IZRADA WEB APLIKACIJE "DRUŠTVENA MREŽA"

Celić, Antonio

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

ANTONIO CELIĆ

Z A V R Š N I R A D

Izrada web aplikacije „Društvena mreža“

Split, rujan 2024.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

Predmet: Izrada web aplikacija

Z A V R Š N I R A D

Kandidat: Antonio Celić

Naslov rada: Izrada web aplikacije „Društvena mreža“

Mentor: Marina Rodić, viši predavač

Split, rujan 2024.

Sadržaj

Sažetak

Ovaj završni rad bavi se izradom web aplikacije „Društvena mreža“ koja korisnicima nudi interaktivno iskustvo te omogućuje dijeljenje sadržaja u stvarnom vremenu. Završni rad je realiziran korištenjem modernih tehnologija kao što su ReactJS i Typescript za stranu klijenta (engl. *frontend*), te Express.js i MySQL (Structured Query Language) za stranu poslužitelja (engl. *backend*). Aplikacija omogućuje korisnicima registraciju, uređivanje profila, kreiranje i interakciju s objavama kroz `sviđa mi se`, `ne sviđa mi se` i komentare. Implementacija WebSocket tehnologije omogućava praćenje komentara na objavama u stvarnom vremenu. Dodatno, sustav upravljanja ulogama osigurava različite razine pristupa i mogućnosti za registrirane korisnike, administratore i superkorisnike. Sigurnost aplikacije osigurana je hashiranjem lozinki, tokena za potvrdu identiteta i autorizaciju te validacijom korisničkih unosa na strani klijenta i poslužitelja.

Ključne riječi: autentifikacija, hashiranje, društvena mreža, sigurnost, WebSocket

Summary

“Social Network” application development

This final thesis focuses on the development of social network that offers users an interactive experience and enables real-time content sharing. The project was realized using modern technologies such as ReactJS and Typescript for the frontend, and Express.js and MySQL for the backend. The application allows users to register, edit their profiles, create posts, and interact with posts through likes, dislikes, and comments. The implementation of WebSocket technology enables real-time tracking of comments on posts. Additionally, the role management system ensures different levels of access and capabilities for registered users, administrators, and superusers. The security of the application is ensured by using password encryption, tokens for authentication and authorization, and validation of user inputs on both the frontend and backend.

Keywords: authentication, hashing, security, social network, WebSocket

1. Uvod

U današnjem digitalnom dobu, društvene mreže postale su neizostavan dio svakodnevnog života, pružajući platformu za komunikaciju, dijeljenje sadržaja i povezivanje korisnika širom svijeta. Međutim, unatoč brojnim postojećim rješenjima, postoji kontinuirana potreba za platformama koje nude poboljšanu interaktivnost, sigurnost i prilagodljivost specifičnim potrebama korisnika.

Ovaj završni rad fokusira se na razvoj društvene mreže koja nastoji odgovoriti na navedene izazove pružajući korisnicima bogato interaktivno iskustvo uz mogućnost dijeljenja sadržaja u stvarnom vremenu. Motivacija za ovaj završni rad proizašla je iz potrebe za platformom koja ne samo da omogućuje standardne funkcionalnosti poput registracije, uređivanja profila te objavljivanja sadržaja, već i integrira napredne funkcionalnosti kao što su praćenje interakcija u stvarnom vremenu putem WebSocket tehnologije.

Prilikom razvoja aplikacije, korištene su moderne tehnologije kako bi se osigurala visoka razina funkcionalnosti i korisničkog iskustva. Na strani klijenta, implementirani su ReactJS i TypeScript, omogućujući modularnost i tipizaciju kôda, dok je MaterialUI korišten za dosljedan i estetski ugodan dizajn sučelja. Za upravljanje stanjima unutar aplikacije korišten je Redux, dok su Formik te i18n & react-i18next osigurali validaciju unosa i internacionalizaciju aplikacije.

Na strani poslužitelja, Express.js je poslužio kao okvir za razvoj, uz sustav za upravljanje bazama podataka za pohranu informacija. Sigurnosni aspekti aplikacije naglašeni su korištenjem Bcrypt biblioteke za hashiranje lozinki te implementacijom tokena za autentifikaciju i autorizaciju korisnika. Validacija podataka osigurana je kroz Express Validator, dok je WebSocket tehnologija omogućila interakciju u stvarnom vremenu, poput praćenja komentara na objavama.

Glavni doprinos ovog rada ogleda se u integraciji navedenih tehnologija kako bi se stvorila platforma koja pruža visok stupanj interaktivnosti uz naglasak na sigurnost i prilagodljivost korisničkim potrebama. Posebna pažnja posvećena je implementaciji sustava

upravljanja ulogama, čime su definirane različite razine pristupa i ovlasti za registrirane korisnike, administratore i superkorisnike.

Struktura rada organizirana je na sljedeći način. U drugom poglavlju detaljno su opisane korištene tehnologije i alati, pružajući uvid u njihovu funkcionalnost i način integracije unutar aplikacije. Treće poglavlje pružit će uvid u postavljanje baze podataka i način pohrane podataka sa društvene mreže. U četvrtom poglavlju opisat će samu izradu društvene mreže i njenog sučelja. U zadnjem poglavlju, kroz zaključak, opisat će se postignuti rezultati rada, analizirati uspješnost završnog rada i naglasiti ključne aspekte koji su doprinijeli ostvarenju ciljeva, kao i potencijalne smjernice za budući razvoj.

2. Korištene tehnologije

U razvoju web aplikacije `4mat`, društvene mreže koja korisnicima nudi interaktivno iskustvo i omogućuje dijeljenje sadržaja u stvarnom vremenu, korišten je niz suvremenih tehnologija. Ovdje su detaljno opisane ključne tehnologije korištene tijekom razvoja aplikacije.

2.1. Visual Studio Code

Visual Studio Code (VS Code) je besplatan, moćan i prilagodljiv uređivač kôda razvijen od strane Microsofta. Koristi se široko u industriji zbog svoje brzine, jednostavnog sučelja i podrške za veliki broj programskih jezika. VS Code nudi pametno dovršavanje kôda, debuggiranje, integrirani terminal te ugrađenu podršku za Git, čime olakšava upravljanje verzijama kôda [1]. Također, podržava ekstenzije koje omogućuju dodatne funkcionalnosti, prilagođavajući uređivač specifičnim potrebama završnog rada [1]. Zbog ovih karakteristika, Visual Studio Code bio je ključan alat u razvoju aplikacije 4mat, osiguravajući efikasan i organiziran radni proces.

2.2. Strana klijenta

2.2.2. ReactJS

ReactJS je popularna JavaScript knjižnica razvijena od strane Facebooka, namijenjena izgradnji dinamičnih i interaktivnih korisničkih sučelja. Njegova glavna prednost je komponentno bazirani pristup, koji omogućava ponovnu upotrebu kôda i jednostavnije održavanje aplikacije. React koristi virtualni DOM (Document Object Model), što znatno poboljšava performanse aplikacije, omogućujući brza i efikasna ažuriranja sučelja. Jednosmjerni tijek podataka olakšava praćenje i upravljanje stanjem aplikacije [2]. ReactJS je odabran za razvoj `4mat` aplikacije zbog svoje brzine, fleksibilnosti i široke podrške unutar zajednice programera, što je omogućilo brži i organiziraniji razvoj.

2.2.3. Typescript

TypeScript je programski jezik razvijen od strane Microsofta, koji nadograđuje JavaScript dodavanjem statičke tipizacije. Ovo znači da TypeScript omogućava provjeru tipova tijekom razvoja, što pomaže u otkrivanju grešaka prije nego što aplikacija bude pokrenuta [3]. Također, TypeScript omogućava bolju organizaciju kôda i olakšava suradnju među programerima kroz jasnije definirane strukture podataka. TypeScript je korišten u razvoju `4mat` aplikacije zbog svoje sposobnosti da poboljša kvalitetu kôda i smanji mogućnost grešaka. Statička tipizacija omogućila je jasniju strukturu i lakše održavanje kôda, što je bilo ključno za razvoj složene aplikacije poput društvene mreže. Korištenje TypeScript-a osiguralo je da razvojni proces bude efikasniji, s manje grešaka i boljim alatima za programere.

2.2.4. MaterialUI

MaterialUI je popularna knjižnica komponenti za React koja implementira smjernice dizajna poznate kao Material Design, koje je razvila tvrtka Google. Ova knjižnica omogućava brzu izradu estetski ugodnih i dosljednih korisničkih sučelja pomoću unaprijed dizajniranih komponenti koje su prilagodljive i lako integriraju u React aplikacije [4]. MaterialUI je odabran za izradu korisničkog sučelja aplikacije `4mat` zbog svoje jednostavne integracije s React-om i mogućnosti brzog razvoja sučelja visoke kvalitete. Korištenje unaprijed dizajniranih komponenti znatno je ubrzalo razvojni proces, omogućujući fokus na funkcionalnosti aplikacije umjesto na dizajn od nule. MaterialUI je također osigurao da sučelje bude estetski ugodno, dosljedno i prilagođeno različitim uređajima, što je bilo ključno za pružanje pozitivnog korisničkog iskustva.

2.2.5. Formik

Formik je popularna knjižnica za upravljanje obrascima u React aplikacijama, koja pojednostavljuje proces validacije unosa i rukovanje podacima iz forma. Formik omogućava jednostavno definiranje i validaciju forma, čime se smanjuje količina kôda potrebna za upravljanje složenim formama [5]. Formik je korišten u aplikaciji `4mat` kako bi se olakšalo upravljanje formama i validacijom korisničkih unosa. Njegova jednostavnost i fleksibilnost omogućili su brzo postavljanje forma, što je osiguralo kvalitetnu i pouzdanu interakciju korisnika s aplikacijom, uz smanjenje mogućnosti pogrešaka.

2.2.6. Paketi i18n i react-i18next

Alati i18n (internacionalizacija) i react-i18next su alati koji omogućuju jednostavnu podršku za više jezika u React aplikacijama. Alat i18n je standard za upravljanje prijevodima i lokalizacijom, dok je react-i18next popularna knjižnica koja integrira ovaj standard s React-om. [6]. Alati i18n i react-i18next su korišteni u aplikaciji `4mat` kako bi se omogućila podrška za više jezika, uključujući hrvatski i engleski. Ovi alati omogućili su lako upravljanje prijevodima i lokalizacijom, čime je aplikacija postala pristupačnija korisnicima iz različitih jezičnih područja, bez komplikacija u implementaciji.

