Web aplikacija - portal za studente

Budano, Bernard

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

Stručni studij smjer Informatika

Web aplikacija – Portal za studente

Završni rad

Bernard Budano

Osijek, godina 2020.

SADRŽAJ

1. UVOD

U ovom završnom radu bit će opisana i objašnjena funkcionalnost i metode izrade web aplikacije Portal za studente. Glavne funkcionalnosti koje ova aplikacija pruža su mogućnost postavljanja pitanja putem objave, mogućnost odgovora na pitanja komentiranjem na objavu te mogućnost označavanja objave sa „Sviđa mi se“. Korisnici web aplikacije Portal za studente mogu uređivati svoj profil dodavanjem osobnih podataka poput svoje web stranice, *Github* profila i slično. Imaju i mogućnost dodavanja profilne slike koju vide ostali korisnici. Administrator aplikacije ima dodatne mogućnosti brisanja objava svih korisnika i davanja administratorskih privilegija ostalim korisnicima te oduzimanje istih.

Najprije se opisuje idejno rješenje zadatka rada. Nakon toga se pojašnjavaju sličnosti i razlike postojećih portala koji imaju približno jednak temeljni cilj kao web aplikacija Portal za studente. U nastavku su rada navedene i opisane sve tehnologije korištene za razvoj ove web aplikacije, na koji način su se koristile i u koju svrhu. Na kraju se opisuje funkcionalnost aplikacije kroz odlomke o poslužiteljskom te klijentskom dijelu aplikacije.

1.1. Zadatak rada

Opisati funkcionalnosti portala za studente gdje bi registrirani studenti bili u mogućnosti postavljati poruke s upitima za pomoć oko problema vezanih za pojedine sadržaje tijekom studija. Dizajnirati bazu podataka u koju će se pohranjivati svi relevantni podaci. Portal treba imati funkcionalnost postavljanja sadržaja, odgovora i komentara za registrirane korisnike. Izraditi administratorski panel za upravljanje sadržajima. Opisati tehnologije koje će se koristiti u izradi portala kao i sam postupak izrade.

2. IDEJNO RJEŠENJE

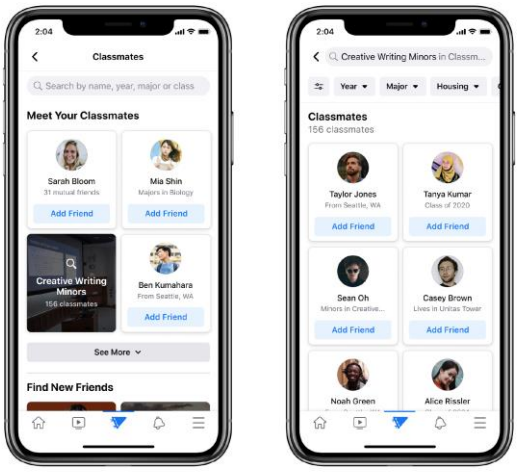
Ideja sustava je da se registriranim korisnicima omogući postavljanje pitanja drugim registriranim korisnicima putem objava te odgovaranje na pitanja drugih registriranih korisnika putem komentara. Ukoliko korisnik nema račun, može vidjeti objave registriranih korisnika, ali ne može kreirati svoje objave te komentirati i označavati objave registriranih korisnika sa „Sviđa mi se“, odnosno *like*-ati ih. Kako bi ostvario ta prava, korisnik registracijom treba kreirati korisnički račun te se prijaviti u sustav. Nakon prijave, korisnika se preusmjerava na početnu stranicu gdje može vidjeti postojeće objave poredane po datumu i vremenu objave te svoj profil. Na početnoj stranici korisnik ima mogućnost kreiranja i brisanja vlastitih objava, uređivanja svog profila, komentiranja i *like*-anja tuđih objava te posjećivanja profila registriranih korisnika. Ukoliko prijavljeni korisnik ima administracijske privilegije, on ima mogućnost brisanja svih objava, a posjećivanjem profila drugih korisnika ima mogućnost davanja i oduzimanja administracijskih privilegija. Cilj ove aplikacije je poboljšati umrežavanje studenata te olakšati njihovu međusobnu komunikaciju.

3. POSTOJEĆI PORTALI

U ovom odlomku bit će navedene sličnosti i razlike web aplikacije Portal za studente s već postojećim web aplikacijama. Za primjer web aplikacija koje imaju ili su originalno imale sličan cilj kao web aplikacija Portal za studente odabrane su *Facebook* i *LinkedIn*.

3.1. *Facebook*

Facebook je internetska društvena mreža čiji je primarni cilj pri osnivanju bio umrežavanje studenata na *Harvardu* kako bi mogli lakše međusobno komunicirati. Danas ima oko 2.7 milijardi aktivnih korisnika te osim komunikacije i izmjene informacija, podržava mnoge funkcionalnosti poput *Facebook* tržnice (engl. *Marketplace*), igrica, prikupljanja sredstava u dobrotvorne svrhe i *Facebook* Kampusa (engl. *Campus*). *Facebook* Kampus je prostor namijenjen fakultetima koji je dizajniran da pomogne studentima kako bi se povezali s kolegama. Korisničko sučelje *Facebook* Kampus projekta vrlo je intuitivno što je prikazano slikom 3.1. Kampus je projekt kojim se *Facebook* vratio svojim korijenima te je glavna poveznica današnjeg *Facebook*-a i web aplikacije Portal za studente. Za razliku od *Facebook*-a, web aplikacija Portal za studente je *lightweight* društvena mreža koja podržava isključivo funkcionalnosti potrebne studentima i djelatnicima fakulteta.



Slika 3.1. *Facebook Campus* korisničko sučelje

Izvor: https://about.fb.com/news/2020/09/introducing-facebook-campus/

3.2. *LinkedIn*

Iako je glavna misija *LinkedIn* društvene mreže olakšati nezaposlenima pronalazak nove karijere, poslodavcima pronalazak potencijalnih novih zaposlenika te umrežavanje zaposlenika unutar tvrtki, na *LinkedIn* društvenoj mreži registriran je velik broj studenata i djelatnika fakulteta. Sličnost web aplikacije Portal za studente i *LinkedIn* društvene mreže su funkcionalnosti poput kreiranja objava i postavljanja upita te olakšana komunikacija s drugim studentima i profesorima. Osim navedenih osnovnih funkcionalnosti, *LinkedIn* studentima olakšava pronalazak posla za vrijeme studija ili nakon završetka studija.

4. TEHNOLOGIJE

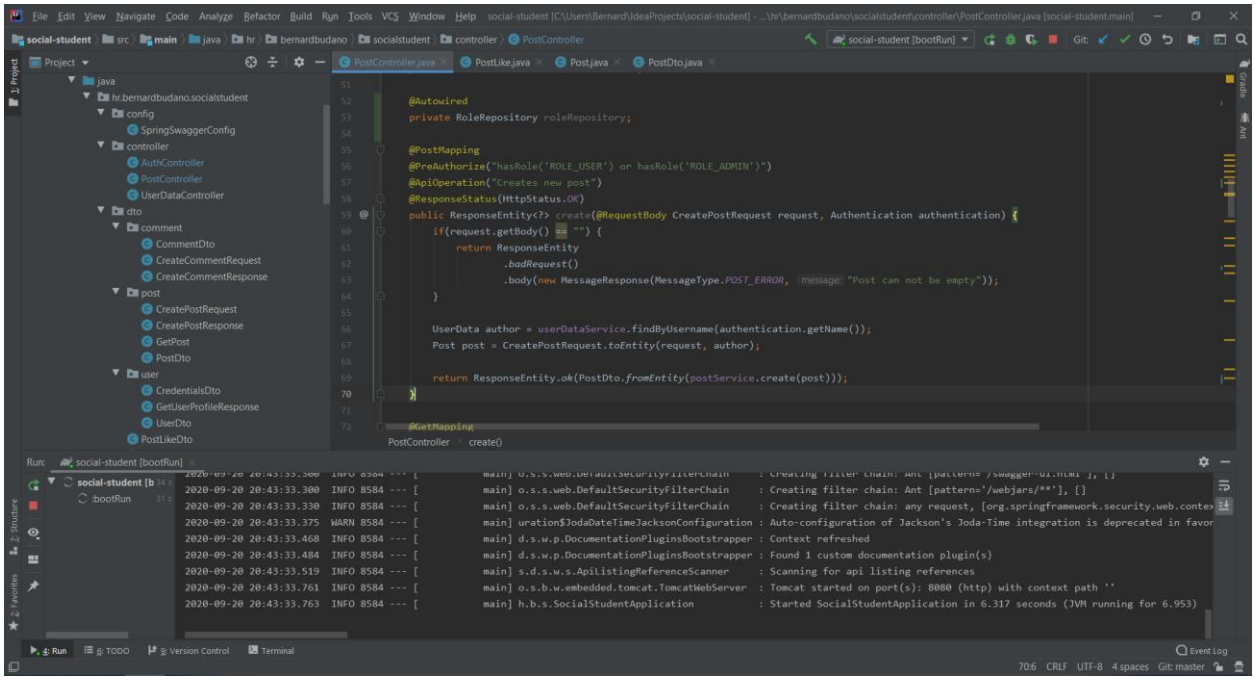
Tehnologije koje su korištene za izradu web aplikacije Portal za studente su podijeljene na dva dijela: klijentski i poslužiteljski. Korištene tehnologije na području klijentskog dijela aplikacije su *JavaScript* programski jezik, *React Javascript* programski okvir (engl. *framework*), *Yarn* alat za upravljanje ovisnostima (engl. *dependency management*), *Redux* biblioteka za upravljanjem stanja aplikacija (engl. *state management*) i *Material-UI React* UI (engl. *User Interface*) programski okvir. U poslužiteljskom dijelu aplikacije korišten je *Java* programski jezik, *Gradle* alat za izgradnju (engl. *build tool*), *Spring* programski okvir, *Hibernate* programski okvir, *Java Persistence* API (engl. *Application Programming Interface*) te *PostgreSQL* baza podataka uz *pgAdmin 4* grafičko sučelje za upravljanje bazom podataka. Za spremanje slika korišten je *Firebase storage*.

4.1. *IntelliJ IDEA* razvojno okruženje

IntelliJ IDEA je integrirano razvojno okruženje koje je razvila tvrtka *JetBrains* te je prikazano na slici 4.1. Pruža podršku za JVM (engl. *Java Virtual Machine*) jezike, odnosno jezike koji se mogu *compile*-ati u JVM bytecode:

- *Java*
- *Kotlin*
- *Scala*
- *Groovy*

Dizajnirano je kako bi poboljšalo produktivnost razvojnih programera te obavlja mnoge ponavljajuće zadatke svojom mogučnošću pametnog dovršavanja koda, statičkom analizom koda te refaktoriranjem.



Slika 4.1. *IntelliJ IDEA* razvojno okruženje

4.2. *Java* programski jezik

Java je programski jezik i računalna platforma koju je razvila tvrtka *Sun Microsystems* 1995. *Java* SE (engl. *Standard Edition*) omogućuje razvoj i korištenje *Java* aplikacija na radnim površinama ili poslužiteljima. *Java* EE (engl. *Enterprise Edition*) pruža API i okruženje za razvoj vrlo velikih, pouzdanih i sigurnih poslovnih *Java* aplikacija čija su glavna obilježja prenosivost i skalabilnost. *Java* programski jezik čini osnovu razvoja poslužiteljskog dijela web aplikacije Portal za studente.

