblioteka za stvaranje ruta i kontrolera za rukovanje raznim radnjama povezanim s objavama kao što su povlačenje objava pratitelja, lajkanje, komentiranje i brisanje objava. Također uvozi *verifyToken* međuprogram za provjeru autentičnosti. Definirane su 4 GET rute što je i prikazano na ispisu 22. One dohvaćaju određene podatke po postavljenim parametrima. PATCH rute ažuriraju već postojeće podatke u bazi. Također imaju određene parametre koji određuju koji će se sve podaci ažurirati. Na kraju tu je DELETE ruta koja briše objavu iz same baze podataka.

```
/* READ */
router.get("/", verifyToken, getFeedPosts);
router.get("/:userId/posts", verifyToken, getUserPosts);
router.get("/user/:userId/friends-posts", verifyToken,
getUserFriendPosts);

/* UPDATE */
router.patch("/:id/like", verifyToken, likePost);
router.patch("/:id/comment", verifyToken, addComment);

/* DELETE */
router.delete("/:id", verifyToken, deletePost);
```

Ispis 22: Prikaz ruta za objave

4.1.13 Spajanje s bazom podataka

Postavlja se poslužitelj *Express.js* za slušanje na portu definiranom varijablom okruženja PORT ili prema zadanim postavkama na portu 6001. Povezuje se s *MongoDB* bazom podataka, koristeći URL naveden u varijabli okruženja *MONGO_URL*, čija implementacija je prikazana na ispisu 23. Ako je veza uspješna, poslužitelj se pokreće, a njegov port se bilježi. U slučaju pogreške u vezi, bilježi se poruka o pogrešci. Kôd obrađuje inicijalizaciju poslužitelja i povezivanje s bazom podataka, osiguravajući glatko pokretanje web aplikacije.

```
const PORT = process.env.PORT || 5001;
mongoose
  .connect(process.env.MONGO_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => {
    app.listen(PORT, () => console.log(`Server Port: ${PORT}`));
  })
  .catch((error) => console.log(`${error} did not connect`));
```

Ispis 23: Kôd za spajanje s bazom podataka

4.1.14 Verzije korištenih biblioteka za *backend*

U datoteci *package.json* nalazi se popis i verzije svih vanjskih *backend* biblioteka koje koristimo (ispis 24). Najbolja praksa je uvijek imati zadnje stabilne verzije korištenih biblioteka.

```
{
  "dependencies": {
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "gridfs-stream": "^1.1.1",
    "helmet": "^6.0.1",
    "jsonwebtoken": "^9.0.0",
    "mongodb": "^5.3.0",
    "mongoose": "^7.0.0",
    "morgan": "^1.10.0",
    "multer": "^1.4.4",
    "multer-gridfs-storage": "^5.0.2"
  },
```

Ispis 24: Korištene vanjske biblioteke u backend kôdu

4.2 *Frontend* kôd

Frontend kôd je dio web aplikacije okrenut korisniku. Uključuje vizualno sučelje, izgrađeno s tehnologijama kao što su HTML, CSS i *JavaScript* okviri, primjerice *React* ili *Angular*. Omogućuje korisnicima interakciju s aplikacijom, pregled sadržaja i izvođenje radnji, a sve to tijekom komunikacije s pozadinskim poslužiteljem radi dohvaćanja i ažuriranja podataka.

4.2.1 Početna stranica

Komponenta početne stranice generira izgled web stranice koristeći *React* i *Material-UI*. Započinje provjerom širine zaslona kako bi se utvrdilo radi li se o zaslonu koji nije mobilni. Zatim izdvaja ID korisnika i putanju slike iz *Redux* trgovine. Struktura stranice sastoji se od komponente *Navbar* i više odjeljaka: *UserWidget*, koji prikazuje podatke o korisniku, *MyPostWidget* za korisničke objave i *PostsWidget* za druge objave. Ako se radi