2.2.7. Paket react-router-dom

Knjižnica react-router-dom omogućuje jednostavno upravljanje navigacijom i rutama unutar jednostraničnih stranica (SPA - Single Page Applications). Omogućava definiranje različitih ruta za različite komponente i olakšava navigaciju kroz aplikaciju bez ponovnog učitavanja stranice [7]. Knjižnica react-router-dom je korišten u aplikaciji `4mat` za jednostavno upravljanje navigacijom između različitih stranica i dijelova aplikacije. Ova knjižnica omogućila je dinamično i intuitivno korisničko iskustvo, olakšavajući navigaciju kroz različite funkcionalnosti aplikacije bez potrebe za ponovnim učitavanjem stranice.

2.2.8. Redux

Redux je popularna knjižnica za upravljanje globalnim stanjem aplikacije u React-u. Omogućuje centralizirano pohranjivanje i upravljanje stanjem, što olakšava praćenje promjena stanja i dijeljenje podataka između različitih komponenti unutar aplikacije [8]. Redux je korišten u aplikaciji `4mat` za efikasno upravljanje globalnim stanjem aplikacije, što je omogućilo dosljedno i pouzdano dijeljenje podataka između različitih komponenti. Centralizirano upravljanje stanjem olakšalo je praćenje promjena i omogućilo jednostavniju implementaciju složenih funkcionalnosti unutar aplikacije.

2.3. Strana poslužitelja

2.3.1. Express.js

Express.js je lagani i fleksibilni web okvir za Node.js, koji olakšava izradu web aplikacija i API-a. Pruža jednostavan način za postavljanje ruta, rukovanje HTTP zahtjevima i upravljanje međuprograma (engl. *middleware*), što ga čini idealnim za razvoj poslužiteljskih aplikacija aplikacija. Express.js je korišten u aplikaciji `4mat` kao okvir poslužitelja zbog svoje jednostavnosti i brzine. Njegova fleksibilnost omogućila je učinkovito rukovanje HTTP zahtjevima i integraciju s bazom podataka, što je bilo ključno za implementaciju logike poslužiteljske aplikacije [9].

2.3.2. MySQL

MySQL je popularan sustav za upravljanje relacijskim bazama podataka (RDBMS) koji se koristi za pohranu i upravljanje podacima u aplikacijama. Razvijen od strane Oracle Corporation, MySQL je poznat po svojoj brzini, pouzdanosti i jednostavnosti korištenja. Podržava SQL (Structured Query Language) za definiranje, manipulaciju i upravljanje podacima. MySQL je odabran za bazu podataka aplikacije `4mat` zbog svoje sposobnosti rukovanja velikim količinama podataka, visokih performansi i široke podrške unutar zajednice programera [10]. Također, njegova integracija s Express.js olakšala je razvoj i održavanje aplikacije.

2.3.3. Express Validator

Express Validator je popularna knjižnica za validaciju podataka unutar Express.js aplikacija. Omogućuje jednostavnu provjeru korisničkih unosa kako bi se osiguralo da podaci ispunjavaju određene uvjete prije obrade ili pohrane u bazu podataka. Express Validator je korišten u aplikaciji `4mat` za osiguravanje ispravnosti i sigurnosti podataka, smanjujući rizik od pogrešaka i potencijalnih sigurnosnih prijetnji kroz validaciju na serverskoj strani [11].

2.3.4. Bcrypt

Bcrypt je sigurnosna knjižnica za šifriranje lozinki koja se široko koristi u web aplikacijama za zaštitu korisničkih podataka. Korištenjem složenih algoritama, Bcrypt omogućuje hashiranje lozinki na način koji je otporan na brute-force napade. U aplikaciji `4mat`, Bcrypt je korišten za sigurno pohranjivanje lozinki korisnika, osiguravajući da su korisnički podaci zaštićeni čak i u slučaju neovlaštenog pristupa bazi podataka [12].

2.3.5. WebSocket

WebSocket je protokol za dvosmjernu komunikaciju u stvarnom vremenu između klijenta i servera preko jedne TCP veze. Za razliku od klasičnih HTTP zahtjeva, WebSocket omogućuje stalnu vezu koja omogućava trenutačno slanje i primanje podataka. U aplikaciji `4mat`, WebSocket je korišten za omogućavanje trenutnih ažuriranja komentara pružajući korisnicima brže i interaktivnije iskustvo [13].

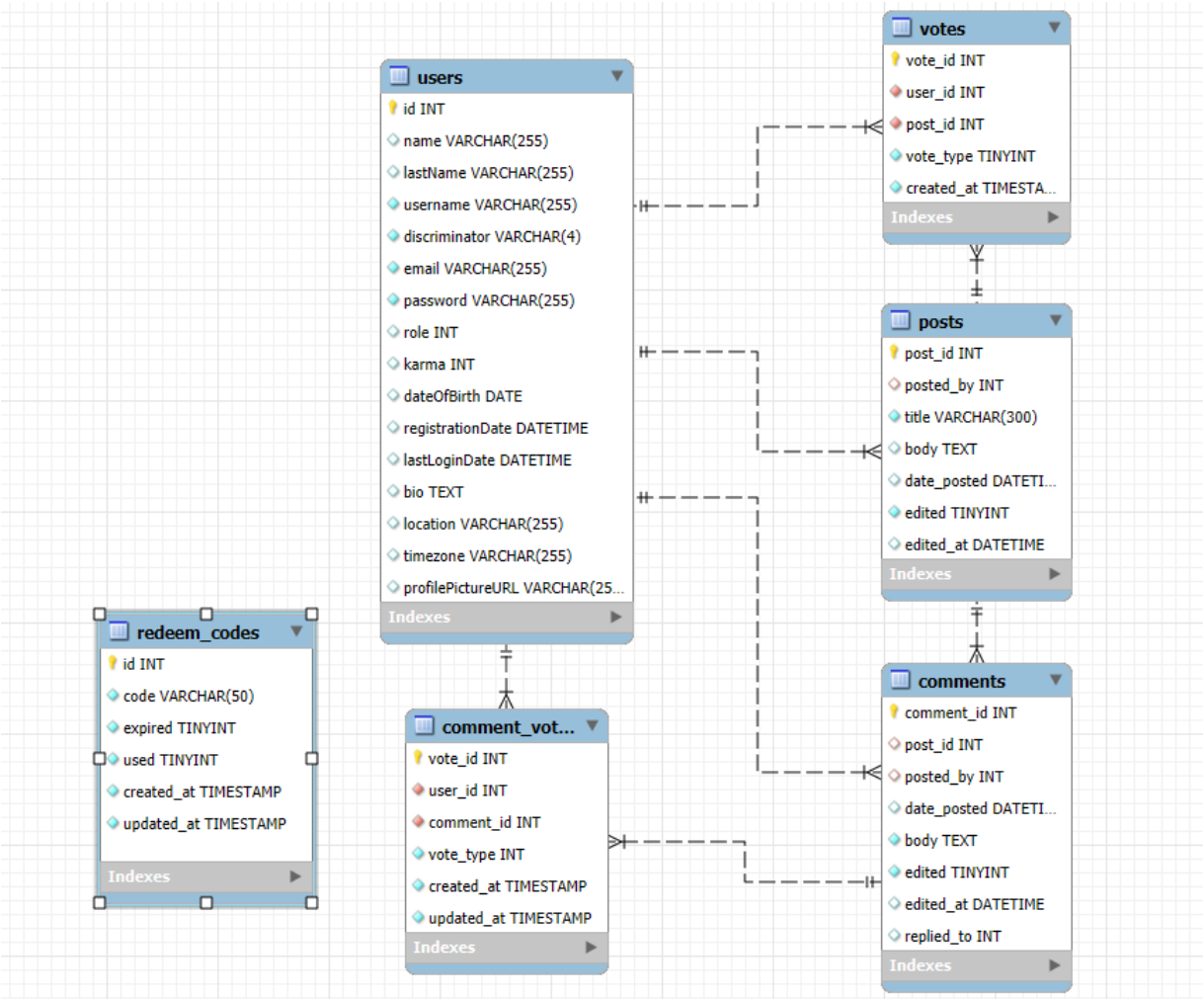
3. Baza podataka

U razvoju web aplikacije 4mat korištena je relacijska baza podataka MySQL, koja služi za pohranu i upravljanje podacima korisnika, objavama, interakcijama i ulogama unutar aplikacije. Relacijske baze podataka kao MySQL koriste tablice za organizaciju podataka, gdje su tablice povezane kroz strane ključeve, omogućujući učinkovito pohranjivanje i pretraživanje informacija.

MySQL je odabran zbog svoje brzine, pouzdanosti i široke podrške u zajednici programera. Omogućava učinkovito rukovanje velikim količinama podataka i pruža mogućnosti kao što su skalabilnost i sigurnost, što je ključno za aplikacije koje se brzo razvijaju i rastu, poput društvenih mreža.

3.1. Entiteti

Entiteti u bazi podataka predstavljaju realne objekte ili pojmove iz stvarnog svijeta koji imaju svoje atribute i odnose s drugim entitetima. U kontekstu aplikacije `4mat`, entiteti su osnovne jedinice podataka koje se pohranjuju u bazu podataka i koje modeliraju ključne aspekte funkcionalnosti aplikacije. Relacije između entiteta se mogu vidjeti na Slici 1.