4.3. *Gradle*

Gradle je *open-source* alat korišten za automatizaciju izrade koji se fokusira na performanse i fleksibilnost, a njegove se skripte za izgradnju pišu korištenjem *Groovy*-ja ili *Kotlin* DSL-a. U izradi ove web aplikacije korištene su dvije *Gradle* konfiguracije, *bootRun* za pokretanje aplikacije i *clean build* za ponovnu izgradnju.

4.5. *Hibernate* ORM (*Object-Relational Mapping*)

Hibernate ORM (engl. *Object-Relational Mapping*) je alat za *Java* programski jezik. On pruža programski okvir za mapiranje modela na objektno-orijentiranoj domeni s relacijskom bazom podataka. *Hibernate* rješava probleme objektno-relacijske neusklađenosti zamjenom izravnih pristupa bazi podataka s funkcijama za rukovanje objektima na visokoj razini. Olakšava progameru pisanje aplikacija kod kojih je životni vijek podataka dulji od samog procesa aplikacije. *Hibernate* je korišten kako bi se entiteti definirani u modelu mapirali u tablice u relacijskoj bazi podataka.

4.6. *Java Persistence* API (JPA)

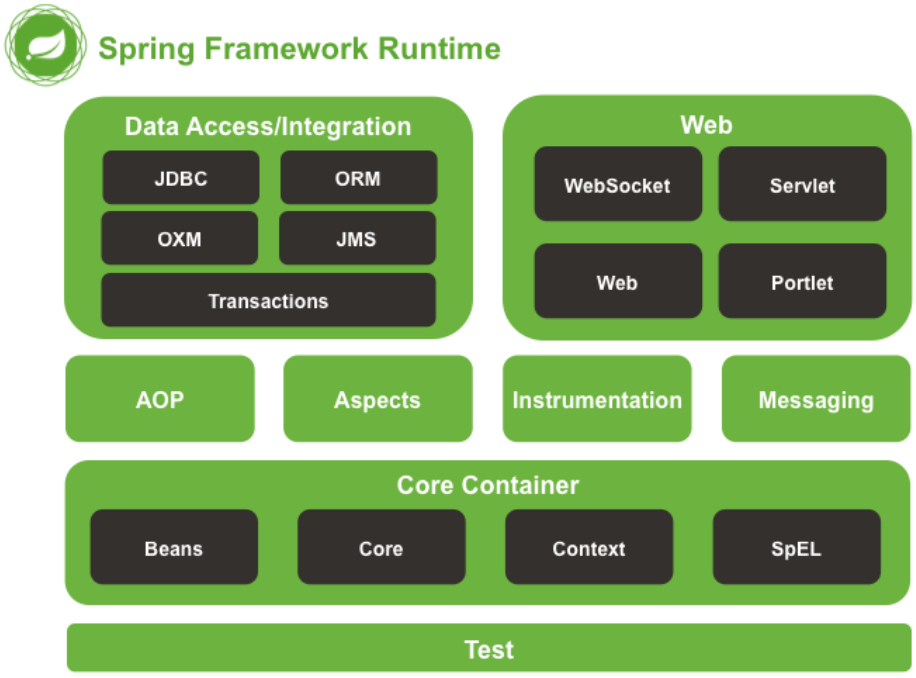
Java Persistence API (engl. *Application Programming Interface*) ili JPA je rješenje za dosljednost (engl. *persistence*) na temelju *Java* standarda. JPA koristi pristup objektno-relacijskog mapiranja kako bi spojio objektno-orijentirani model i relacijsku bazu podataka. JPA se također može koristiti i u *Java* SE aplikacijama, izvan *Java* EE okruženja. *Java Persistence* sastoji se od sljedećih područja:

- *Java Persistence* API
- Jezik upita (engl. *query language*)
- *Java Persistence Criteria* API
- Metapodaci objekto-relacijskog mapiranja

4.7. *Spring* programski okvir

Spring je *open-source* programski okvir za *Java* platformu. Pregled *Spring* programskog okvira prikazano je slikom 4.2. Njegove se glavne značajke mogu koristiti u bilo kojoj *Java* aplikaciji, ali sadrži ekstenzije za izradu web aplikacija na temelju *Java* EE platforme. Iako *Spring* ne nameće nikakav specifičan programerski model, postao je popularan u *Java* zajednici kao dodatak na EJB (engl. *Enterprise JavaBeans*) model. Glavni aspekt *Spring*-a je njegov spremnik inverzije kontrole (engl. *Inversion of Control Container*) koji pruža konzistentno sredstvo konfiguriranja i upravljanja *Java* objektima koristeći refleksiju. Spremnik je odgovoran za upravljanje životnim ciklusima objekata: stvaranje tih objekata, pozivanje njihovih metoda za inicijalizaciju i konfiguriranje tih

objekata njihovim međusobnim povezivanjem (engl. *autowiring*). Objekt koji spremnik kreira zove se upravljani objekt (engl. *managed object*) ili *Bean*. Moguća je konfiguracija spremnika korištenjem XML (engl. *Extensible Markup Language*) datoteka ili korištenjem specifičnih *Java* i *Spring* anotacija unutar konfiguracijskih klasa. Objekti se u *Spring* programskom okviru dobivaju korištenjem pretraživanjem ovisnosti ili naknadnim dodavanjem ovisnosti (engl. *dependency injection*). *Spring* programski okvir je baza uspostavljanja komunikacije između poslužiteljske i klijentske strane ove web aplikacije putem REST (engl. *Representational State Transfer*) API-ja.



Slika 4.2. Pregled *Spring* programskog okvira

Izvor: https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html

4.7.1. *Spring Boot*

Spring Boot je ekstenzija *Spring* programskog okvira koja eliminira *boilerplate* konfiguracije, odnosno dijelove konfiguracije koji se ponavljaju na puno mjesta u kodu te koji su potrebni za postavljanje *Spring* aplikacije. Olakšava kreiranje samostalnih aplikacija produkcijske kvalitete temeljenih na *Spring*-u koje jednostavno mogu pokrenuti. *Spring Boot* nudi ugrađen *Tomcat* server

koji je korišten u ovoj web aplikaciji, eksternaliziranu konfiguraciju te automatsku konfiguraciju *Spring* funkcionalnosti.

4.7.2. *Spring* MVC

Spring Web MVC (engl. *Model View Controller*) programski okvir je dizajniran oko *DispatcherServlet*-a koji otprema zahtjeve rukovateljima. Temeljni rukovatelj je baziran na *@Controller* i *@RequestMapping* anotacijama te nudi širok spektar fleksibilnih metoda rukovanja. Mehanizam *@Controller* omogućuje stvaranje *RESTful* web aplikacija, odnosno aplikacija koje prate principe REST arhitekture, kroz *@PathVariable* anotacije i druge značajke.

4.7.3. *Spring Data*

Spring Data je programski model baziran na *Spring*-u koji se koristi za pristup podacima, a pritom zadržavajući osobine temeljnih spremišta podataka. Omogućuje jednostavno korištenje tehnologija pristupa podacima, relacijskih i nerelacijskih baza podataka te podatkovnih usluga temeljenih na „oblaku“ (engl. *cloud*).

Spring Data JPA olakšava implementaciju repozitorija baziranih na *Java Persistence* API-ju. Ovaj se modul bavi poboljšanom podrškom za slojeve pristupa podacima bazirane na *Java Persistence* API-ju. Olakšava izradu *Spring* aplikacija koje koriste tehnologije za pristup podacima. *Spring Data* JPA modul korišten je za implementaciju komunikacije *Spring* aplikacije i *PostgreSQL* baze podataka.

4.7.4. *Spring Security*

Spring Security je vrlo moćan i prilagodljiv programski okvir za autentikaciju i kontrolu pristupa. Ovaj je programski okvir standard za zaštitu aplikacija temeljenih na *Spring* programskom okviru. Jedno od ključnih obilježja *Spring Security*-ja je to što se lako može proširiti kako bi se zadovoljile potencijalne dodatne potrebe. U web aplikaciji Portal za studente implementiran je na način da slanjem zahtjeva na API za autentikaciju korisnika vraća JWT (engl. JSON *Web Token*) ukoliko je autentikacija uspješna.

4.8. JSON *Web Token*

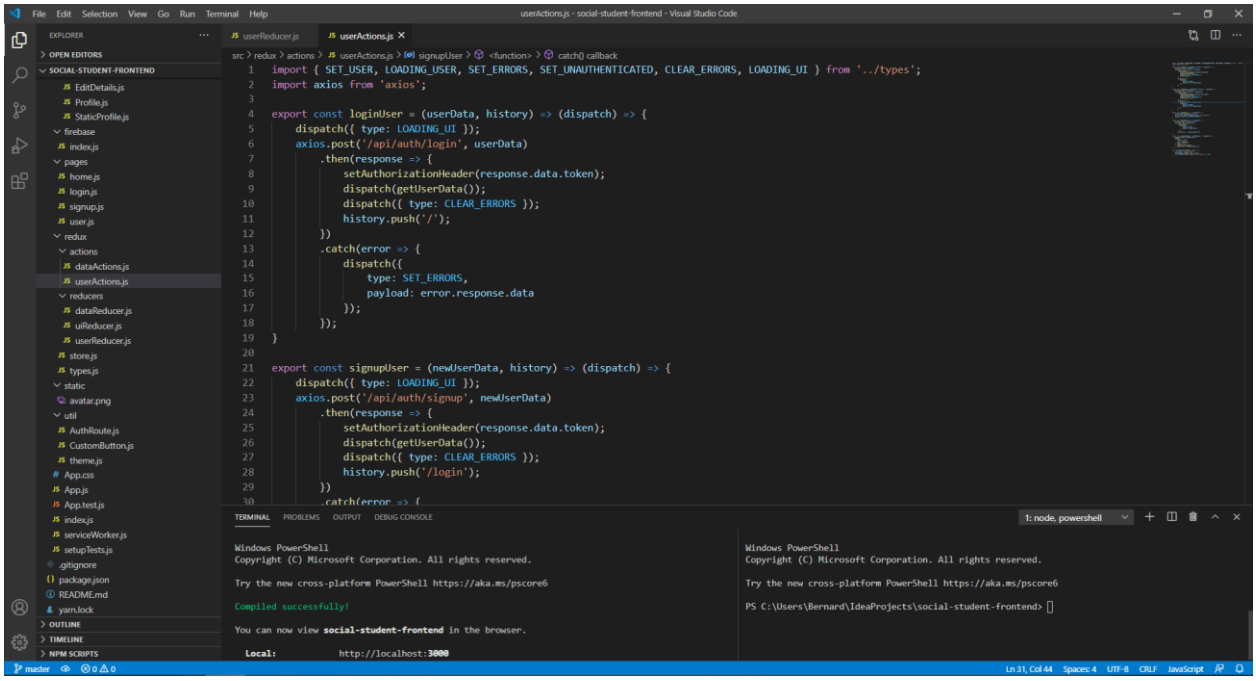
JSON *Web Token* je otvoreni standard (RFC 7519) te predstavlja jednostavan i siguran način prijenosa informacija u obliku JSON objekta. Informacije koje sadrži JWT su digitalno potpisane te se mogu provjeriti. JWT koji poslužitelj vrati nakon autentikacije korisnika, sprema se u lokalno skladište (engl. *local storage*) koje predstavlja *Web storage* objekt te se taj *token* sa svakim zahtjevom šalje na poslužiteljski dio aplikacije kako bi se utvrdilo može li korisnik pristupiti određenom API-ju.