o zaslonu koji nije mobilni, također se prikazuje odjeljak *FollowerListWidget*. Izgled se prilagođava na temelju veličine zaslona, organizirajući komponente unutar fleksibilnog rasporeda okvira za privlačno korisničko iskustvo. Cijela implementacija je prikazana na ispisu 25.

```
return (
    <Box>
      <Navbar />
      <Box
        width="100%"
        padding="2rem 6%"
        display={isNonMobileScreens ? "flex" : "block"}
        gap="0.5rem"
        justifyContent="space-between"
      >
        <Box flexBasis={isNonMobileScreens ? "26%" : undefined}>
          <UserWidget userId={_id} picturePath={picturePath} />
        </Box>
        <Box
          flexBasis={isNonMobileScreens ? "42%" : undefined}
          mt={isNonMobileScreens ? undefined : "2rem"}
        >
          <MyPostWidget picturePath={picturePath} />
          <PostsWidget userId={_id} />
        </Box>
        {isNonMobileScreens && (
          <Box flexBasis="26%">
            <Box m="2rem 0" />
            <FriendListWidget userId={_id} />
          </Box>
        )}
      </Box>
    </Box>
  );
};
```

Ispis 25: Kôd za prikaz početne stranice

4.2.2 Metoda prijave pozivom *backend* API-ja

Funkcija prijave je asinkrona funkcija koja upravlja provjerom autentičnosti korisnika. Šalje POST zahtjev krajnjoj točki prijave lokalnog poslužitelja s vrijednostima koje daje korisnik kao što su email i lozinka. Zatim čeka odgovor i pokušava ga raščlaniti

u JSON format. Ako je uspješan, provjerava sadrži li odgovor token i korisničke podatke. Ako su oba prisutna, ažurira stanje korisnika akcijom *setLogin*, pohranjuje korisničke podatke i token i vodi do početne stranice. Ako odgovor ne sadrži važeće podatke, prikazuje poruku o pogrešci kroz *onSubmitProps.setErrors*. U slučaju bilo kakvih pogrešaka, bilježi ih i na kraju postavlja stanje slanja u *false*. Cijela implementacija je prikazana na ispisu 26.

```
const login = async (values, onSubmitProps) => {
    try {
      const loggedInResponse = await
fetch('http://localhost:3001/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(values),
      });
      const loggedIn = await loggedInResponse.json();

      if (loggedIn.token && loggedIn.user) {
        dispatch(
          setLogin({
            user: loggedIn.user,
            token: loggedIn.token,
          })
        );
        navigate('/home');
      } else {
        onSubmitProps.setErrors({ loginError: 'Wrong email or
password' });
      }
    } catch (error) {
      console.error(error); // Log the error
      onSubmitProps.setErrors({ loginError: 'An error occurred while
logging in. Please try again.' });
    } finally {
      onSubmitProps.setSubmitting(false);
    }
  };
```

Ispis 26: Metoda prijave pozivom backend API-ja

4.2.3 Metoda registracije pozivom backend API-ja

Funkcija registracije (ispis 27) upravlja procesom registracije korisnika. To je asinkrona metoda koja uzima dva argumenta: vrijednosti, koje sadrže podatke o korisničkom unosu, i *onSubmitProps*, koji pruža svojstva vezana za podnošenje obrasca. Unutar metode kreira se objekt *FormData* pod istim nazivom, za pohranu korisničkog unosa. Petlja prolazi kroz svaki par ključ-vrijednost u objektu vrijednosti i dodaje ih u *formData*. Metoda zatim šalje POST zahtjev krajnjoj točki registracije poslužitelja na *'http://localhost:3001/auth/register'*. Zahtjev uključuje *formData* kao tijelo zahtjeva. Nakon što poslužitelj odgovori, rezultat se analizira kao JSON u varijabli *savedUser*. Ako je postupak registracije uspješan poziva se metoda *onSubmitProps.resetForm()* za ponovno postavljanje polja obrasca. Konačno, funkcija *setPageType('login')* nakon uspješne registracija pozicionira korisnika na stranicu za prijavu.

```
const register = async (values, onSubmitProps) => {
    try {
      const formData = new FormData();
      for (let value in values) {
        formData.append(value, values[value]);
      }
      formData.append('picturePath', values.picture.name);

      const savedUserResponse = await
fetch('http://localhost:3001/auth/register', {
        method: 'POST',
        body: formData,
      });
      const savedUser = await savedUserResponse.json();
      onSubmitProps.resetForm();

      if (savedUser) {
        setPageType('login');
      }
    } catch (error) {
      console.error(error);
      onSubmitProps.setErrors({ registerError: 'An error occurred
while registering. Please try again.' });
    }
  };
```