Slika 1: ER dijagram baze podataka za aplikaciju `4mat`

3.1.1. Entitet User

Entitet `User` (korisnik) predstavlja osnovni entitet u bazi podataka aplikacije 4mat, putem kojeg se pohranjuju informacije o svim registriranim korisnicima. Svaki korisnik ima jedinstveni identifikator (skraćeno id) i osnovne informacije poput imena, prezimena, korisničkog imena (engl. *username*), email adrese, i lozinke. Također, entitet bilježi korisničke aktivnosti, kao što su datum registracije i posljednja prijava, te dodatne informacije poput biografije, lokacije, vremenske zone i URL-a za profilnu sliku. Na Ispisu 1. se mogu vidjeti sva polja za entitet `User`.

```
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255),
  lastName VARCHAR(255),
  username VARCHAR(255) NOT NULL,
  discriminator VARCHAR(4) NOT NULL,
  email VARCHAR(255) NOT NULL,
  password VARCHAR(255) NOT NULL,
  role INT DEFAULT 0,
  karma INT DEFAULT 0,
  dateOfBirth DATE,
  registrationDate DATETIME DEFAULT CURRENT_TIMESTAMP,
  lastLoginDate DATETIME,
  bio TEXT,
  location VARCHAR(255),
  timezone VARCHAR(255),
  profilePictureURL VARCHAR(255)
);
```

Ispis 1: Naredba za kreiranje tablice `users` sa definiranom strukturom

3.1.2. Entitet Post

Entitet `Post` (objava) predstavlja objave koje korisnici dijele unutar aplikacije `4mat`. Svaka objava ima jedinstveni identifikator (`post_id`) i povezana je s korisnikom koji ju je objavio putem polja `posted_by`. Entitet uključuje osnovne informacije o objavi, kao što su naslov (`title`), tijelo objave (`body`), datum objave (`date_posted`), te status izmjene (`edited` i `edited_at`). Putem ovog entiteta omogućena je pohrana i upravljanje s korisničkim sadržajem te integracija s drugim entitetima kao sto su komentari i glasovi. Na ispisu 2. mogu se vidjeti sva polja entiteta `Post`.

```
CREATE TABLE IF NOT EXISTS posts (
  post_id INT AUTO_INCREMENT PRIMARY KEY,
  posted_by INT,
  title VARCHAR(300) NOT NULL,
  body TEXT,
  date_posted DATETIME DEFAULT CURRENT_TIMESTAMP,
  edited BOOLEAN NOT NULL DEFAULT FALSE,
  edited_at DATETIME DEFAULT NULL,
  FOREIGN KEY (posted_by) REFERENCES users(id)
);
```

Ispis 2: Naredba za kreiranje tablice `posts` sa definiranom strukturom

3.1.3. Entitet Vote

Entitet `Vote` (glas) predstavlja glas koji korisnik daje na objavi na društvenoj mreži `4mat`. Ovaj entitet omogućuje praćenje korisničkih interakcija s objavama, bilo da podržavaju ili ne podržavaju određeni sadržaj. Ovaj entitet je ključan za omogućavanje funkcionalnosti ocjenjivanja sadržaja i pridonosi interaktivnosti korisničkog iskustva na platformi. Na ispisu 3. mogu se vidjeti sva polja o entitetu `Vote`.

```
CREATE TABLE IF NOT EXISTS votes (
  vote_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  post_id INT NOT NULL,
  vote_type TINYINT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users(id),
  CONSTRAINT fk_post FOREIGN KEY (post_id) REFERENCES posts(post_id)
);
```

Ispis 3: Naredba za kreiranje tablice `votes` sa definiranom strukturom

3.1.4. Entitet comment

Entitet `comment` (komentar) predstavlja komentare koje korisnici ostavljaju na postove. Sadrži jedinstveni identifikator, povezan je s određenim postom i korisnikom, te bilježi vrijeme objave, sadržaj komentara, i eventualne izmjene. Putem ovoga entiteta omogućeno je praćenje interakcija korisnika s objavama i doprinosi dinamičnosti unutar aplikacije. Na Ispisu 4. mogu se vidjeti sva polja entiteta `comment`.

```
CREATE TABLE IF NOT EXISTS comments (
  comment_id INT AUTO_INCREMENT PRIMARY KEY,
  post_id INT,
  posted_by INT,
  date_posted DATETIME DEFAULT CURRENT_TIMESTAMP,
  body TEXT NOT NULL,
  edited BOOLEAN NOT NULL DEFAULT FALSE,
  edited_at DATETIME DEFAULT NULL,
  FOREIGN KEY (post_id) REFERENCES posts(post_id),
  FOREIGN KEY (posted_by) REFERENCES users(id)
);
```

Ispis 4: Naredba za kreiranje tablice `comments` sa definiranom strukturom

3.1.5. Entitet Comment vote

Entitet `Comment vote` (glas na komentar) predstavlja glasove koje korisnici daju na komentare unutar aplikacije `4mat`. Svaki glas je povezan s određenim korisnikom i komentarom te bilježi vrstu glasa (pozitivan ili negativan) i vrijeme kada je glas dodijeljen. Ovaj entitet omogućava praćenje reakcija na komentare, dodajući sloj interaktivnosti u komunikaciji među korisnicima. Na ispisu 5. mogu se vidjeti sva polja entiteta `Comment vote`.

```
CREATE TABLE IF NOT EXISTS comment_votes (
  vote_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  comment_id INT NOT NULL,
  vote_type INT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
  FOREIGN KEY (comment_id) REFERENCES comments(comment_id) ON DELETE
CASCADE,
  CONSTRAINT unique_vote UNIQUE (user_id, comment_id)
);
```

Ispis 5: Naredba za kreiranje tablice `comment_votes` sa definiranom strukturom

4. Komponente aplikacije

Klijentski dio aplikacije 4mat odnosi se na korisničko sučelje i sve vizualne elemente s kojima korisnici izravno komuniciraju. Glavne tehnologije korištene na strani klijenta u razvoju su ReactJS i MaterialUI. ReactJS omogućava izgradnju dinamičnih i responzivnih sučelja putem komponentno baziranog pristupa, što olakšava upravljanje i ponovnu upotrebu kôda. MaterialUI osigurava dosljedan i estetski privlačan dizajn sučelja, s komponentama koje prate smjernice Googleovog Material Designa.

Klijentski dio aplikacije također koristi Formik za upravljanje obrascima i validaciju unosa, što je ključno za funkcionalnosti poput registracije i prijave korisnika. React Router omogućuje jednostavnu navigaciju unutar aplikacije bez potrebe za ponovnim učitavanjem stranice, dok Redux upravlja globalnim stanjem aplikacije, osiguravajući sinkronizaciju podataka između različitih komponenti.

Poslužiteljski dio aplikacije 4mat zadužen je za logiku aplikacije, obradu podataka, sigurnost i komunikaciju s bazom podataka. Izgrađen je koristeći Express.js, minimalan i fleksibilan Node.js okvir, koji omogućava brzo definiranje API ruta i upravljanje HTTP zahtjevima. Strana poslužitelja komunicira s MySQL bazom podataka, u kojoj su pohranjeni svi podaci o korisnicima, objavama, komentarima i glasovima.

Sigurnost aplikacije je osigurana putem bcrypt-a za hashiranje lozinki i Express Validatora za validaciju korisničkih unosa. Strana poslužitelja također koristi JWT (JSON Web Token) za autentifikaciju korisnika i zaštitu osjetljivih podataka. Za real-time komunikaciju, kao što su obavijesti ili ažuriranja, koristi se WebSocket tehnologija, koja omogućuje trenutnu interakciju korisnika unutar aplikacije.

4.1. Korisnički sustav aplikacije

Korisnički sustav aplikacije 4mat omogućuje korisnicima interakciju s aplikacijom kroz različite funkcionalnosti kao što su objavljivanje sadržaja, komentiranje, te glasanje. Korisnici mogu pregledavati sadržaj na aplikaciji bez potrebe za registracijom ili prijavom. Međutim, za kreiranje objava, komentiranje i glasanje, korisnici se moraju registrirati i prijaviti na svoje račune.

Registracija zahtijeva unos osnovnih podataka kao što su *redeem code*, korisničko ime, email adresa, datum rođenja, i lozinka, koja je sigurno pohranjena u bazi podataka pomoću bcrypt algoritma. Korisnici se također moraju složiti s uvjetima korištenja (engl. *terms of service*) tijekom procesa registracije. Nakon registracije, korisnici mogu pristupiti svom profilu i uređivati informacije poput biografije, lokacije i profilne slike.

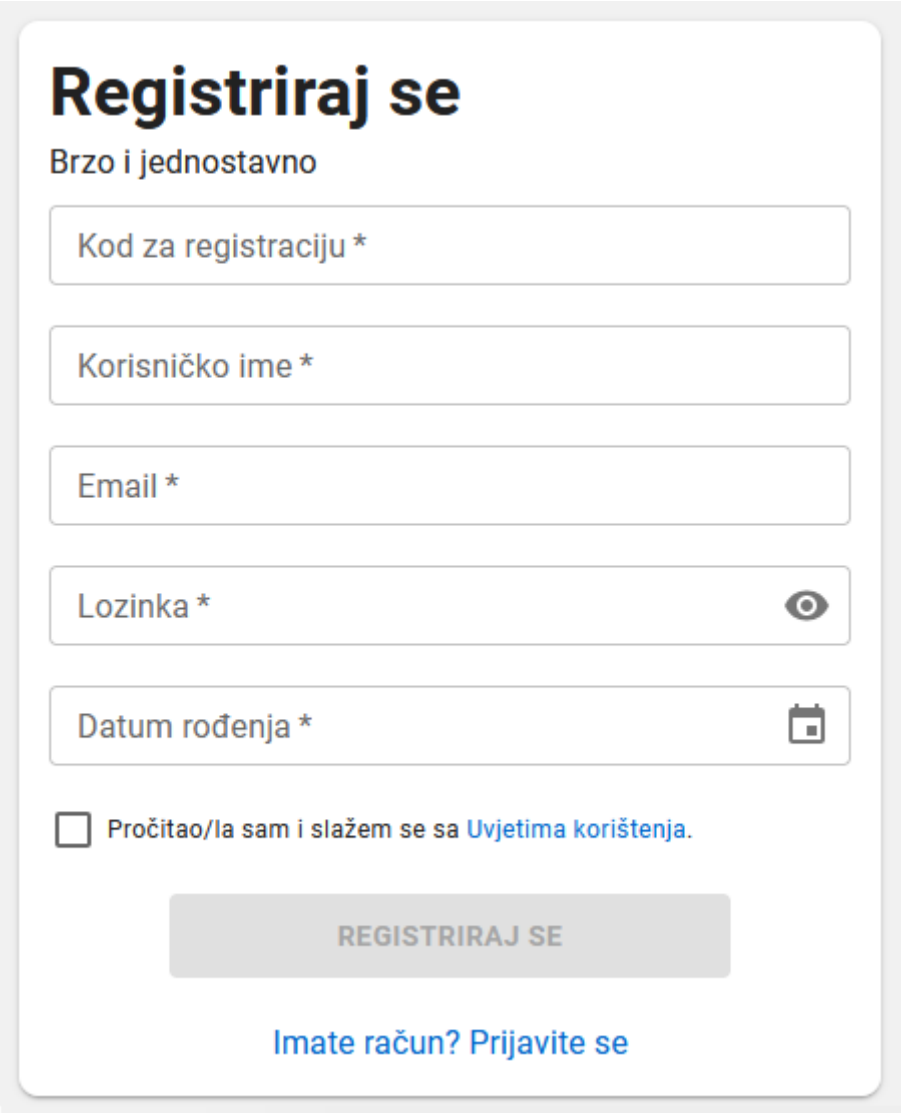
Prijava korisnika provodi se putem autentifikacije koristeći JSON Web Token (JWT), što omogućuje sigurnu provjeru identiteta tijekom cijele sesije. Token se generira prilikom prijave i koristi za ovjeru korisnika pri svakom zahtjevu prema strani poslužitelja aplikacije. Aplikacija koristi različite razine autorizacije kako bi osigurala da samo ovlašteni korisnici mogu pristupiti određenim funkcionalnostima, poput administracije ili moderiranja sadržaja.