4.9. *Swagger*

Swagger omogućuje opisivanje strukture API-ja kako bi ih strojevi mogli čitati. Sposobnost API-ja da opisuju vlastitu strukturu je temelj funkcionalnosti *Swagger* alata. Čitanjem strukture korisničkih API-ja, *Swagger* automatski generira interaktivnu API dokumentaciju. Lokalnim pokretanjem web aplikacije Portal za studente, API dokumentacija se može pregledati posjećivanjem *localhost:8080/swagger-ui.html#/* adrese.

4.10. *Visual Studio Code* razvojno okruženje

Visual Studio Code je lagan, ali moćan uređivač izvornog koda dostupan za *Windows*, *MacOS* i *Linux* te je prikazan na slici 4.3. Podržava *JavaScript*, *TypeScript* i *Node.js* te ima mogućnost instalacije proširenja za druge programske jezike. Korišten je za razvoj klijentskog dijela web aplikacije Portal za studente pisanog u *JavaScript* programskom jeziku koristeći *React* programski okvir.



Slika 4.3. *Visual Studio Code* programsko okruženje

4.11. *JavaScript*

JavaScript je lagan, interpretiran, objektno-orijentiran programski jezik prvoklasnih funkcija najpoznatiji kao skriptni jezik za web stranice. Koristi se i u mnogim okruženjima koja nisu u web pregledniku. Baziran je na prototipima te je dinamičan i podržava objektno-orijentirane, imperativne i funkcionalne stilove programiranja. *JavaScript* se pokreće na strani klijenta, što se može koristiti za dizajniranje i programiranje načina na koji će se web stranica ponašati pojavom određenog događaja. Može funkcionirati kao proceduralni i objektno-orijentirani programski jezik.

4.12. *Yarn*

Yarn je alat za upravljanje ovisnostima (engl. *dependency management*) koji omogućuje upotrebu i dijeljenje koda s drugim razvojnim programerima. Olakšava korištenje riješenja drugih razvojnih programera za različite probleme što znatno olakšava razvoj *software*-a. Također se koristi za podizanje lokalnog razvojnog poslužitelja (engl. *development server*).

4.13. *React*

React je *open-source JavaScript* biblioteka za izradu korisničkih sučelja. Može se koristiti za razvoj jednostranih aplikacija, odnosno SPA (engl. *Single Page Application*) te mobilnih aplikacija. Bavi se samo predajom podataka DOM-u (engl. *Document Object Model*) te zbog toga stvaranje *React* aplikacija obično zahtjeva korištenje dodatnih biblioteka za upravljanje stanjem (engl. *state management*) i usmjeravanje (engl. *routing*). Primjeri takvih biblioteka su *Redux* za *state management* i *React Router* za *routing*. Korišten je za izradu klijentskog dijela web aplikacije Portal za studente u obliku *Single Page* aplikacije.

4.13.1. *Redux*

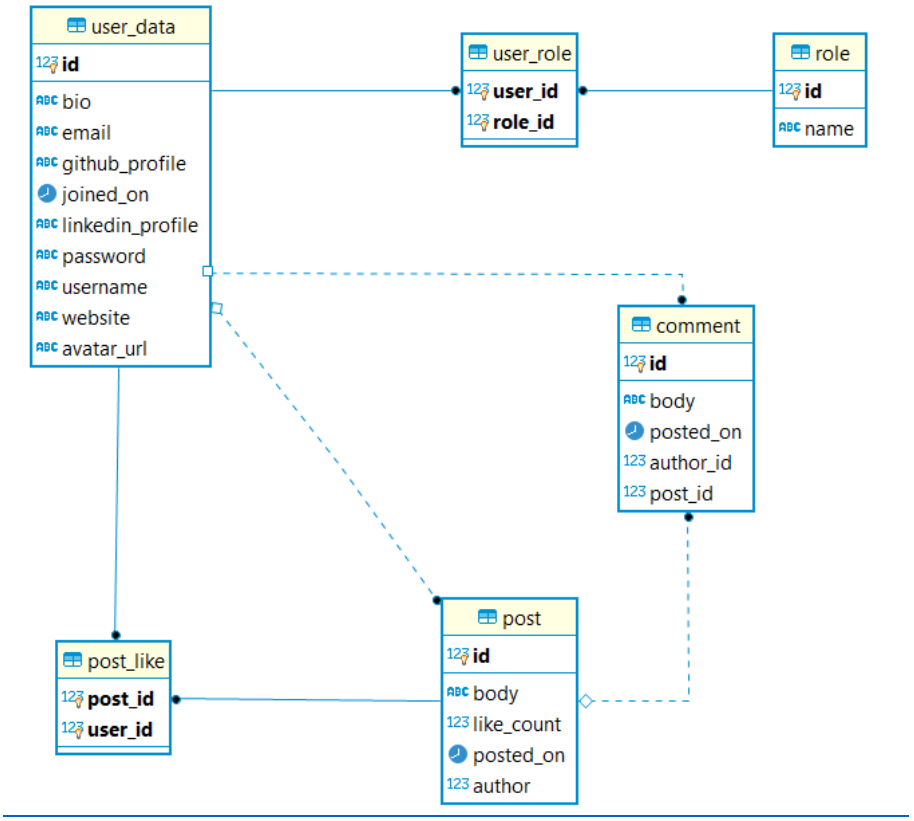
Redux je predvidljivi spremnik stanja (engl. *state container*) za *JavaScript* aplikacije. Znatno olakšava proslijeđivanje svojstava (engl. *properties*) kroz hijerarhiju *React* komponenti.

4.13.2. *Material-UI*

Material-UI je *React* UI programski okvir koji korisniku pruža velik broj gotovih UI komponenti poput gumbova, polja za unos, navigacijske trake, kartica itd. Korišten je za realizaciju osmišljenog dizajna ove web aplikacije.

4.14. *PostgreSQL*

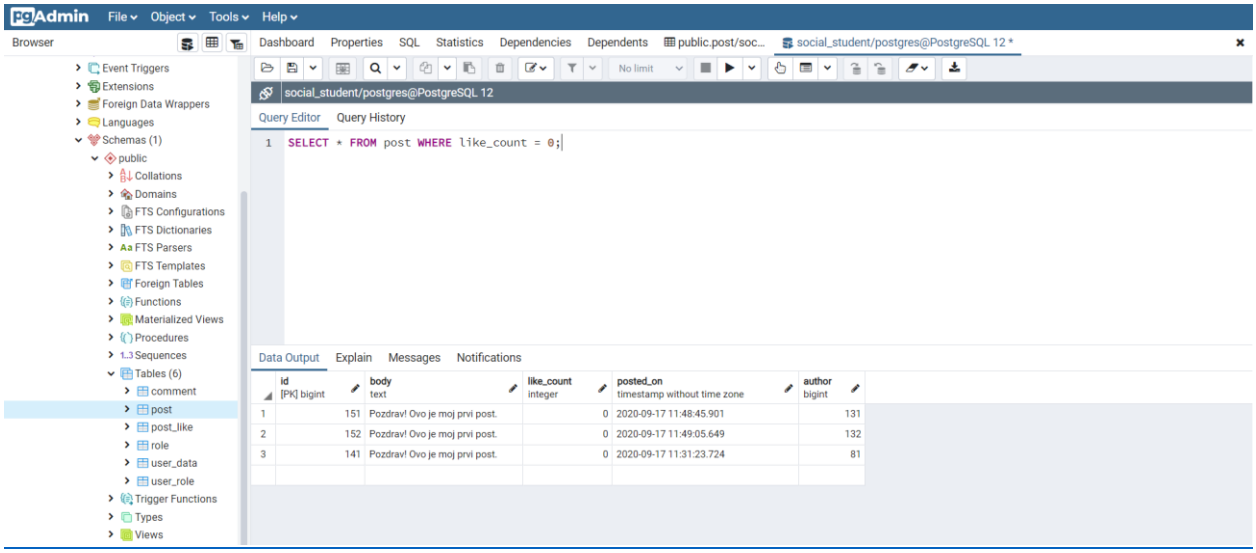
PostgreSQL je moćan, open-source sustav za objektno-relacijske baze podataka koji koristi i proširuje SQL (engl. *Structured Query Language*) jezik u kombinaciji s mnogim značajkama koje sigurno pohranjuju i skaliraju (engl. *scale*) komplicirane podatke. *PostgreSQL* radi na svim glavnim operacijskim sustavima i ima moćne dodatke poput popularnog proširenja geospacijalnih baza podataka *PostGIS*. *PostgreSQL* je odabran sustav za relacijsku bazu podataka web aplikacije Portal za studente, a ER (engl. *Entity Relationship*) dijagram te baze podataka prikazan je na slici 4.4.



Slika 4.4. ER dijagram baze podataka

4.14.1. *PgAdmin 4*

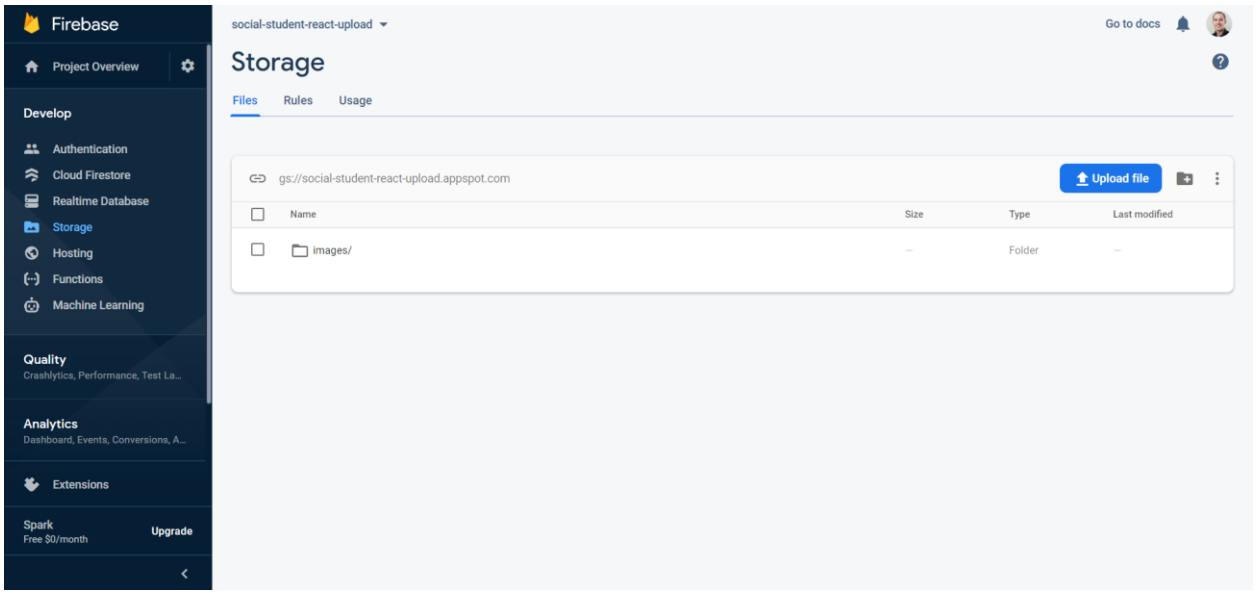
PgAdmin je najpopularniji alat za upravljanje *PostgreSQL* bazama podataka. Dizajniran je za početnike i iskusne razvojne programere koji koriste *PostgreSQL* te pruža moćno i intuitivno korisničko sučelje za kreiranje, održavanje i korištenje objekata iz baze podataka prikazano na slici 4.5.