Ispis 27: Metoda registracije pozivom backend API-ja

4.2.4 Shema za prijavu i registraciju

U shemi za registraciju korištenjem *Yup-a* su sva spomenuta polja postavljena kao obavezna što se vidi na ispisu 28. Email adresa mora imati strukturu klasične email adrese dok lozinka mora imati minimalno 8 znakova, od koji bar jedan znak mora bit slovo, jedan broj i jedan specijalni znak. Kod login sheme email također mora imati klasičnu strukturu dok za password nema posebnih ograničenja.

```
const registerSchema = yup.object().shape({
  firstName: yup.string().required('required'),
  lastName: yup.string().required('required'),
  email: yup.string().email('Invalid email format, Example:
text@text.com').required('required'),
  password: yup
    .string()
    .matches(
      /^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-
z\d@$!%*#?&]{8,}$/,
      'Invalid password format: Minimum 8 characters, at least one
letter, one number, and one special character'
    )
    .required('required'),
  location: yup.string().required('required'),
  occupation: yup.string().required('required'),
  picture: yup.string().required('required'),
});

const loginSchema = yup.object().shape({
  email: yup.string().email('invalid email').required('required'),
  password: yup.string().required('required'),
});
```

Ispis 28: Sheme u Yup-u za prijavu i registraciju

4.2.5 Kalup profilne slike korisnika

Sastoji se od dvije stvari, slike za naziv slikovne datoteke i veličine za dimenzije slike, sa zadanom vrijednošću od 60 piksela što je i prikazano na ispisu 29. Komponenta prikazuje sliku unutar okvira, primjenjujući CSS za odgovarajuću veličinu i kružni radijus obruba. Slika se učitava s krajnje točke poslužitelja pomoću dostavljene slike. To je

komponenta za višekratnu upotrebu koja stvara vizualno dosljedan kružni prikaz korisničke slike, koji se često koristi u profilnim slikama ili avatarima.

```
return (
    <Box width={size} height={size}>
      <img
        style={{ objectFit: "cover", borderRadius: "50%" }}
        width={size}
        height={size}
        alt="user"
        src={`http://localhost:3001/assets/${image}`}
      />
    </Box>
  );
};
```

Ispis 29: Kalup profilne slike korisnika

4.2.6 Redux kôd

Navedeni kôd definira *Redux* odsječak pod nazivom *authSlice* pomoću metode *createSlice*. Ovaj odsječak upravlja stanjem povezanim s autentifikacijom korisnika i podacima povezanim s korisničkim objavama i prijateljima. Isječak ima početne vrijednosti stanja prikazane na ispisu 31.

Uključuje nekoliko metoda reduktora čije su implementacije prikazane na ispisu 30:

- *setMode*: Prebacuje temu aplikacije između svijetle i tamne
- *setLogin*: Ažurira stanje podacima za prijavu korisnika uključujući korisničke podatke, token za provjeru autentičnosti i status administratora.
- *setLogout*: Briše podatke vezane uz korisnika kada se korisnik odjavi.
- *setFollowers*: Ažurira popis pratitelja.
- *setPosts*: Postavlja popis objava
- *setPost*: Ažurira određenu objavu dostavljenim ažuriranim podacima o objavama.

```
reducers: {
    setMode: (state) => {
      state.mode = state.mode === "light" ? "dark" : "light";
    },
    setLogin: (state, action) => {
      state.user = action.payload.user;
      state.token = action.payload.token;
      state.isAdmin = action.payload.user.isAdmin;
    },
    setLogout: (state) => {
      state.user = null;
      state.token = null;
      state.isAdmin = false;
    },
    setFollowers: (state, action) => {
      if (state.user) {
        state.user.friends = action.payload.friends;
      } else {
        console.error("user friends non-existent :(");
      }
    },
    setPosts: (state, action) => {
      state.posts = action.payload.posts;
    },
    setPost: (state, action) => {
      const updatedPosts = state.posts.map((post) => {
        if (post._id === action.payload.post._id) return
action.payload.post;
        return post;
      });
      state.posts = updatedPosts;
    },
  },
```