U aplikaciji 4mat, prikupljanje podataka za registraciju korisnika na strani klijenta obavlja se pomoću knjižnice `Formik`, koja olakšava rad s obrascima, uključujući validaciju i upravljanje stanjem formi. Podaci uneseni u formu, uključujući *redeem code*, korisničko ime, email adresu, lozinku i datum rođenja, šalju se na poslužitelj putem knjižnice `Axios`.

Ako poslužitelj vrati negativan odgovor, korisniku se prikazuje odgovarajuća poruka s informacijom o pogrešci ili problemu s registracijom. Ako je odgovor pozitivan, korisnik se automatski preusmjerava na odgovarajuću stranicu, poput prijave ili glavnog sučelja aplikacije.

4.1.1. Registracija korisnika

Registracija korisnika u aplikaciji 4mat zahtijeva unos nekoliko osnovnih informacija, uključujući *redeem* kôd, korisničko ime, email adresu, datum rođenja i lozinku. Ovaj proces odvija se u dva ključna koraka: validacija unosa na strani klijenta te obrada podataka na strani poslužitelja. Na Slici 1. je prikazan forma za registraciju novog korisnika.



Slika 1: Forma za registraciju korisnika

Na strani klijenta aplikacije, korisnički unos validira se korištenjem knjižnice `Formik`, koja upravlja obrascima, i `Yup` za definiranje pravila validacije. Formik omogućuje jednostavno prikupljanje i validaciju korisničkih podataka prije nego što se pošalju na poslužitelj. Na ispisu 6. može se vidjeti validacija unosa za registraciju na strani klijenta.

```
const registerSchema = Yup.object().shape({
  redeemCode: Yup.string()
    .required('errors.required'),
  username: Yup.string()
    .required('errors.required')
    .min(4, 'errors.usernameShort')
    .max(20, 'errors.usernameLong')
    .matches(/^[a-z0-9]+$/, 'errors.usernameChars'),
  email: Yup.string()
    .required('errors.required')
    .email('errors.invalidEmail'),
  password: Yup.string()
    .required('errors.required')
    .min(8, 'errors.passwordShort')
    .matches(/[0-9]/, 'errors.passwordDigit')
    .matches(/[a-z]/, 'errors.passwordLowercase')
    .matches(/[A-Z]/, 'errors.passwordUppercase')
    .matches(/[@#$%^&+=!]/, 'errors.passwordSpecialCharacter'),
...
```

Ispis 6: Isječak kôda validacije unosa za registraciju na strani klijenta pomoću Yup-a

Ova validacija osigurava da svi uneseni podaci ispunjavaju specifične kriterije prije nego što se pošalju na poslužitelj, čime se sprječava slanje neispravnih ili nepotpunih podataka. Nakon uspješne validacije, podaci se šalju na poslužitelj putem `Axios` knjižnice što se može vidjeti na ispisu 7.

```
  const formik = useFormik({
    initialValues: {
      redeemCode: '',
      username: '',
      email: '',
      password: '',
      dateOfBirth: null
    },
    validationSchema: registerSchema,
    onSubmit: async (values) => {
      try {
        setLoading(true);
        // Validate redeem code
        const { redeemCode, ...registrationData } = values;
        const responseRedeem = await axiosInstance.post('/api/validate-
redeem-code', { redeemCode });

        // If redeem code is valid, proceed with registration
        console.log('Redeem code validated:', responseRedeem.data);

        // Make the registration request
        const response = await axiosInstance.post('/api/register',
registrationData);

        // Registration successful, you can handle the response here
        console.log(response.data.message);
        setSuccess('success.registerSuccess');
        // Introduce a delay before navigating to login
        setTimeout(() => {
          navigate('/login');
        }, 1000); // 1000 milliseconds = 1 seconds delay
        setError('');
...
```

Ispis 7: Isječak kôda za slanje podataka za registraciju sa klijenta na poslužitelj

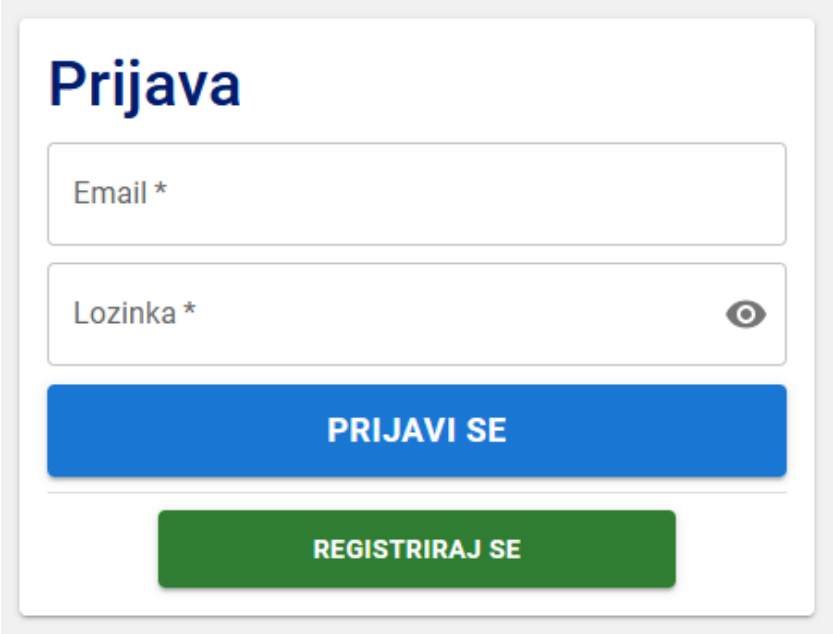
Strana poslužitelja aplikacije, razvijena korištenjem `Express.js` okvira, preuzima i obrađuje podatke dobivene s klijenta. Poslužitelj validira korisničke unose, provjerava postoje li korisnici s istom email adresom, te pohranjuje nove korisnike u bazu podataka. Korištenjem `Express Validator` alata, poslužitelj osigurava da su svi podaci ispravni prije nego što se kreira novi korisnički račun kao što se može vidjeti na ispisu 8.

```
app.post('/api/register', [
  check('username').isLength({ min: 4, max: 20}).matches(/^[a-z0-9]+$/),
  check('email').isEmail(),
  check('password').isLength({ min: 8 }).matches(/\d/).matches(/[A-Z]/),
  check('dateOfBirth').isISO8601().custom((value) => {
    const birthDate = new Date(value);
    const minDate = new Date();
    minDate.setFullYear(minDate.getFullYear() - 18);
    if (birthDate >= minDate) throw new Error('Must be 18 years or
older');
    return true;
  }),
], async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) return res.status(400).json({ message: 'Invalid
input' });

  try {
          const [results] = await pool.execute(
          'INSERT INTO users (username, email, password, dateOfBirth,
discriminator) VALUES (?, ?, ?, ?, ?)',
          [username, email, hashedPassword, formattedDate, discriminator]
        );
    res.status(201).json({ message: 'User registered successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Ispis 8: Isječak kôda na poslužitelju za obradu podataka poslanih sa klijenta

4.1.2. Prijava korisnika

Prijava korisnika u aplikaciju `4mat` omogućuje korisnicima pristup njihovim računima te korištenje personaliziranih funkcionalnosti aplikacije. Proces prijave uključuje unos adrese elektroničke pošte i lozinke te provjeru tih podataka na poslužitelju kako bi se osiguralo da su unosi ispravni i da korisnik postoji u sustavu. Na slici 2. može se vidjeti forma za prijavu korisnika.



Slika 2: Forma za prijavu korisnika

Na strani klijenta, prijava je implementirana pomoću `Formik` za upravljanje obrascima te `Yup` za validaciju unosa. Korisnički unos validira se kako bi se osiguralo da su email adresa i lozinka pravilno uneseni prije slanja na poslužitelj kao što se vidi na ispisu 9.

```
const loginSchema = Yup.object().shape({
  email: Yup.string()
    .required('errors.required')
    .email('errors.invalidEmail'),
  password: Yup.string()
    .required('errors.required')
});
```

Ispis 9: Kôd za validaciju unosa polja za prijavu

Nakon što korisnik unese email i lozinku te pošalje formu, podaci se šalju na poslužiteljsku stranu putem `Axios` biblioteke. Ako je prijava uspješna, korisnički `JWT token` pohranjuje se u *localStorage*, a korisnik se preusmjerava na početnu stranicu.

Na strani poslužitelja, podaci se primaju i obrađuju putem `Express.js` poslužitelja. Provjerava se postojanje korisnika u bazi podataka na temelju email adrese. Ako korisnik postoji, uspoređuje se unesena lozinka s onom pohranjenom u bazi podataka pomoću `bcrypt` biblioteke. Ako su podaci ispravni, strana poslužitelja generira `JWT token` s korisničkim podacima, uključujući ulogu korisnika, i vraća ga na stranu klijenta. Token omogućava autorizaciju i pristup zaštićenim resursima unutar aplikacije. Logika autentifikacije korisnika može se vidjeti na ispisu 10.