Slika 4.5. *pgAdmin 4* korisničko sučelje

4.15. *Firebase*

Firebase je *Google*-ova platforma za razvoj web i mobilnih aplikacija. Od njegovih mnogih servisa, za razvoj ove web aplikacije korišten je *Firebase storage* u svrhu pohrane profilnih slika korisnika, prikazan na slici 4.6.



Slika 4.6. *Firebase storage*

5. FUNKCIONALNOSTI WEB APLIKACIJE

Web aplikacija Portal za studente je podijeljena na klijentski (*frontend*) dio te poslužiteljski (*backend*) dio. U ovom će poglavlju biti opisane i objašnjene funkcionalnosti i metode izrade tih dijelova.

5.1. Poslužiteljski dio aplikacije

Poslužiteljski dio aplikacije podijeljen je u nekoliko slojeva. Sastoji se od modela, objekata za prijenos podataka (engl. *Data Transfer Object*), repozitorija, servisa i kontrolera. Entiteti su klase koje će pomoću *Hibernate* programskog okvira biti mapirane u tablice u bazi podataka. DTO-ovi su objekti koji prenose podatke između procesa. Repozitoriji su *Java* sučelja koje upravljaju tokom podataka između poslužitelja i baze podataka. Servisi su klase koje sadrže poslovnu logiku aplikacije. Kontroleri, odnosno REST (engl. *Representational State Transfer*) kontroleri sadrže API-je koji služe za komunikaciju poslužiteljskog i klijentskog dijela aplikacije.

5.1.1. Entiteti

Entiteti, odnosno JPA entiteti su POJO-i (engl. *Plain old Java object*) koji predstavljaju podatke zadržane u bazi podataka. Svaki entitet predstavlja tablicu spremljenu u bazi podataka. Svaka instanca entiteta predstavlja jedan redak u toj tablici. Entiteti se označavaju s *@Entity* anotacijom kako bi JPA bio svjestan da ga treba premapirati u tablicu. Svaki entitet mora sadržavati prazan konstruktor te primarni ključ koji se označava anotacijom *@Id*.

```java
@Entity
@Table(name = "post")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Post {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "post_id_seq_generator")
    @SequenceGenerator(name = "post_id_seq_generator", sequenceName = "post_id_seq", allocationSize = 10, initialValue = 10)
    @Column(name = "id", updatable = false)
    private Long id;

    @Column(columnDefinition = "TEXT", name = "body", nullable = false)
    private String body;

    @Column(name = "posted_on", nullable = false)
    private DateTime postedOn = DateTime.now();

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "author", nullable = false)
    private UserData author;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(
            mappedBy = "post",
            cascade = CascadeType.ALL,
            orphanRemoval = true
    )
    private List<Comment> comments = new ArrayList<>();

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(
            mappedBy = "post",
            cascade = CascadeType.ALL,
            orphanRemoval = true
    )
    private List<PostLike> likes = new ArrayList<>();

    public void removeComment(final Iterator i, Comment comment) {
        comment.setPost(null);
        i.remove();
    }

    public void removeLike(final Iterator i, PostLike postLike) {
        postLike.setPost(null);
        i.remove();
    }

}
```

Slika 5.1. *Post* entitet

Entiteti, osim standardnih atributa, sadrže i atribute koji predstavljaju veze s drugim entitetima, odnosno s drugim tablicama u bazi podataka. Takve atribute označavamo s jednom od četiri *Hibernate* anotacije: *@OneToOne*, *@OneToMany*, *@ManyToOne* i *@ManyToMany*, ovisno o potrebnoj vezi između entiteta. *Post* entitet, prikazan na slici 5.1., sadrži nekoliko ovakvih veza. Sadrži

@ManyToOne vezu s *UserData* tablicom, što znači da će u tablici *post* postojati strani ključ iz tablice *user_data* koji predstavlja korisnika koji je objavio taj *post*, odnosno autora. Kolekcija *Comment* entiteta označena je anotacijom *@OneToMany*, što znači da će tablica *comment* sadržavati kolonu *post_id* koja predstavlja strani ključ iz tablice *post*. Taj podatak je bitan kako bi se znalo na koji je *post* objavljen svaki od komentara. Entitet *Post* sadrži i kolekciju *PostLike* entiteta. *PostLike* je entitet koji predstavlja mapirajuću tablicu u kojoj postoje samo dva strana ključa, strani ključ *post_id* koji označava koja je objava *like*-ana te *user_id* koji označava koji je korisnik *like*-ao objavu. Klasa *Post* označena je i s *@Data* anotacijom. Ova anotacija dolazi iz *Lombok* biblioteke i omogućava pozadinsko generiranje *getter, setter, toString, equals* i *hashCode* metoda bez ručne implementacije istih. Označena je i s *@NoArgsConstructor* te *@AllArgsConstructor* koje pozadinski kreiraju prazan konstruktor i konstruktor sa svim atributima klase *Post*.

5.1.2. *Repository* sučelja

Spring anotacija *@Repository* označava da je označena klasa repozitorij. Repozitorij je mehanizam za enkapsulaciju pohrane, dohvaćanja i pretraživanja podataka. Anotacija *@Repository* je specijalizirana *@Component* anotacija koja omogućava da se implementacijske klase automatski detektiraju putem skeniranja klasa.

```
11  @Repository
12  public interface PostRepository extends JpaRepository<Post, Long> {
13
14      Page<Post> findAllByOrderByPostedOnDesc(Pageable pageable);
15
16      Optional<Post> findById(Long id);
17
18  }
19
```

Slika 5.2. *PostRepository* sučelje

PostRepository sučelje sa slike 5.2. koristi se za komunikaciju poslužiteljskog dijela aplikacije s *post* tablicom u bazi podataka. Sadrži dvije metode: *findAllByOrderByPostedOnDesc* i *findById*. Metoda *findById* kao argument prima *id* te traži entitet iz tablice *post* koji ima točno taj *id* u svojoj *id* koloni. Ova metoda vraća *Optional<Post>* objekt. *Optional* je objekt spremnik koji može i ne mora

sadržavati *null* vrijednost. *Optional* je korišten iz razloga što se može dogoditi da u bazi podataka ne postoji podatak koji tražimo te će u tom slučaju metoda vratiti *null*. Naziv metode *findById* dobar je primjer funkcionalnosti i vrlo intuitivnog načina korištenja *Spring Data* JPA modula koji će od imena metode pozadinski formirati SQL upit. U ovom slučaju za ime metode *findById*, formirat će se upit koji glasi „SELECT * FROM post WHERE id = :id“. Metoda *findAllByOrderByPostedOnDesc* prima argument *Pageable pageable*. *Pageable* je apstraktno sučelje za informacije o paginaciji te dolazi iz *Spring* programskog okvira. Ova metoda pretražuje i vraća sve redove iz tablice *post* te ih sortira po datumu objave, odnosno koloni *posted_on*, silaznim redoslijedom. Kolekciju pronađenih entiteta vratit će u obliku *Page* objekta. *Page* je klasa koja dolazi iz *Spring* programskog okvira i predstavlja podlistu liste objekata. Omogućuje dobivanje informacija o njenom položaju u kompletnoj listi.

5.1.3. *Service* klase

Spring anotacija *@Service* označava da se radi o servisu te da je to klasa koja drži poslovnu logiku aplikacije. Kao i *@Repository* anotacija, ona je specijalizirana *@Component* anotacija.

```
8  public abstract class AbstractService<Entity, Repository extends PagingAndSortingRepository<Entity, Long>> {
9
10     protected final Repository repository;
11
12     public AbstractService(final Repository repository){
13         this.repository = repository;
14     }
15
16     public Page<Entity> getPaged(final Pageable pageable) { return repository.findAll(pageable); }
19
20     public Page<Entity> getPaged(final Integer page, final Integer size) {
21         return repository.findAll(PageRequest.of(page, size));
22     }
23
24     public Entity create(final Entity entity) { return repository.save(entity); }
27
28     public void delete(final Entity entity) { repository.delete(entity); }
31 }
32
```

Slika 5.3. *AbstractService* klasa

```java
@Service
public class PostService extends AbstractService<Post, PostRepository> {

    @Autowired
    private PostRepository postRepository;

    public PostService(PostRepository repository) { super(repository); }

    public Page<Post> findAllByOrderByPostedOnDesc(Pageable pageable) {
        return postRepository.findAllByOrderByPostedOnDesc(pageable);
    }

    public Post findById(final Long id) {
        return postRepository.findById(id).orElseThrow(() -> new RuntimeException("Post not found."));
    }

}

```

Slika 5.4. *PostService* klasa

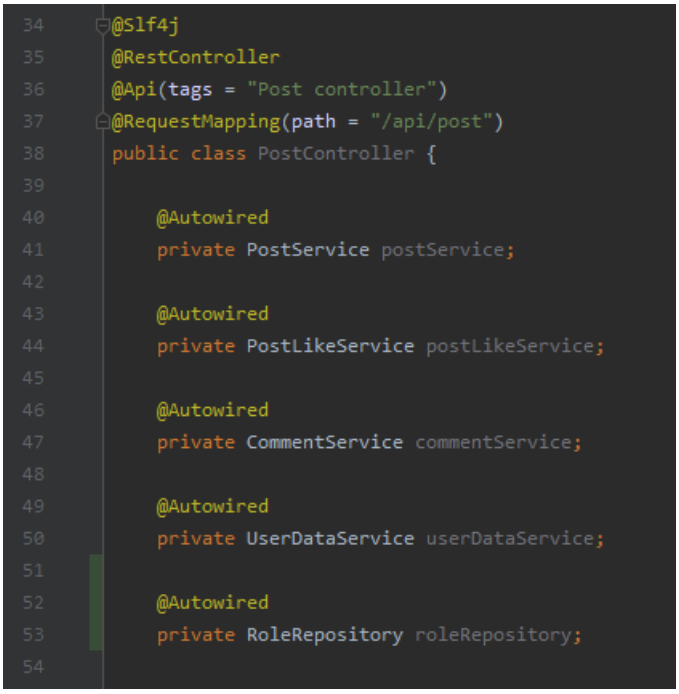
PostService klasa prikazana na slici 5.3. sadrži poslovnu logiku koja se odnosi na *Post* entitet i nasljeđuje *AbstractService* apstraktnu klasu prikazanu na slici 5.4. U njoj se nalaze metode čija je zadaća izvršavanje operacija nad bazom podataka, odnosno pozivanje metoda definiranih u *Repository* sučelju ovog entiteta. *PostService* klasa sadrži dvije metode:

- *findById* koja kao argument prima *id* tipa *Long*
- *findAllByOrderByPostedOnDesc* koja kao argument prima instancu *Pageable* klase

Ove metode pozivaju metode definirane u *PostRepository* sučelju, a pristup tim metodama te samom *PostRepository* sučelju ostvaruje se korištenjem *@Autowired* anotacije. U ovom primjeru, kreiranjem *PostService* servisa, *Spring* traži i naknadno dodaje ovisnost (engl. *dependency injection*) *PostRepository* u ovaj servis i time omogućava korištenje svih metoda u *PostRepository* sučelju. Osim implementacije metoda iz repozitorija, servisi sadrže bilo kakvu implementaciju poslovne logike.