Ispis 30: Redux metode stanja

```
    const initialState = {
      mode: "light",
      user: {
        friends: [],
      },
      token: null, isAdmin: false, team: "", posts: [],
    };
```

Ispis 31: Redux početne vrijednosti

4.2.7 Rute web aplikacije

Ispis 32 predstavlja strukturu glavne komponente *React* web aplikacije, gdje se različite stranice prikazuju na temelju definiranih ruta. Definirana je početna putanja „/“ koja vodi na stranicu za prijavu. Zatim je definirana putanja „*/home*“ koja vodi na početnu stranu web aplikacije. Putanja „*/profile/:userId*“ vodi na profilnu stranicu određenog korisnika.

```
return (
    <div className="app">
      <BrowserRouter>
        <ThemeProvider theme={theme}>
          <CssBaseline />
          <Routes>
            <Route path="/" element={<LoginPage />} />
            <Route
              path="/home"
              element={isAuthorize ? <HomePage /> : <Navigate to="/"
/>}
            />
            <Route
              path="/profile/:userId"
              element={isAuthorize ? <ProfilePage /> : <Navigate
to="/" />}
            />
          </Routes>
        </ThemeProvider>
      </BrowserRouter>
    </div>
  );
}
```

Ispis 32: Rute web aplikacije

4.2.8 Učitavanje slike, videozapisa ili GIFa

Ispis 33 definira komponentu korisničkog sučelja za rukovanje učitavanjem datoteka u području *dropzone*. Predstavlja ga komponenta *Box* koja se može kliknuti ili povući za dodavanje datoteka. Kada je datoteka odabrana, prikazuje ili naziv datoteke ili pregled sadržaja datoteke, ovisno o vrsti datoteke. Ako je odabrana slika ili video, prikazuje odgovarajući sadržaj s opcijom za uređivanje. Ispis 34 stvara niz dugme ikona za odabir različitih vrsta datoteka u *dropzoni*. Svako dugme povezano je s određenom vrstom datoteke,

poput slike, videozapisa ili GIF-a. Kada se klikne, dugme mijenja boju kako bi označio trenutno odabranu vrstu datoteke.

```
<Box
        {...getRootProps()}
        border={`2px dashed ${palette.primary.main}`}
        p="1rem"
        width="100%"
        sx={{ "&:hover": { cursor: "pointer" } }}
      >
        <input {...getInputProps()} />
        {!file ? (
          <p>
            {dropzoneType === "image"
              ? "Add Image Here": dropzoneType === "video"
              ? "Add Video Here"
              : "Add GIF Here"}
          </p>
        ) : (
          <FlexBetween>
            {dropzoneType === "image" &&
      file.type.includes("image") ? (
              <img
                src={URL.createObjectURL(file)}
                alt={file.name}
                width="100%"
                height="auto"
              />
            ) :
            dropzoneType === "video" &&
      file.type.includes("video") ? (
              <video
                src={URL.createObjectURL(file)}
                controls
                width="100%"
                height="auto"
              />
            ) : (
              <p>{file.name}</p>
            )}
            <EditOutlined />
          </FlexBetween>
        )}
      </Box>
```

Ispis 33: Dropzone slike, videozapisa i GIF-a

```
<FlexBetween gap="1rem">
          <IconButton
            onClick={() => handleDropzoneOpen("image")}
            sx={{ color: dropzoneType ===
"image" ? palette.primary.main : mediumMain }}
          >
            <ImageOutlined />
          </IconButton>
          <IconButton
            onClick={() => handleDropzoneOpen("video")}
            sx={{ color: dropzoneType ===
"video" ? palette.primary.main : mediumMain }}
          >
            <VideoLibraryOutlined />
          </IconButton>
          <IconButton
            onClick={() => handleDropzoneOpen("gif")}
            sx={{ color: dropzoneType ===
"gif" ? palette.primary.main : mediumMain }}
          >
            <GifOutlined />
          </IconButton>
        </FlexBetween>
      </FlexBetween>
```