```
app.post('/api/login', loginRegisterLimiter, async (req, res) => {
  try {
    const { email, password } = req.body;
    const [users] = await pool.execute('SELECT * FROM users WHERE email =
?', [email]);
    const usersArray = users as mysql.RowDataPacket[];

    if (usersArray.length === 0) {
      return res.status(404).json({ message: 'User not found' });
    }

    const user = usersArray[0];
    const isPasswordValid = await bcrypt.compare(password, user.password);

    if (!isPasswordValid) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    const tokenPayload = {
      sub: user.id,
      role: user.role
    };

    const token = jwt.sign(tokenPayload, env.jwtSecretKey, { expiresIn:
'1h' });

    res.status(200).json({ message: 'Login successful', token: token,
user: limitedUser });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Ispis 10: Isječak kôda za prijavu korisnika na strani poslužitelja

Ovaj isječak prikazuje ključne korake u procesu prijave korisnika, uključujući validaciju podataka, generiranje tokena i vraćanje odgovora s potrebnim informacijama za stranu klijenta. Time se osigurava sigurna autentifikacija korisnika i pristup njihovom računu.

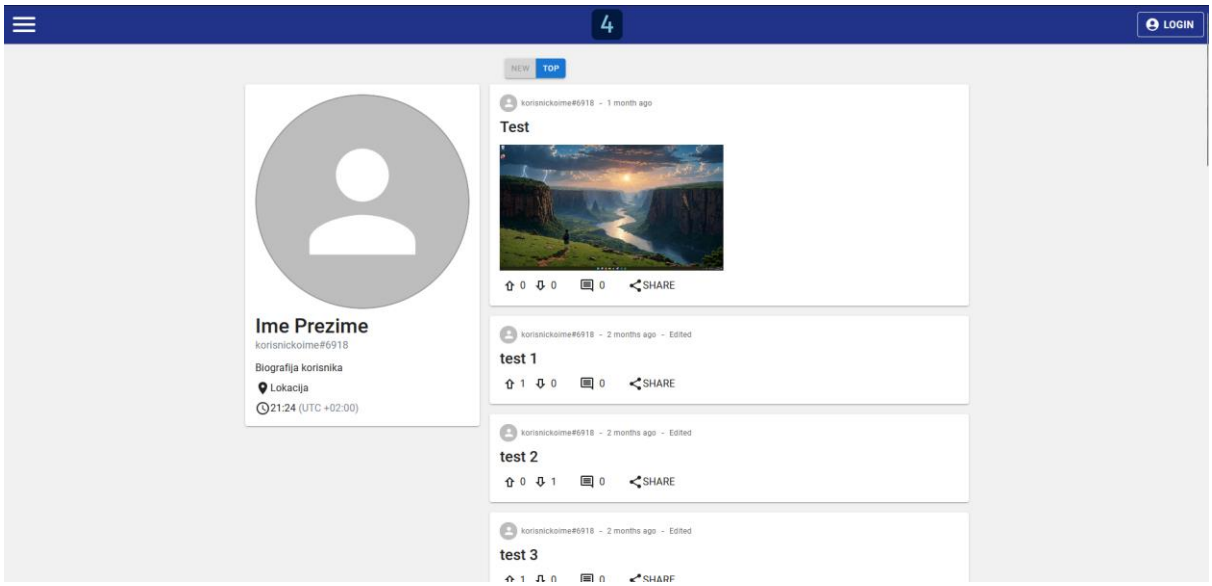
4.1.3. Korisnički profil

Korisnički profil unutar aplikacije 4mat predstavlja središnje mjesto gdje su prikazane ključne informacije o korisniku. Svaki korisnik ima jedinstven profil, kojem se pristupa putem URL-a u formatu `localhost/user/username#discriminator`. Ovaj URL koristi kombinaciju korisničkog imena i četveroznamenkastog diskriminatora kako bi se osiguralo da svaki korisnik ima jedinstvenu URL adresu.

Korisnički profili su javno dostupni i mogu ih pregledavati svi korisnici aplikacije. Međutim, samo vlasnik profila ima pravo na uređivanje svojih podataka. Uređivanje profila omogućava korisniku da ažurira svoje osobne informacije poput `imena`, `prezimena`, `biografije (Bio)`, `lokacije` i `vremenske zone`. Također, korisnik ima mogućnost promjene svog korisničkog imena, što može rezultirati promjenom diskriminatora kako bi se osigurala jedinstvenost unosa. Primjer korisničkog profila se može vidjeti na slici 3.

Ukoliko korisnik nije unio sve podatke, prikazat će se samo korisničko ime i diskriminator, dok će ostala polja biti prazna. Profilna slika se također može postaviti ili promijeniti, a ukoliko nije postavljena, prikazat će se prazno polje za profilnu sliku.

Uz to, na profilu se prikazuju sve objave korisnika koje se mogu sortirati po `najnovijim` ili `najbolje ocijenjenim` (tj. objave s najviše lajkova).



Slika 3: Primjer korisničkog profila

Korisnički profil i sve relevantne informacije dohvaćaju se putem API-a sa poslužiteljske strane kao što se može vidjeti na ispisu 11.

```
app.get('/api/user/profile', async (req, res) => {
  try {
    const { username, discriminator } = req.query;
    const [rows] = await pool.execute(
      'SELECT id, name, lastName, username, discriminator,
registrationDate, bio, location, timezone, profilePictureURL FROM users
WHERE username = ? AND discriminator = ?',
      [username, discriminator]
    );
    const userRows = rows as mysql.RowDataPacket[];
    if (userRows.length === 0) {
      return res.status(404).json({ message: 'User not found' });
    }
    const user = userRows[0];
    const userResponse = {
      id: user.id,
      name: user.name,
      lastName: user.lastName,
      username: user.username,
      discriminator: user.discriminator,
      registrationDate: user.registrationDate,
      bio: user.bio,
      location: user.location,
      timezone: user.timezone,
      profilePictureURL: user.profilePictureURL
    };
    res.json(userResponse);
  } catch (error) {
    console.error('Error fetching user profile data:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Ispis 11: Isječak kôda koji dohvaća profil korisnika

Korisnici mogu ažurirati svoje profile koristeći relevantne web API-e, a JWT token se koristi za potvrdu prava za korištenjem resursa. Isto tako, na poslužiteljskoj strani validiraju se sve unesene informacije. Na ispisu 12. može se vidjeti kôd za ažuriranje profila.

```
app.post('/api/user/update', passport.authenticate('jwt', { session: false
}), async (req, res) => {
  try {
    const user = req.user as IUser;
    if (user.id !== req.body.userID) return res.status(403).json({ message:
'Access denied.' });

    // Handle username and discriminator update
    let { username } = req.body;
    let discriminator = user.discriminator;
    if (username !== user.username) {
      while (true) {
        const [existingUsers] = await pool.execute('SELECT * FROM users
WHERE username = ? AND discriminator = ?', [username, discriminator]);
        if (existingUsers.length === 0) break;
        discriminator = String(Math.floor(Math.random() * 9999) +
1).padStart(4, '0');
      }
    }

    // Prepare updated values
    const updateValues = {
      name: req.body.name?.trim() || null,
      lastName: req.body.lastName?.trim() || null,
      bio: req.body.bio?.trim() || null,
      location: req.body.location?.trim() || null,
      timezone: req.body.timezone?.trim() || null
    };

    // Update profile
    await pool.execute(`
      UPDATE users SET name = ?, lastName = ?, username = ?, discriminator
= ?, bio = ?, location = ?, timezone = ? WHERE id = ?;
```

```
    `, [
      updateValues.name,
      updateValues.lastName,
      username,
      discriminator,
      updateValues.bio,
      updateValues.location,
      updateValues.timezone,
      req.body.userID
    ]);

    // Return updated user info
    const [updatedUsers] = await pool.execute('SELECT name, lastName,
username, discriminator, bio, location, timezone FROM users WHERE id = ?',
[req.body.userID]);
    const updatedUser = updatedUsers[0];
    if (!updatedUser) return res.status(404).json({ message: 'User not
found' });

    res.status(200).json({ message: 'Profile updated successfully', user:
updatedUser });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Ispis 12: Isječak kôda za ažuriranje profila

Korisnici mogu promijeniti svoju profilnu sliku pomoću API rute koja prihvaća slike i obrađuje ih na poslužiteljskoj strani. Slike se komprimiraju i pohranjuju na poslužitelj, a njihov URL se ažurira u korisničkom profilu.

Prvo, za postavljanje slika koristi se `Multer`, međuprogram za upravljanje multipart/form-data. Konfiguracija `Multera` uključuje pohranu slika u memoriji, ograničenje veličine datoteke na 10 MB i filtriranje datoteka kako bi se omogućio samo JPEG format i to se može vidjeti na ispisu 13.

```
const storage = multer.memoryStorage();
const upload = multer({
  storage: storage,
  limits: {
    fileSize: 10 * 1024 * 1024, // 10MB limit
  },
  fileFilter: (req, file, cb) => {
    if (!file.originalname.match(/\.(jpg|jpeg)$/)) {
      return cb(new Error('Only JPEG images are allowed'));
    }
    cb(null, true);
  },
});
```

Ispis 13: Konfiguriranje paketa `Mutler`

Nakon što se slika učita u memoriju, koristi se biblioteka Sharp za njenu kompresiju. Slika se smanjuje na dimenzije 400x400 piksela i komprimira se u JPEG format s kvalitetom od 80% kao što se vidi na ispisu 14.

```
import sharp from 'sharp';
    const compressedImageBuffer = await sharp(profilePicture.buffer)
      .resize({ width: 400, height: 400 })
      .jpeg({ quality: 80 }) // Compress with 80% quality
      .toBuffer();
```

Ispis 14: Kompresija slike korištenjem knjižnice `Sharp`

Na ispisu 15. može se vidjeti kako se komprimirana slika sprema na disk. Prvo se provjerava postoji li direktorij za postavljanje; ako ne postoji, kreira se. Slika se sprema s jedinstvenim nazivom kako bi se spriječilo prepisivanje postojećih slika. URL spremljene slike generira se na temelju hosta i putanje do slike.