5.1.4. Kontroleri i REST kontroleri

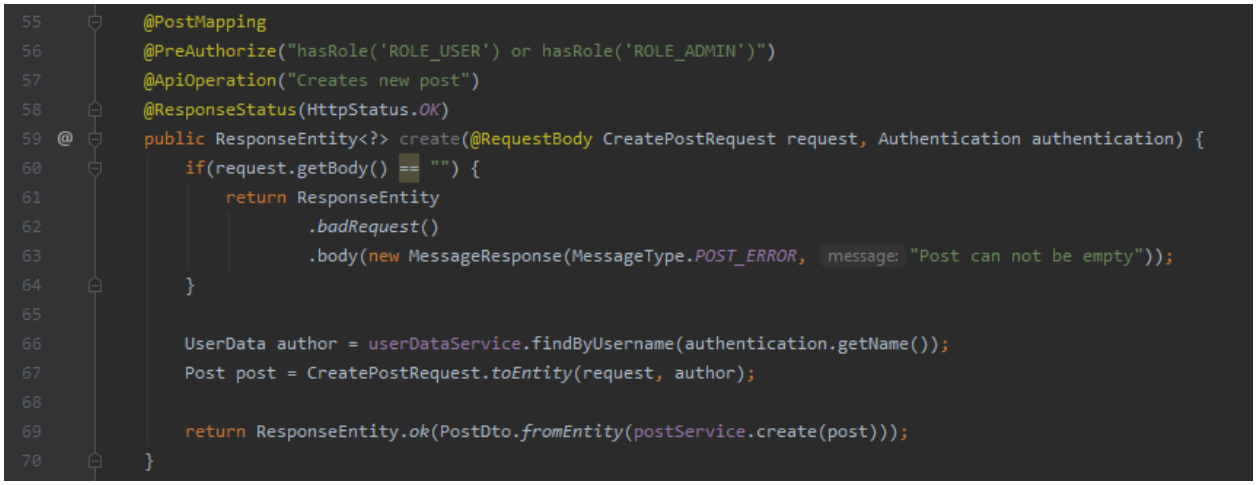
Kontroleri su klase koji presreću dolazeće zahtjeve, pretvaraju podatke iz zahtjeva u unutarnju strukturu podataka, šalju podatke modelu na daljnju obradu te dobivaju određene podatke iz modela i prosljeđuju ih u klijentski dio aplikaije kako bi se prikazali. Klasični kontroler se označava anotacijom *@Controller* u kombinaciji s anotacijom *@RequestMapping* koja se koristi nad metodama za obradu zahtjeva. Anotacija *@Controller* je specijalizirana *@Component* anotacija i omogućuje automatsko detektiranje implementacijskih klasa putem skeniranja klasa. REST kontroler je specijalizirana verzija kontrolera. Objedinjuje *@Controller* i *@ResponseBody* anotacije u *@RestController* anotaciji te time pojednostavljuje implementaciju samog kontrolera. REST (engl. *Representational State Transfer*) je stil razvijanja *software*-a koji definira skup ograničenja koja će se koristiti za stvaranje web servisa. *PostController* klasa sadrži metode koje predstavljaju REST API-je. Metode su označene anotacijama *@GetMapping*, *@PostMapping* i *@DeleteMapping*. Te anotacije označavaju koju je HTTP metodu potrebno koristiti kako bi klijentski dio aplikacije došao do ispravnog API-ja implementiranog u poslužiteljskom dijelu aplikacije, u odgovarajućem kontroleru. Metode implementirane u *PostController* klasi koriste se kako bi koristeći *PostService* klasu i *PostRepository* sučelje izvršile željene operacije nad bazom podataka poput dohvaćanja objave po *id*-u ili brisanja retka iz tablie *post*. Osim toga, one vraćaju tražene podatke u obliku odgovora (engl. *response*) na klijentski dio aplikacije, odnosno *frontend*.



Slika 5.5. *PostController* klasa

PostController klasa prikazana na slici 5.5. označena je s nekoliko anotacija. *@Slf4j* anotacija omogućuje korištenje *logger* objekta koji služi za ispisivanje informacija u konzolu. *@Api* anotacija uz *tag* parametar dodjeljuje ime ovom kontroleru u *Swagger* korisničkom sučelju. *@RequestMapping* anotacija sa svojim *path* parametrom daje krajnju točku (engl. *endpoint*) *Post* kontroleru. Prije implementacije API-ja, korištenjem *@Autowired* anotacije naknadno se dodaju ovisnosti potrebne za funkcioniranje kontrolera:

- *PostService* koji sadrži poslovnu logiku vezanu uz objave
- *PostLikeService* koji sadrži poslovnu logiku vezanu uz *like*-ove
- *CommentService* koji sadrži poslovnu logiku vezanu uz komentare
- *UserDataService* koji sadrži poslovnu logiku vezanu uz korisnike
- *RoleRepository* koji sadrži metode za slanje upita na tablicu *role* u bazi podataka

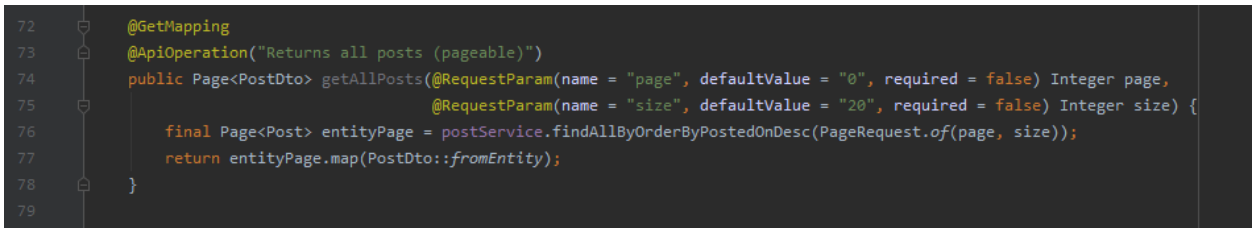


Slika 5.6. API za kreiranje nove objave

Kako bi korisnik korisnik kreirao novu objavu mora biti autenticiran, odnosno prijavljen u sustav, mora imati određenu ulogu (engl. *role*), mora poslati zahtjev na odgovarajući *endpoint* te njegov zahtjev mora sadržavati odgovarajuće tijelo (engl. *body*). Metoda *create* prikazana na slici 5.6. označena je s nekoliko anotacija. *@PostMapping anotacija* označava da ova metoda mapira HTTP POST zahtjeve. *@PreAuthorize* anotacija dolazi iz *Spring Security* programskog okvira te određuje kakvi korisnici mogu pristupiti ovoj metodi, u ovom slučaju korisnici koji imaju ulogu administratora ili standardnog korisnika. *@ApiOperation* je anotacija koja daje ime ovom API-ju u *Swagger* korisničkom sučelju. Anotacija *@ResponseStatus* koristi se za postavljanje status koda HTTP odgovora. Metoda *create* prima dva parametra:

- Objekt *request* klase *CreatePostRequest* koji predstavlja tijelo zahtjeva
- Objekt *authentication* klase *Authentication* koji sadrži informacije o autenticiranom korisniku

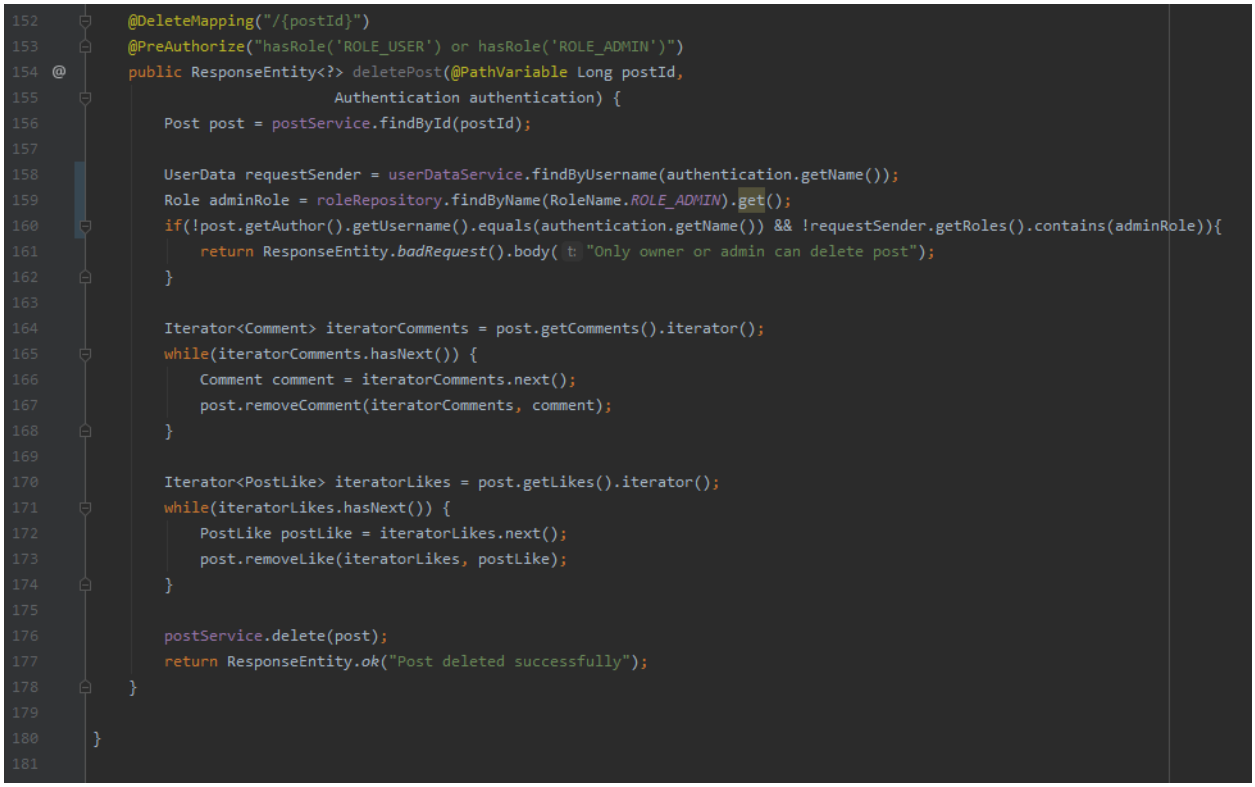
Objekt *request* klase *CreatePostRequest* označen je *@RequestBody* anotacijom koja označava da će se tijelo zahtjeva mapirati u ovaj objekt te sadrži atribut *body* koji predstavlja sadržaj nove objave. Objekt *authentication* služi kao dokaz da je zahtjev poslao korisnik koji je prijavljen u sustav. Jedan od atributa *authentication* objekta je korisničko ime korisnika koji je poslao zahtjev te se preko korisničkog imena dolazi do odgovarajućeg *UserData* entiteta. Kako bi se kreirala nova objava koristi se *UserData* entitet koji predstavlja autora objave te *body* atribut iz *request* objekta.



```
72      @GetMapping
73      @ApiOperation("Returns all posts (pageable)")
74      public Page<PostDto> getAllPosts(@RequestParam(name = "page", defaultValue = "0", required = false) Integer page,
75                                       @RequestParam(name = "size", defaultValue = "20", required = false) Integer size) {
76          final Page<Post> entityPage = postService.findAllByOrderByPostedOnDesc(PageRequest.of(page, size));
77          return entityPage.map(PostDto::fromEntity);
78      }
79
```