Ispis 34: Dugme ikone slike, videa i GIF-a

4.2.9 Filter objava na osnovu izabranog tima

Ispis 35 definira funkciju pod nazivom *handleFilterByTeam*, koja je odgovorna za filtriranje objava na temelju odabranog tima. Ako je odabrani tim "Sve objave", dohvaća sve objave pomoću funkcije *getPosts*. U suprotnom, šalje zahtjev poslužitelju da dohvati sve objave, a zatim filtrira te objave kako bi uključio samo one koji odgovaraju odabranom timu. Filtrirani postovi se poništavaju i šalju radi ažuriranja stanja pomoću radnje *setPosts*. Ako se tijekom procesa pojave bilo kakve pogreške, poruka o pogrešci ispisuje se u konzoli.

```
const handleFilterByTeam = async (selectedTeam) => {
    if (selectedTeam === "All Posts") {

      await getPosts();
    } else {
      try {
        const response = await fetch(`http://localhost:3001/posts`, {
```

```
          method: "GET",
          headers: { Authorization: `Bearer ${token}` },
        });
        const data = await response.json();

        const filteredPosts = data.filter((post) => post.team ===
selectedTeam);

        dispatch(setPosts({ posts: filteredPosts.reverse() }));
      } catch (error) {
        console.error("Error fetching posts:", error);
      }
    }
  };
```

Ispis 35: Filter objava na osnovu izabranog tima

4.2.10 Metoda komentiranja pozivom *backend* API-ja

Ispis 36 definira funkciju pod nazivom *submitComment* koja upravlja slanjem komentara na post. Priprema podatke o komentarima, uključujući tekst komentara, ime i prezime korisnika. Zatim šalje zahtjev *PATCH* krajnjoj točki poslužitelja koji odgovara navedenom *postId*-u za dodavanje komentara. Zahtjev uključuje korisnički token za provjeru autentičnosti i podatke komentara u JSON formatu. Ako je odgovor uspješan (kod statusa 200), ažurirani post s novim komentarom dohvaća se iz odgovora i šalje za ažuriranje stanja pomoću radnje *setPost*. Konačno, unos komentara briše se za novi komentar.

```
const submitComment = async () => {
    const commentData = {
      comment: commentInput,
      firstName: firstName,
      lastName: lastName,
    };

    const response = await
fetch(`http://localhost:3001/posts/${postId}/comment`, {
      method: "PATCH",
      headers: {
        Authorization: `Bearer ${token}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify(commentData),
```

```
    });

    if (response.ok) {
      const updatedPost = await response.json();
      dispatch(setPost({ post: updatedPost }));
      setCommentInput("");
    }
  };
```

Ispis 36: Metoda komentiranja pozivom backend API-ja

4.2.11 Metoda brisanja objava pozivom *backend* API-ja

Ispis 37 definira funkciju deletePost odgovornu za brisanje objave. Provjerava ima li trenutni korisnik ovlasti za brisanje objave na temelju svog ID-a i statusa administratora. Šalje DELETE zahtjev poslužitelju s odgovarajućom autorizacijom. Ako je uspješan, ažurira popis postova u skladu s tim. Dodano je rukovanje pogreškama pomoću bloka try-catch za rješavanje potencijalnih pogrešaka tijekom procesa dohvaćanja i brisanja. U slučaju pogreške, bilježi detalje u konzolu. Za neovlašteni pristup, bilježi se poruka "Pristup odbijen". To poboljšava pouzdanost operacije brisanja elegantnim rukovanjem pogreškama i neovlaštenim radnjama.

```
const deletePost = async () => {
    if (loggedInUserId === postUserId || isAdmin) {
      try {
        const response = await
fetch(`http://localhost:3001/posts/${postId}`, {
          method: "DELETE",
          headers: {
            Authorization: `Bearer ${token}`,
          },
        });

        if (response.ok) {
          dispatch(setPosts({ posts: [] }));
          if (isProfile) {
            getUserPosts();
          } else {
            getPosts();
          }
        } else {
          throw new Error("Error deleting post.");
```

```
      }
    } catch (error) {
      console.error("Error deleting post:", error);
    }
  } else {
    console.error("Access denied, cant delete post.");
  }
};
```

Ispis 37: Metoda brisanja objava pozivom backend API-ja

4.2.12 Verzije korištenih biblioteka za *frontend*

U datoteci *package.json* nalazi se popis i verzije svih vanjskih *frontend* biblioteka koje se koriste (ispis 38). Najbolja praksa je uvijek imati zadnje stabilne verzije korištenih biblioteka.