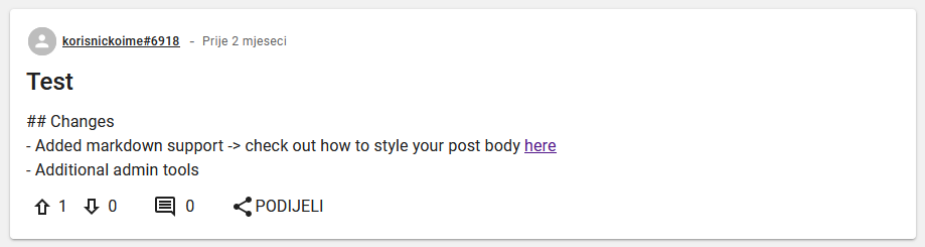
```
import fs from 'fs';
import path from 'path';
if (!fs.existsSync(uploadDir)) {
  fs.mkdirSync(uploadDir);
}

const filename = `profile_${Date.now()}.jpg`;

fs.writeFileSync(path.join(uploadDir, filename), compressedImageBuffer);

const profilePictureURL =
`${req.protocol}://${req.get('host')}/uploads/${filename}`;

await pool.execute('UPDATE users SET profilePictureURL = ? WHERE id = ?',
[profilePictureURL, user.id]);
```

Ispis 15: Spremanje slike na disk

Nakon što je profilna slika uspješno ažurirana, API vraća URL nove slike i poruku o uspjehu. Na taj način, korisnici mogu učinkovito ažurirati svoje profilne slike, a slike se obrađuju kako bi se optimizirala njihova veličina i očuvala kvaliteta.

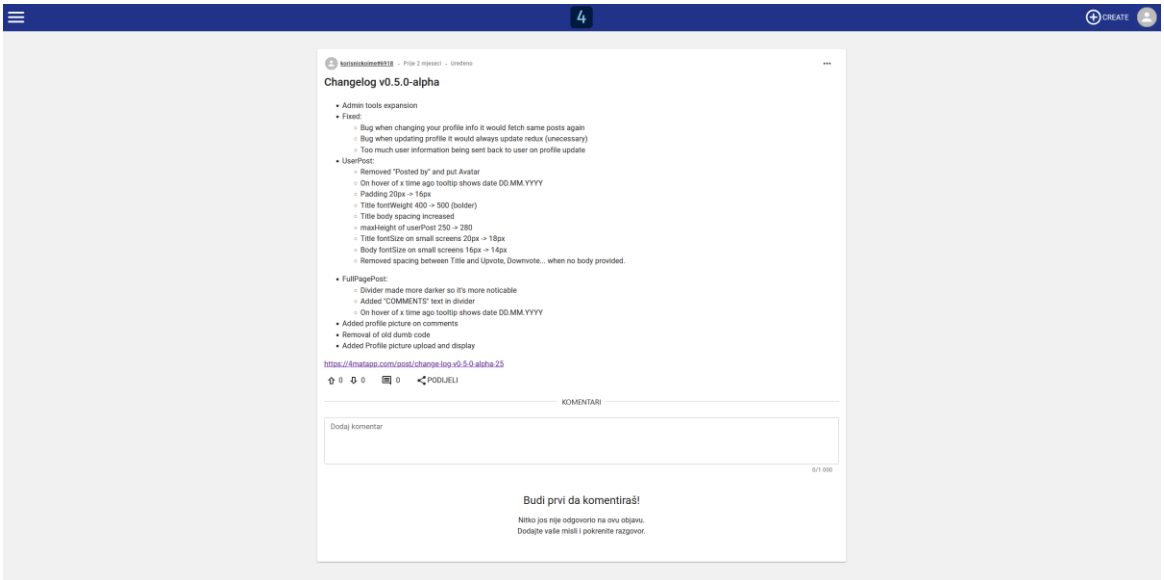
4.2. Objave

Objave može kreirati svaki registrirani korisnik nakon što potvrdi indentitet. Svaka objava sastoji se od naslova i tijela (engl. *body*), a uz nju se prikazuje korisničko ime autora objave i vrijeme kada je objava postavljena. Pored toga, objave sadrže brojače reakcija `sviđa mi se` i `ne sviđa mi se`, broj komentara te botun za dijeljenje objave, koji kopira poveznicu na objavu. Primjer korisničke objave može se vidjeti na slici 4.



Slika 4: Primjer objave

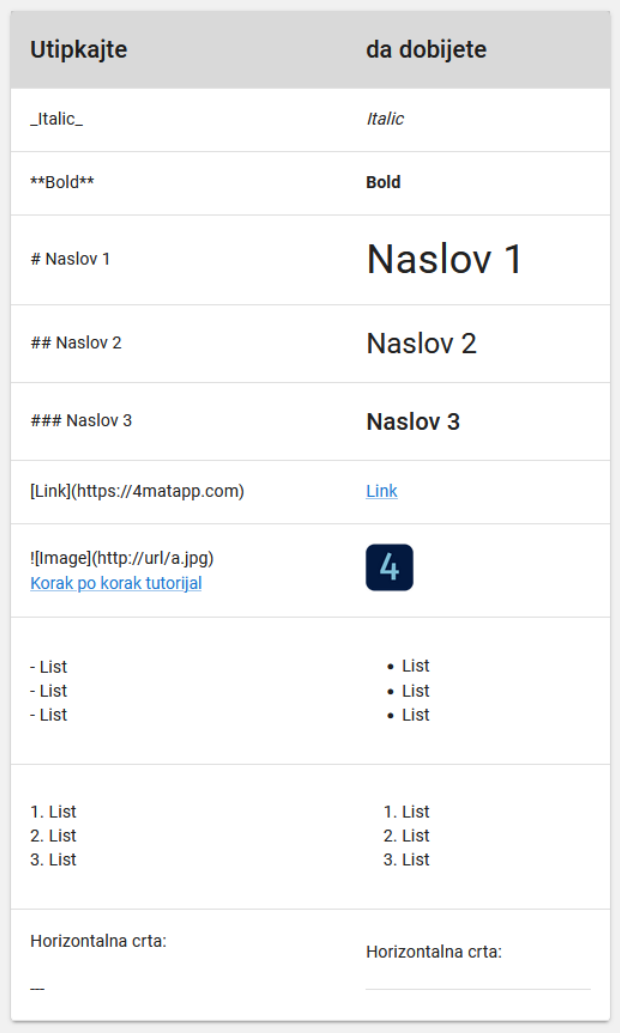
Na stranicama poput početne stranice (engl. *home*) i korisničkih profila, objave su prikazane u skraćenom obliku. Klikom na objavu, korisnik se preusmjerava na stranicu s punim prikazom objave, primjerice na URL poput `localhost/post/naslov-id_objave`, gdje može vidjeti cijelu objavu u njenom nepromijenjenom obliku. Na slici 5. vidimo prikaz pune korisničke objave.



Slika 5: Primjer prikaza cijele objave

Korisnici mogu uređivati i brisati vlastite objave. Ako korisnik izmijeni svoju objavu, na objavu se automatski dodaje oznaka „Uređeno“ kako bi ostali korisnici bili svjesni da je objava promijenjena.

Objave podržavaju stilizaciju pomoću `Markdown` sintakse. Za pretvaranje `Markdown-a` u stilizirani tekst koristi se `MuiMarkdown` biblioteka. Pri izradi objave, korisniku je dostupan botun „Stilizacija“, koji otvara kratki vodič s uputama koji možemo vidjeti na slici 6. Vodič ukratko prikazuje kako koristiti `Markdown` za postizanje različitih stilova teksta, poput Italic, Bold, naslova, linkova, slika, lista i horizontalne crte. Ovaj vodič pomaže korisnicima da svoje objave učine vizualno atraktivnijima i preglednijima.



Utipkajte	da dobijete
Italic	*Italic*
Bold	**Bold**
# Naslov 1	Naslov 1
## Naslov 2	Naslov 2
### Naslov 3	Naslov 3
[Link](https://4matapp.com)	Link
![Image](http://url/a.jpg) Korak po korak tutorijal	4
- List - List - List	• List • List • List
1. List 2. List 3. List	1. List 2. List 3. List
Horizontalna crta: ---	Horizontalna crta:

Slika 6: Stilizacijska pomoć korisniku pri kreiranju objave

Na ispisu 16. možemo vidjeti kako registrirani korisnici mogu kreirati nove objave putem API rute `/api/post/new`. API ruta je osigurana JWT autentifikacijom. Objava mora sadržavati naslov, a tijelo je opcionalno. Naslov i tijelo su validirani da budu unutar specificirane dužine.

```
app.post('/api/post/new', passport.authenticate('jwt', { session: false }),
[
  check('title').isString().isLength({ min: 4, max: 300 }).notEmpty(),
  check('body').isString().isLength({ max: 10000 }).optional()
], async (req, res) => {
  try {
    const { title, body } = req.body;
    const userID = (req.user as IUser).id;

    const [results] = await pool.execute(
      'INSERT INTO posts (posted_by, title, body) VALUES (?, ?, ?)',
      [userID, title, body]
    );
    const postId = (results as mysql.ResultSetHeader).insertId;

    res.status(201).json({ message: 'Post created successfully', postId });
  } catch (error) {
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Ispis 16: Isječak kôda sa strane poslužitelja za kreiranje nove objave

Na ispisu 17. vidimo API rutu `/api/posts/all` koja omogućava dohvaćanje svih objava, koje se mogu sortirati po najnovijima ili najpopularnijima. Umjesto tradicionalne paginacije, koristi se beskonačno listanje, što znači kako se dodatne objave automatski dohvaćaju kako korisnik lista prema dnu stranice.

Umjesto prebacivanja stranica, implementiran je mehanizam za beskonačno listanje koji omogućava dohvaćanje objava u dijelovima. API ruta podržava dohvaćanje objava na temelju *offseta* i broja objava po zahtjevu.