Slika 5.7. API za dohvaćanje svih objava

Metoda *getAllPosts* prikazana na slici 5.7. služi kao API za dohvaćanje svih objava. Označena je s *@GetMapping* anotacijom što govori da ova metoda mapira HTTP GET zahtjeve. Metoda *getAllPosts* kolekciju pronađenih objekata pretvorenih u DTO-ove (engl. *Data Transfer Object*) vraća u obliku *Page* objekta. Parametar *page* označava broj stranice, a parametar *size* označava koliko će *Post* entiteta biti dohvaćeno iz baze podataka po svakoj stranici. Oba parametra označena su s *@RequestParam* anotacijom koja omogućava dodavanja *defaultValue* atributa. U ovom slučaju, atribut *page* ima zadanu vrijednost 0, a atribut *size* ima zadanu vrijednost 20 što znači ako se pošalje zahtjev bez tih parametara, metoda će vratiti prvih 20 pronađenih redova iz tablice *post*.



```
@DeleteMapping("/{postId}")
@PreAuthorize("hasRole('ROLE_USER') or hasRole('ROLE_ADMIN')")
public ResponseEntity<?> deletePost(@PathVariable Long postId,
                    Authentication authentication) {
    Post post = postService.findById(postId);

    UserData requestSender = userDataService.findByUsername(authentication.getName());
    Role adminRole = roleRepository.findByName(RoleName.ROLE_ADMIN).get();
    if(!post.getAuthor().getUsername().equals(authentication.getName()) && !requestSender.getRoles().contains(adminRole)){
        return ResponseEntity.badRequest().body( t: "Only owner or admin can delete post");
    }

    Iterator<Comment> iteratorComments = post.getComments().iterator();
    while(iteratorComments.hasNext()) {
        Comment comment = iteratorComments.next();
        post.removeComment(iteratorComments, comment);
    }

    Iterator<PostLike> iteratorLikes = post.getLikes().iterator();
    while(iteratorLikes.hasNext()) {
        PostLike postLike = iteratorLikes.next();
        post.removeLike(iteratorLikes, postLike);
    }

    postService.delete(post);
    return ResponseEntity.ok("Post deleted successfully");
}

}
```

Slika 5.8. API za brisanje objave

Metoda *deletePost* prikazana na slici 5.8. označena je s *@DeleteMapping* anotacijom koja označava da ova metoda mapira HTTP DELETE zahtjeve. Ona prima dva argumenta:

- Objekt *postId* klase *Long* koji predstavlja *id* objave koju korisnik želi obrisati
- Objekt *authentication* klase *Authentication* koji sadrži informacije o autenticiranom korisniku

Preko *postId* objekta pronalazi se *Post* entitet koji korisnik želi obrisati iz baze. Unutar metode se zatim potvrđuje da je korisnik koji je poslao zahtjev autor objave koju želi obrisati ili da ima ulogu administratora. Ako je jedan od tih uvijeta ispunjen, prvo se brišu svi komentari i *like*-ovi vezani za objavu koja će biti obrisana, a zatim se briše sama objava.

5.1.5. *Data Transfer* objekti (DTO)

Objekt za prijenos podataka, odnosno *Data Transfer* objekt ili DTO je objekt koji prenosi podatke između procesa. DTO dizajnerski uzorak (engl. *design pattern*) je u ovoj aplikaciji korišten kako bi se izbjegao prijenos nepotrebnih podataka te time smanjio promet podataka između poslužiteljskog i

klijentskog dijela aplikacije. U slučaju slanja cijelog entiteta koji sadrži relacije s drugim tablicama, može doći do problema beskonačne rekurzije pri prijenosu podataka.

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PostDto {

    private Long id;

    private String body;

    private String author;

    private String authorAvatar;

    private DateTime postedOn;

    private int likeCount;

    private int commentCount;

    public static PostDto fromEntity(Post post){
        PostDto postDto = new PostDto();
        postDto.setId(post.getId());
        postDto.setBody(post.getBody());
        postDto.setAuthor(post.getAuthor().getUsername());
        postDto.setAuthorAvatar(post.getAuthor().getAvatarUrl());
        postDto.setPostedOn(post.getPostedOn());
        postDto.setLikeCount(post.getLikes().size());
        postDto.setCommentCount(post.getComments().size());
        return postDto;
    }

}
```

Slika 5.9. *PostDto* klasa

PostDto klasa sa slike 5.9. korištena je u *getAllPosts* metodi iz kontrolera koja vraća kolekciju ovih objekata u *Page* obliku. Klijentski dio aplikacije poziva tu metodu kako bi prikazao objave na početnoj stranici. Iz tog razloga *PostDto* klasa sadrži samo podatke koji se prikazuju na klijentskom dijelu aplikacije te *id* kako bi svaka objava na početnoj stranici imala jedinstveni identifikator. S obzirom da baza podataka vraća cijele entitete, *PostDto* sadrži metodu *fromEntity* kojom se svaki od njih prevodi u *PostDto* objekt prije slanja na klijentski dio aplikacije.

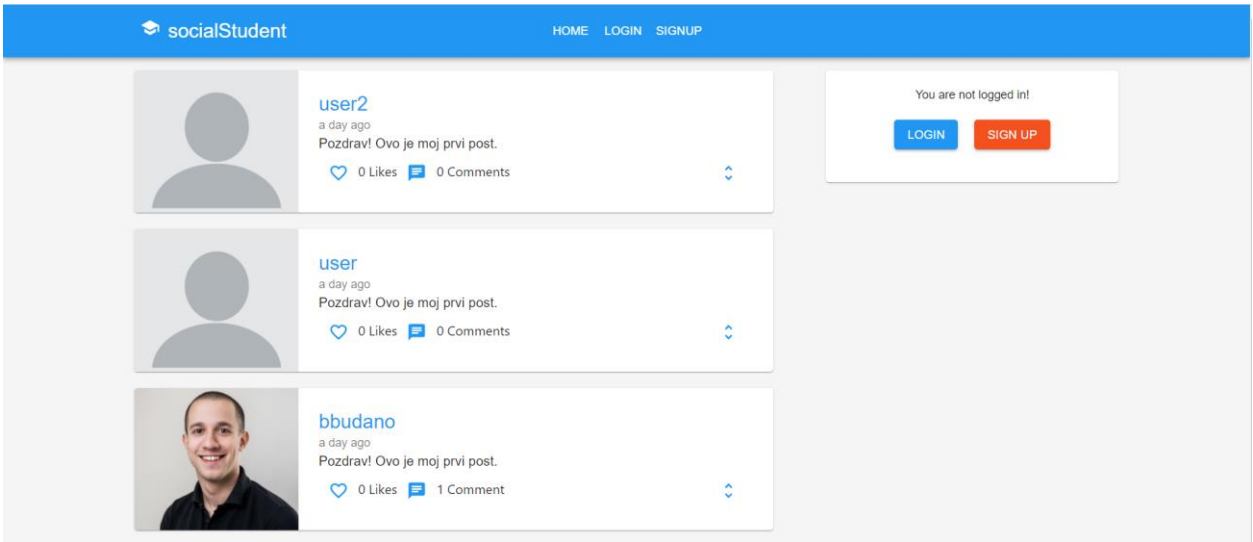
5.2. Klijentski dio aplikacije

Klijentski dio aplikacije predstavlja dio web aplikacije koji je prikazan korisniku, odnosno onaj koji se odvija na klijentskoj strani. Izrađen je korištenjem *React JavaScript* programskog okvira. *Redux* biblioteka korištena je za upravljanje stanjem aplikacije te omogućava pristup globalnim podacima bez proslijeđivanja istih kroz hijerarhiju *React* komponenti. Za dizajn kompletne web aplikacije zaslužan je *Material-UI React* UI programski okvir. Komunikaciju s poslužiteljskim dijelom aplikacije omogućava *Axios* HTTP klijent pomoću kojega se šalju zahtjevi. Usmjeravanje (engl. *Routing*) je implementirano korištenjem *React Router* biblioteke koja se sastoji od navigacijskih komponenti, a čija je implementacija prikazana na slici 5.10.