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.11.16",
    "@mui/material": "^5.13.4",
    "@reduxjs/toolkit": "^1.9.5",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "dotenv": "^16.1.4",
    "formik": "^2.4.1",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-dropzone": "^14.2.3",
    "react-redux": "^8.0.7",
    "react-router-dom": "^6.12.1",
    "react-scripts": "5.0.1",
    "redux-persist": "^6.0.0",
    "web-vitals": "^2.1.4",
    "yup": "^1.2.0"
  },
```

Ispis 38: Korištene vanjske biblioteke u frontend kôdu

5 ZAKLJUČAK

Socijalne mreže danas su jedne od najkorištenijih web aplikacija na svijetu. Iako Meta i Google drže monopol u području javnih socijalnih mreža, kod privatnih socijalnih mreža nema tako istaknutih kompanija. *Techbook* je privatna socijalna mreža namijenjena tvrtkama koje žele unaprijediti povezanost i motiviranost među svojim zaposlenicima. Bitno je istaknuti da je web aplikacija izrađena tako da podržava dodatnu nadogradnju gdje developeri mogu implementirati svoje nove ideje u postojeći sustav.

Korištene su zadnje verzije navedenih tehnologija te je potrebno upoznati se sa svim tehnologijama prije nastavka daljnjeg razvoja. Kroz rad su opisane i prikazane sve funkcionalnosti aplikacije te se svaka od njih može mijenjati ako tvrtka to zahtijeva. UML dijagrami i faze razvoja detaljno opisuju razvoj aplikacije. Agilni način rada koji kreće od planiranja i dizajna, sve do implementacije i testiranja je glavni oslonac uspješnosti ove web aplikacije.

Implementacija novih ideja je više nego moguća i dobrodošla. *Chat* između korisnika, video i glasovni razgovori, korisnička ploča, administratorska ploča, dijeljenje objava drugih korisnika na svom profilu, implementacija *emojia* samo su neke od ideja koje mogu biti implementirane u ovu web aplikaciju. Kroz sam razvoj naišao sam na par problema s *Reactom* koji bi davao poruke o pogrešci za samo neke korisnike ili objave. Rješenje je bilo brisanje starih korisnika i objava koji nisu bili ažurirani posljednjim promjenama u *mongoose* modelu. U fazi planiranja ideja o dodavanju priloga unutar objave je odbačena iz razloga što se prilozi na stranici ne mogu prikazati u vizualnom obliku te je bolje da se ta ideja ostavi za kasniju implementaciju skupa sa *chat* funkcionalnosti.

6 LITERATURA

[1] *React documentation*, dohvaćeno iz https://devdocs.io/react/ , (posjećeno 20.8.2023.)

[2] *Mongoose documentation*, dohvaćeno iz https://mongoosejs.com/docs/ , (posjećeno 22.8.2023.)

[3] *Node.js documentation*, dohvaćeno iz https://nodejs.org/api/all.html , (posjećeno 22.8.2023.)

[4] *Express.js documentation*, dohvaćeno iz https://devdocs.io/express/ , (posjećeno 22.8.2023.)

[5] *JWT introduction*, dohvaćeno iz https://jwt.io/introduction , (posjećeno 24.8.2023.)

[6] *A Pragmatic Guide To Agile Ways Of Working*, dohvaćeno iz https://agilexl.com/blog/a-pragmatic-guide-to-agile-ways-of-working , (posjećeno 3.9.2023.)

[7] *Front-End vs. Back-End: The Complete Guide*, dohvaćeno iz https://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend , (posjećeno 28.8.2023.)