```
app.get('/api/posts/all', async (req: express.Request, res:
express.Response) => {
  try {
```

```
    // Extract the offset and sort criteria from the query parameters
    const offset = parseInt(req.query.offset as string) || 0;
    const sort = req.query.sort as string || 'new';

    // Number of posts to load at a time
    const postsPerPage = 10;

    // Determine the sorting criteria
    let orderByClause: string;
    switch (sort) {
      case 'top':
        orderByClause = `
          ORDER BY
          (SELECT COALESCE(SUM(CASE WHEN v.vote_type = 1 THEN 1 WHEN
v.vote_type = -1 THEN -1 ELSE 0 END), 0)
           FROM votes v WHERE v.post_id = p.post_id) DESC, p.date_posted
DESC`;
        break;
      case 'new':
      default:
        orderByClause = 'ORDER BY p.date_posted DESC';
        break;
    }

    // Fetch posts based on the sorting criteria and offset for infinite
scroll
    const [allPosts] = await pool.execute(
      `SELECT p.*,
              (SELECT COALESCE(SUM(CASE WHEN v.vote_type = 1 THEN 1 WHEN
v.vote_type = -1 THEN -1 ELSE 0 END), 0)
               FROM votes v WHERE v.post_id = p.post_id) AS score
       FROM posts p
       ${orderByClause}
       LIMIT ${postsPerPage} OFFSET ${offset}`
    );

    res.status(200).json({ allPosts });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Ispis 17: Isječak kôda za dohvaćanje objava sa strane poslužitelja

Na ispisu 18. koristimo biblioteku `react-virtuoso` za implementaciju beskonačnog listanja u kombinaciji s Reactom. Komponenta omogućuje dohvaćanje i prikaz objava dok korisnik lista, s mogućnošću sortiranja po najnovijima ili najpopularnijima. Kada korisnik dođe do kraja prikazanih objava, automatski se dohvaćaju nove.

```
      <Virtuoso
        useWindowScroll
        data={posts}
        endReached={loadMorePosts}
        itemContent={(index, post) => (
          <Box key={post.post_id} paddingTop={2}>
            <Container
              sx={{
                width: { xs: '100%', sm: '600px', md: '900px' },
                margin: { xs: 0, sm: 'auto' },
                padding: { xs: 0, sm: 'auto' }
              }}
            >
              <UserPost info={post} onDelete={handleDeletePost} />
            </Container>
          </Box>
        )}
        components={{
          Footer: () => (
            <Box sx={{ display: 'flex', justifyContent: 'center', padding:
2 }}>
              {loading ? <CircularProgress /> : !hasMore ?
t('endPostsComments') : null}
            </Box>
          ),
        }}
      />
```

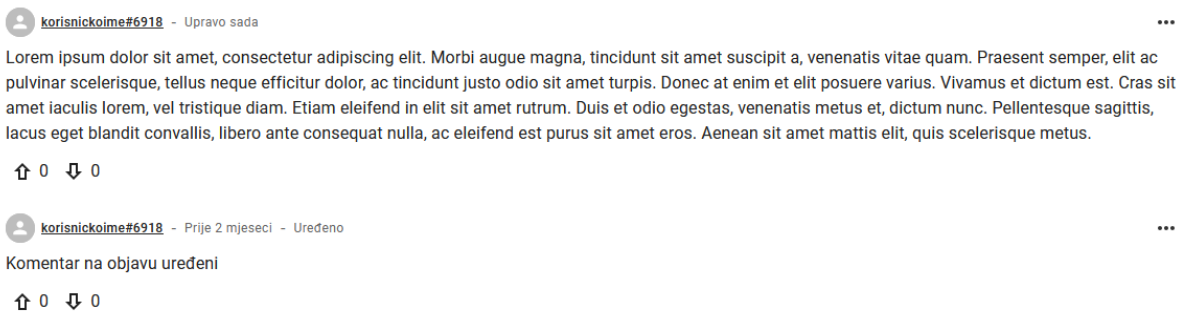
Ispis 18: Isječak kôda `virtuoso` koji omogućava beskonačno listanje

4.3. Komentari

Komentari u sustavu funkcioniraju slično kao i objave, ali su specifično povezani s određenom objavom. Korisnici mogu reagirati na komentare opcijama `sviđa mi se` ili `ne sviđa mi se`, što im omogućuje brzo izražavanje mišljenja o određenom komentaru. Svaki komentar jasno prikazuje tko ga je napisao i koliko je vremena prošlo od trenutka objave, što korisnicima pomaže u razumijevanju konteksta i relevantnosti komentara u odnosu na glavnu objavu. Na slici 8. možemo vidjeti primjer korisničkih komentara.

Korisnici imaju mogućnost uređivanja i brisanja svojih komentara. Ako je komentar izmijenjen, automatski se prikazuje oznaka „uređeno“ koja informira druge korisnike da je originalni sadržaj promijenjen. Također, korisnici mogu brisati svoje komentare, što im omogućuje uklanjanje komentara koji su možda pogrešno objavljeni ili više nisu relevantni.

Komentari su organizirani hijerarhijski ispod relevantnih objava, što korisnicima omogućuje jednostavan pregled i interakciju. Ove funkcionalnosti osiguravaju kvalitetnu i relevantnu komunikaciju unutar zajednice, omogućujući korisnicima strukturirano i organizirano sudjelovanje u raspravama.



Slika 8: Primjer komentara ispod objave

Ovaj kôd omogućuje korisnicima stavljanje novih komentara na određenim objavama. Komentari su zaštićeni autentifikacijom i imaju određene uvjete za unos. U nastavku slijedi detaljna razrada svakog dijela kôda.

Na ispisu 19. postavlja se nova ruta koja korisnicima omogućuje objavu novih komentara. Ruta koristi HTTP POST metodu i zahtijeva autentifikaciju putem JWT tokena, što znači kako samo prijavljeni korisnici mogu objavljivati komentare. Također, koristi se međuprogram za provjeru ispravnosti podataka komentara, osiguravajući da komentar bude niz i da se kreće u rasponu od 4 do 1000 znakova, bez praznih unosa.

```
app.post('/api/post/:postId/new/comment', passport.authenticate('jwt', {
session: false }), [
  check('body')
    .isString()
    .isLength({ min: 4, max: 1000 })
    .notEmpty(),
], async (req: express.Request, res: express.Response) => {
```

Ispis 19: Postavljanje nove rute API za objavu komentara

Na ispisu 20. prikazan je kôd koji izvlači sadržaj komentara i ID korisnika koji je trenutno prijavljen. Također dohvaća ID objave na koju se komentar odnosi. Prije nego što omogući dodavanje komentara, kôd provjerava postoji li objava s navedenim ID-om. Ako objava ne postoji, vraća se HTTP status kôd 404 s porukom „*Post not found*“ (Objava nije pronađena).

```
    const { body } = req.body;
    const userID = (req.user as IUser).id; // Use the authenticated user's
ID
    const postId = req.params.postId;
    // Check if the post exists
    const [post] = await pool.execute('SELECT * FROM posts WHERE post_id =
?', [postId]);
    const postRows = post as mysql.RowDataPacket[];
    if (postRows.length === 0) {
      return res.status(404).json({ message: 'Post not found' });
    }
```

Ispis 20: Provjera postoji li objava na koju se pokušava staviti komentar

Na ispisu 21. vidimo provjeru dio kôda koji pokušava umetnuti novi komentar u bazu podataka ako objava postoji. Koristi se SQL INSERT upit za dodavanje komentara u tablicu komentara, uključujući ID objave, ID korisnika koji je napisao komentar i tijelo komentara. Nakon umetanja, dohvaća se ID novog komentara.

```
    const [results] = await pool.execute(
      'INSERT INTO comments (post_id, posted_by, body) VALUES (?, ?, ?)',
      [postId, userID, body]
    );

    const commentId = (results as mysql.ResultSetHeader).insertId;
```

Ispis 21: Umetanje komentara u bazu podataka

Na ispisu 21. vidimo da nakon što je komentar uspješno spremljen, kôd emitira novi komentar svim povezanim `WebSocket` klijentima. Ovo omogućuje trenutačno ažuriranje komentara na svim korisničkim sučeljima koji su spojeni na sustav. Na kraju, vraća se HTTP status kôd 201 s porukom Komentar je uspješno kreiran i ID-om komentara.

```
    // Broadcast the new comment to all connected WebSocket clients
    broadcast({ type: 'new_comment', data: newComment });
```

Ispis 21: Slanje novog komentara svim povezanim `WebSocket` klijentima

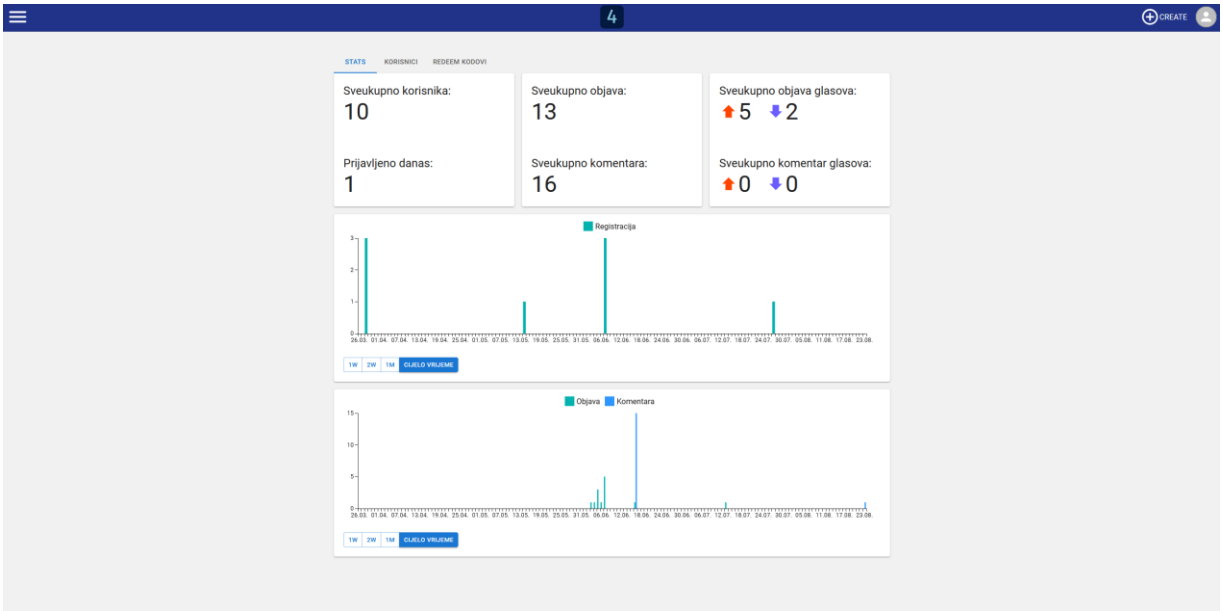
Na ispisu 22. vidimo kôd za `WebSocket` koji omogućava dvosmjernu komunikaciju između klijenta i poslužitelja preko stalne veze. Ovo omogućava poslužitelju slanje ažuriranja klijentima u stvarnom vremenu, što je korisno za funkcionalnosti poput chatova, live feedova ili ažuriranja podataka bez potrebe za ponovnim učitavanjem stranice.

```
// Create the HTTP server and integrate the WebSocket server
const server = http.createServer(app);
const { broadcast } = setupWebSocket(server);
```

Ispis 22: Postavljanje `WebSocket` integracije

4.4. Administrativni alati

Na stranici postoji administrativni panel koji nudi različite alate za upravljanje aplikacijom. Ovaj panel omogućuje pristup statistici, prikazu svih registriranih korisnika s njihovim podacima te upravljanje *redeem* kodovima.