```
39  class App extends Component {
40    render() {
41      return (
42        <MuiThemeProvider theme={theme}>
43          <Provider store={store}>
44            <Router>
45              <div className="container">
46                <Navbar />
47                <Switch>
48                  <Route exact path="/" component={home} />
49                  <AuthRoute exact path="/login" component={login} />
50                  <AuthRoute exact path="/signup" component={signup} />
51                  <Route exact path="/users/:username" component={user} />
52                </Switch>
53              </div>
54            </Router>
55          </Provider>
56        </MuiThemeProvider>
57      );
58    }
59  }
60
61  export default App;
62
```

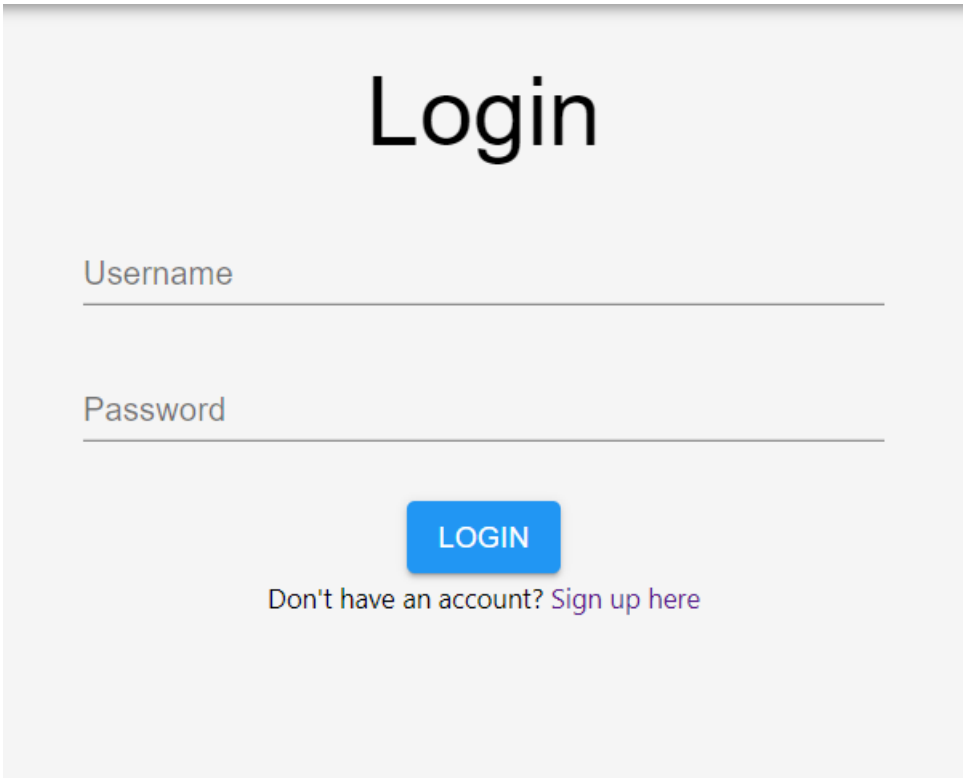
Slika 5.10. Implementacija *React Router*-a

5.2.1. Prijava u sustav



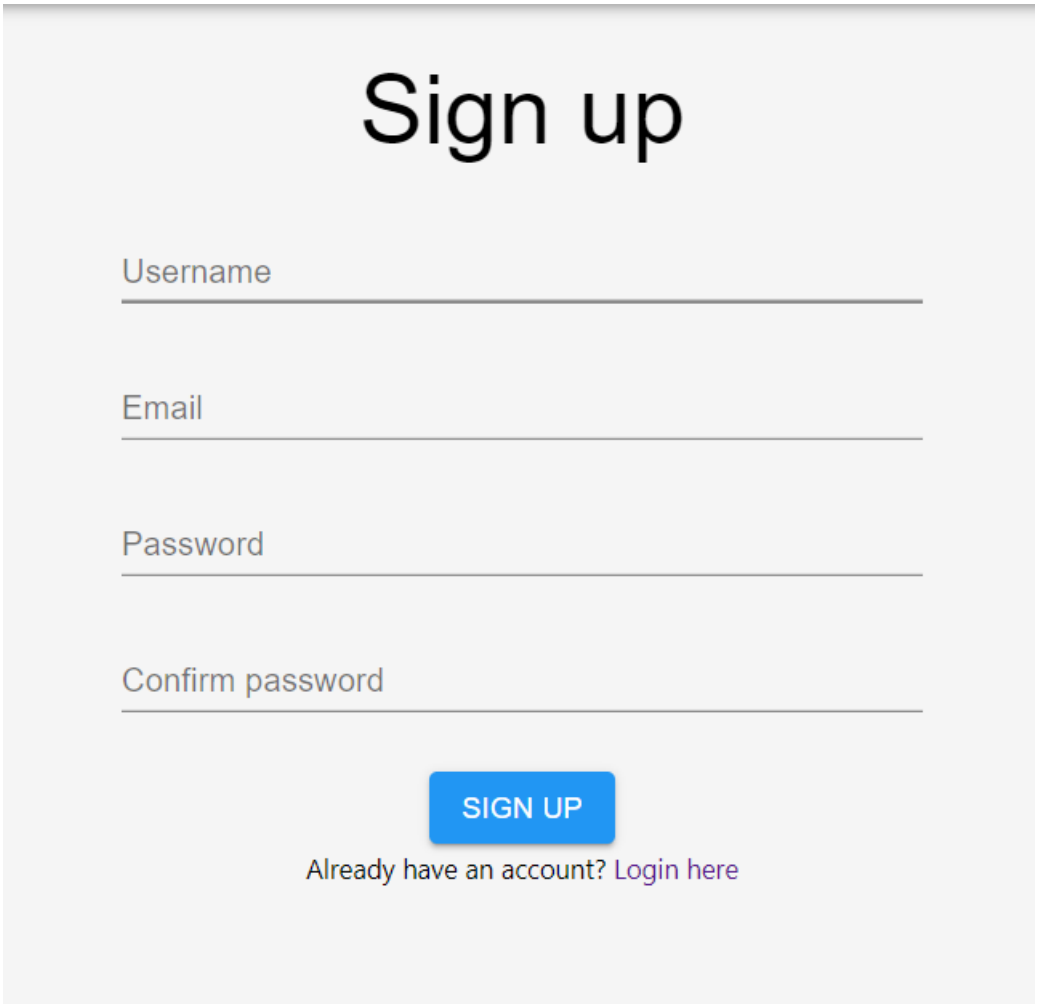
Slika 5.11. Početna stranica (neprijavljeni korisnik)

Dolaskom na početnu stranicu aplikacije prikazanu na slici 5.11., neprijavljeni korisnik, odnosno korisnik bez korisničkog računa može vidjeti objave, ali ne može s njima ući u interakciju. Unutar kartice s desne strane mu je prezentirana poruka u kojoj piše da nije prijavljen i ponuđene su mu dvije opcije, prijava (engl. *login*) ili registracija (engl. *sign up*). U ovom stanju aplikacije, kada korisnik nije prijavljen, navigacijska traka sadrži tri gumba od kojih jedan vodi na početnu stranicu, jedan na prijavu i jedan na registraciju.



Slika 5.12. Obrazac za prijavu u sustav

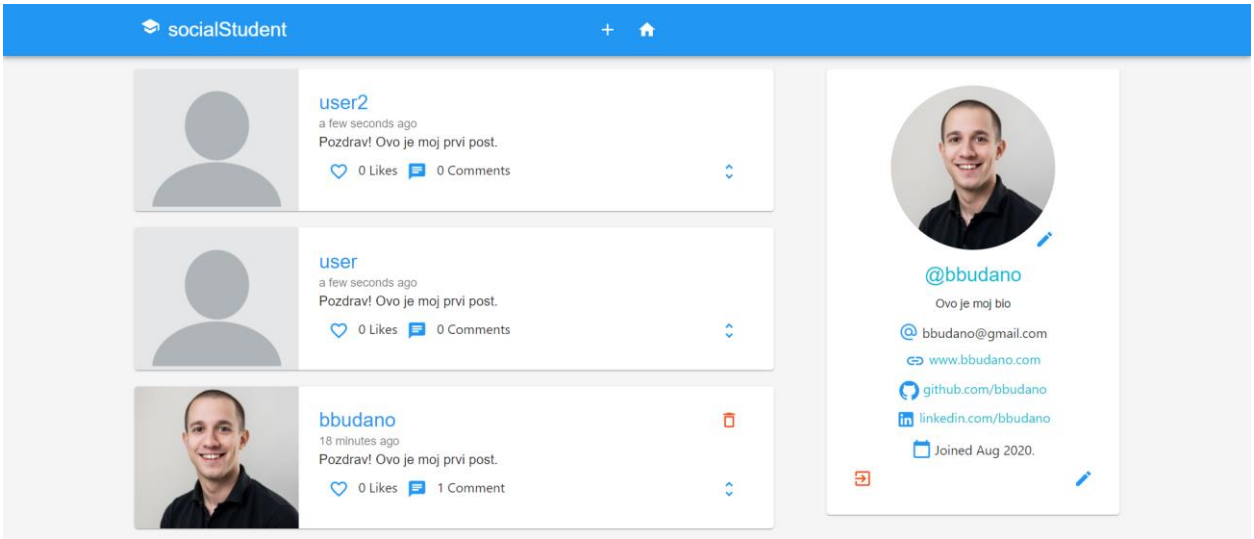
Klikom na *Login* gumb u navigacijskoj traci ili u kartici na desnoj strani početne stranice, korisnika se odvodi na novu stranicu u kojoj se nalazi obrazac za prijavu prikazan na slici 5.12. Obrazac se sastoji od dva polja za unos, jedno za korisničko ime (engl. *username*) i jedno za lozinku (engl. *password*), te *Login* gumba kojim se nakon unosa korisničkog imena i lozinke prijavljuje u sustav. Ispod *Login* gumba nalazi se poruka u kojoj piše „Nemate račun? Prijavite se ovdje“ te se u njoj nalazi poveznica (engl. *link*) koji odvodi na stranicu za registraciju.



Slika 5.13. Obrazac za registraciju

Klikom na *Sign up* gumb u navigacijskoj traci ili u kartici na desnoj strani početne stranice, korisnika se preusmjerava na stranicu u kojoj se nalazi obrazac za registraciju novog korisnika, odnosno kreiranje novog korisničkog računa, prikazan na slici 5.13. Ona se sastoji od polja za unos korisničkog imena, email adrese, lozinke te polja za ponovni unos lozinke kako bi se osiguralo da korisnik nije krivo unio željenu lozinku. Na dnu obrasca nalazi se *Sign up* gumb. Klikom na taj gumb se od unesenih podataka kreira novi korisnički račun te se korisnika šalje na stranicu za prijavu u sustav. Ispod gumba nalazi se poruka u kojoj piše „Već imate račun? Prijavite se ovdje“ u kojoj se nalazi poveznica (engl. *link*) koji korisnika preusmjerava do stranice za prijavu u sustav.

5.2.2. Početna stranica

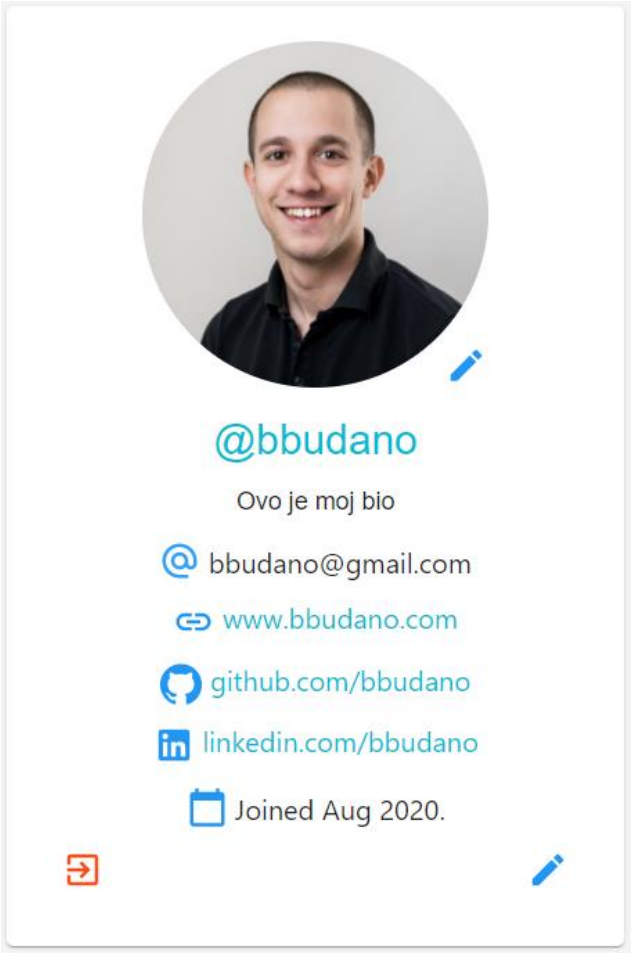


Slika 5.14. Početna stranica

Nakon uspješne prijave u sustav, korisnika se dovodi na početnu stranicu prikazanu na slici 5.14. Na lijevoj strani početne stranice korisnik može vidjeti objave, komentirati i *like*-ati. S desne se strane nalazi kartica koja predstavlja profil prijavljenog korisnika.

5.2.3. Korisnički profil

U kartici koja prikazuje korisnički profil prikazanoj na slici 5.15. nalazi se profilna slika, korisničko ime, opis profila, email adresa, web stranica, *Github* profil, *LinkedIn* profil te datum kada se korisnik registrirao u sustav. Osim navedenih informacija, na kartici profila se nalaze tri gumba. Uz profilnu sliku stoji gumb za uređivanje iste, u donjem lijevom kutu se nalazi gumb za odjavu iz aplikacije, a u donjem desnom kutu nalazi se gumb za uređivanje korisničkih podataka.



Slika 5.15. Kartica korisničkog profila

5.2.4. Profilna slika

Gumb uz profilnu sliku omogućuje korisniku promjenu svoje profilne slike. S obzirom da se profilne slike spremaju u *Firebase storage*, potrebno je koristeći *Yarn* alat instalirati *Firebase* ovisnost (engl. *dependency*) te konfigurirati klijentski dio aplikacije za spajanje s *Firebase*-om. Podaci za konfiguraciju poput API ključa nalaze se na stranici *Firebase* projekta. Konfiguracija za spajanje aplikacije s *Firebase*-om prikazana je na slici 5.16.