Slika 7: Primjer statistike na administrativnom panelu

Na slici 7. nalazi se statistika na administrativnom panelu koja prikazuje ključne podatke kao što su sveukupni broj korisnika, današnje prijave, broj objava, komentara, te glasova na objavama i komentarima. Također, panel sadrži grafove koji prikazuju dnevni broj registriranih korisnika, kao i broj objava i komentara po danu.

Korisnici aplikacije mogu imati jednu od tri različite uloge, a svaka od njih donosi specifična prava i funkcionalnosti. Registrirani korisnici, administratori i superkorisnici imaju različite razine pristupa i mogućnosti unutar platforme.

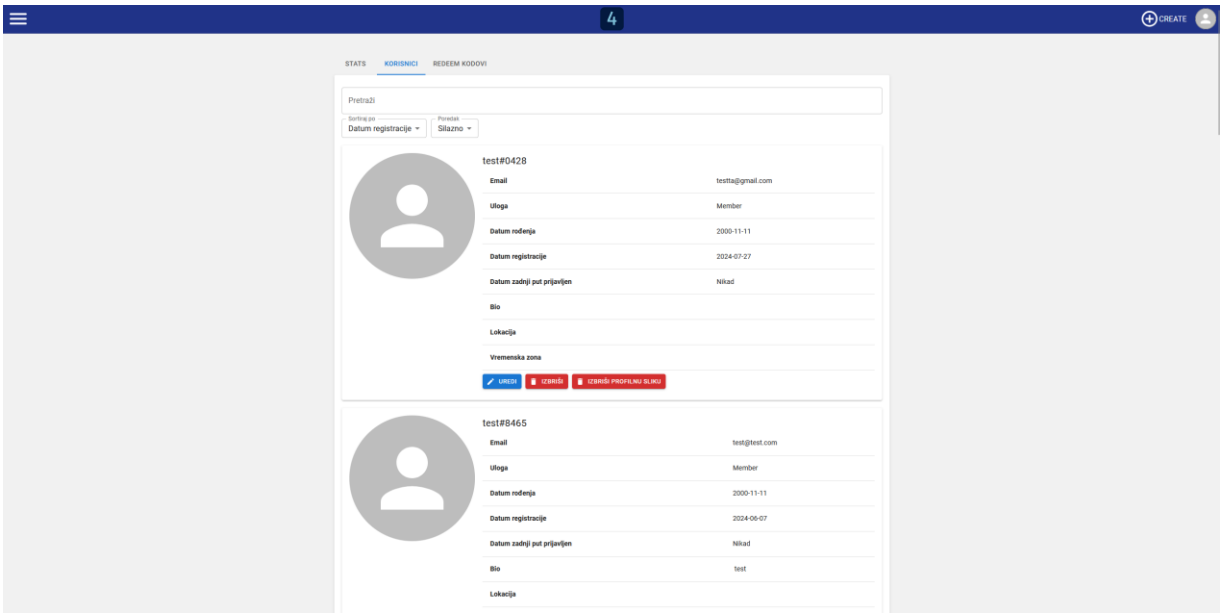
`Registrirani korisnici` mogu uređivati svoj profil, stvarati nove objave te mijenjati objave koje su sami stvorili. Također, mogu izražavati svoje mišljenje o objavama kroz funkcije `sviđa mi se` i `ne sviđa mi se` te komentirati objave drugih korisnika.

`Administratori` imaju sve mogućnosti koje imaju registrirani korisnici, ali i dodatne ovlasti. Uz mogućnost uređivanja i kreiranja vlastitih objava, administratori mogu i uređivati te brisati objave koje su stvorili drugi korisnici. Također, mogu pregledavati popis registriranih korisnika s određenim informacijama.

`Superkorisnici` posjeduju sve ovlasti administratora, ali i dodatne funkcionalnosti. Oni mogu pregledavati sve informacije o registriranim korisnicima i imati potpuni uvid u objave svih korisnika. Osim toga, superkorisnici mogu uređivati profile drugih korisnika, brisati korisnike te mijenjati njihove uloge kao što se vidi na slici 8.

Ovaj sustav uloga omogućava učinkovito upravljanje i kontrolu nad sadržajem i korisnicima unutar aplikacije, osiguravajući da se odgovornosti i ovlasti pravilno raspodijele prema potrebama i hijerarhiji unutar platforme.

Na strani klijenta, uloge korisnika pohranjuju se u *localStorage*. Kada korisnik izvrši bilo kakav zahtjev prema poslužitelju, poslužitelj zatim provjerava je li ta uloga autentična i ima li korisnik potrebne privilegije za pristup određenim resursima. Ova provjera osigurava da samo ovlašteni korisnici mogu pristupati specifičnim funkcionalnostima ili informacijama unutar aplikacije.



Slika 8: Primjer svih podataka registriranih korisnika za superkorisnika

Administrativni panel sadrži dio gdje su prikazane sve informacije o svim korisnicima, uključujući njihove profile i aktivnosti na platformi. Superkorisnici mogu uređivati preko ovoga dijela profile drugih korisnika, brisati korisnike, brisati njihove profilne slike te mijenjati njihove uloge.

Na ispisu 23. vidimo kôd za upravljanje pristupom. Koristi se međuprogramska funkcija *checkAdmin*, koja je dio Express aplikacije. Ova funkcija osigurava da samo korisnici s određenom ulogom mogu pristupati zaštićenim resursima ili funkcionalnostima.

```
export const checkAdmin = (req: Request, res: Response, next: NextFunction)
=> {
  passport.authenticate('jwt', { session: false }, (err: any, user: IUser,
info: any) => {
    if (err) {
      return res.status(500).json({ message: 'Internal server error' });
    }
    if (!user) {
      return res.status(401).json({ message: 'Unauthorized: User not logged
in' });
    }
    if (user.role === 0) {
      return res.status(403).json({ message: 'Forbidden: Admins only'});
    }
    req.user = user;
    next();
  })(req, res, next);
};
```

Ispis 23: Međuprogram za provjeru je li korisnik član administrativnog tima

5. Zaključak

Završni rad je ostvario svoj cilj razvoja moderne web aplikacije koja omogućava korisnicima jednostavno dijeljenje sadržaja, interakciju putem komentara te ocjenjivanje objava. Kroz pažljivo osmišljenu strukturu, aplikacija pruža ugodno korisničko iskustvo i omogućuje korisnicima intuitivno kretanje kroz različite funkcionalnosti. Uvođenjem sustava za administraciju i detaljne kontrole pristupa, osigurana je sigurnost i efikasno upravljanje sadržajem, što je ključno za dugoročno održavanje i razvoj aplikacije.

Jedan od ključnih elemenata uspjeha ovog završnog rada je korištenje suvremenih tehnologija i alata. Implementacija RESTful API-a omogućila je dosljednu i fleksibilnu komunikaciju između klijenta i servera, dok je JWT autentifikacija osigurala siguran pristup resursima. Uvođenjem `WebSocketa`, aplikacija je dobila mogućnost komunikacije u stvarnom vremenu preko komentara na objavama, što je dodatno unaprijedilo korisničko iskustvo.

Također, korištenje beskonačnog listanje uz pomoć `react-virtuoso` biblioteke omogućilo je glatko i kontinuirano učitavanje sadržaja, eliminirajući potrebu za tradicionalnom paginacijom i stvarajući osjećaj neprekinutog toka informacija. Ovaj pristup ne samo da poboljšava korisničko iskustvo, već optimizira performanse aplikacije.

Administrativni alati, integrirani u aplikaciju, omogućuju nadzor nad korisnicima i sadržajem, što je ključno za održavanje kvalitete i sigurnosti na platformi. Različite uloge korisnika, od običnih korisnika do superkorisnika, omogućuju detaljnu kontrolu pristupa i jasno definirane odgovornosti, što je osiguralo učinkovitost upravljanja.

Zaključno, ovaj završni rad predstavlja čvrstu osnovu za daljnji razvoj i prilagodbu aplikacije prema budućim potrebama i izazovima. Implementirane funkcionalnosti i korištene tehnologije pokazale su se vrlo učinkovitim, pružajući stabilan temelj za skaliranje i dodavanje novih mogućnosti. Završni rad je postigao balans između funkcionalnosti, sigurnosti i jednostavnosti korištenja, što ga čini uspješnim primjerom moderne web aplikacije.

LITERATURA

[1] Visual Studio Code, https://code.visualstudio.com/docs (posjećeno 11.9.2024)

[2] ReactJS https://react.dev/learn (posjećeno 11.9.2024)

[3] Typescript https://www.typescriptlang.org/docs/ (posjećeno 11.9.2024)

[4] MaterialUI https://mui.com/material-ui/getting-started/ (posjećeno 11.9.2024)

[5] Formik https://formik.org/docs (posjećeno 11.9.2024)

[6] i18n https://react.i18next.com/ (posjećeno 11.9.2024)

[7] react-router-dom https://reactrouter.com/en/main/start/overview (posjećeno 11.9.2024)

[8] Redux https://redux.js.org/introduction/getting-started (posjećeno 11.9.2024)

[9] Express.js https://expressjs.com/ (posjećeno 11.9.2024)

[10] MySQL https://dev.mysql.com/doc/ (posjećeno 11.9.2024)

[11] Express Validator https://express-validator.github.io/docs/ (posjećeno 11.9.2024)

[12] Bcrypt https://github.com/kelektiv/node.bcrypt.js#usage (posjećeno 11.9.2024)

[13] WebSocket https://github.com/websockets/ws?tab=readme-ov-file#usage-examples (posjećeno 11.9.2024)