```
1  import firebase from 'firebase/app';
2  import 'firebase/storage';
3
4  const firebaseConfig = {
5      apiKey: "AIzaSyA1q_hQlSG5Lcjd-9azf2jY-yCLKPtUaPE",
6      authDomain: "social-student-react-upload.firebaseapp.com",
7      databaseURL: "https://social-student-react-upload.firebaseio.com",
8      projectId: "social-student-react-upload",
9      storageBucket: "social-student-react-upload.appspot.com",
10     messagingSenderId: "757244682109",
11     appId: "1:757244682109:web:68996326b99a7c21757614",
12     measurementId: "G-EF5FRWZFPE"
13 };
14
15 firebase.initializeApp(firebaseConfig);
16
17 const storage = firebase.storage();
18
19 export { storage, firebase as default };
```

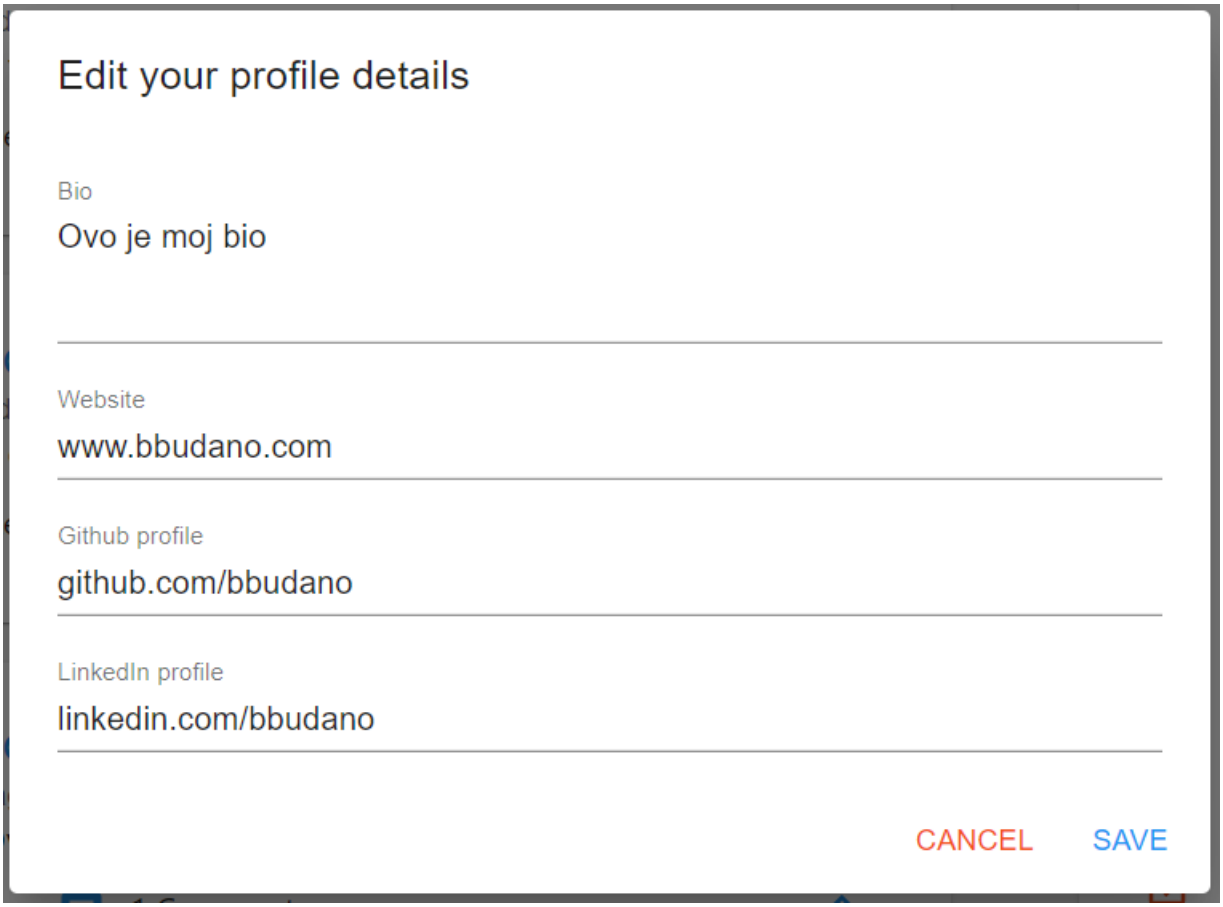
Slika 5.16. *Firebase* konfiguracija

Prije nego korisnik odabere profilnu sliku, na njegovom će se korisničkom profilu prikazivati zadana profilna slika prikazana na slici 5.17.

Slika 5.17. Zadana profilna slika

5.2.5. Korisnički podaci

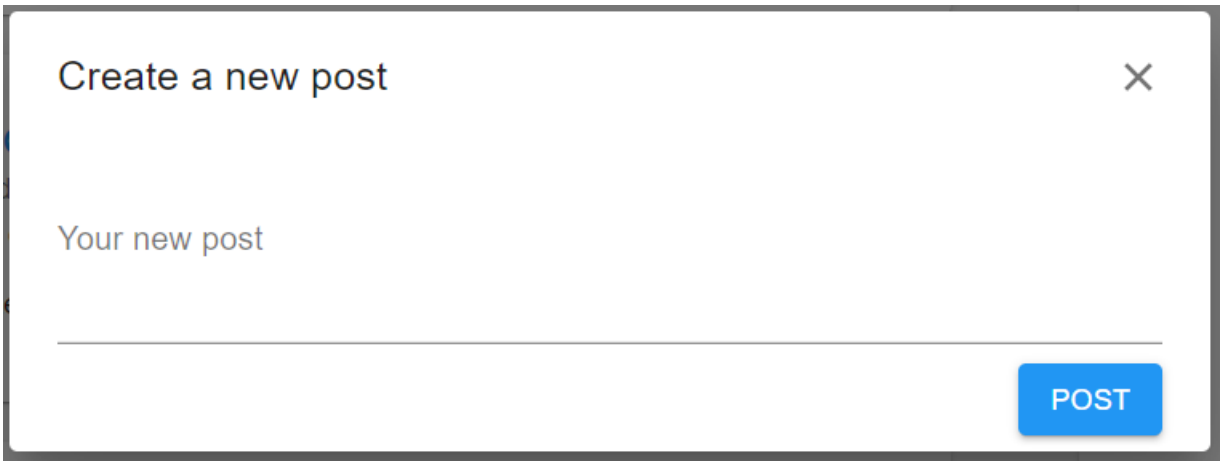
Kada korisnik klikne na gumb za uređivanje korisničkih podataka, otvara se dijalog prikazan na slici 5.18. u kojem se nalazi obrazac za promjenu korisničkih podataka koji je popunjen s trenutnim korisničkim podacima. Korisnik može promijeniti sve podatke osim korisničkog imena i email adrese. Na dnu obrasca nalazi se gumb za spremanje promjena (gumb *Save*) te gumb za poništavanje promjena i zatvaranje dijaloga (gumb *Cancel*).



Slika 5.18. Obrazac za uređivanje korisničkih podataka

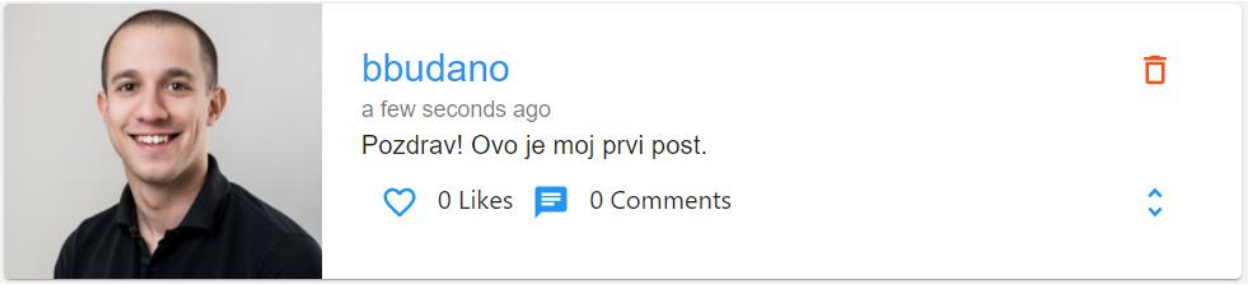
5.2.6. Objave

Klikom na gumb „+“ u navigacijskoj traci otvara se dijalog prikazan na slici 5.19. koji korisniku omogućuje kreiranje nove objave.



Slika 5.19. Dijalog za kreiranje nove objave

Unutar dijaloga nalazi se tekstualno područje u koje korisnik treba upisati sadržaj svoje nove objave te gumbovi za objavu i zatvaranje dijaloga. Klikom na gumb *Post*, korisnik kreira novu objavu prikazanu na slici 5.20. koja će korisnicima biti prezentirana na početnoj stranici.



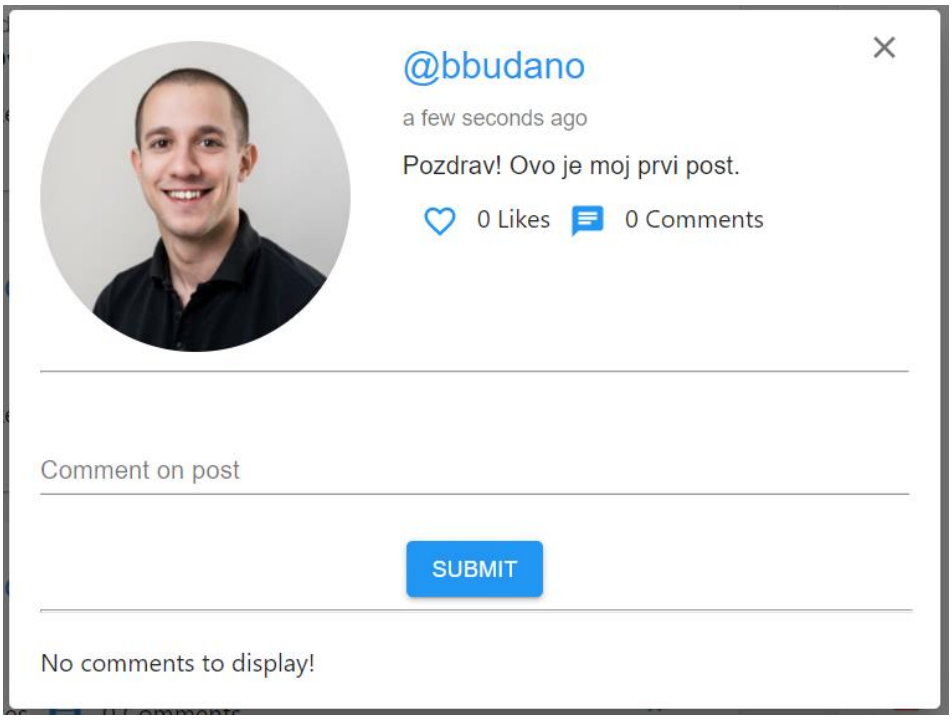
Slika 5.20. Objava

Objava ima izgled kartice čija je implementacija prikazana na slici 5.21. te sadrži profilnu sliku i korisničko ime autora objave, vrijeme kada je objava objavljena, sadržaj objave, broj *like*-ova i broj komentara pridruženih objavi. Osim navedenih atributa sadrži i nekoliko gumbova. Uz broj *like*-ova stoji gumb čija je funkcionalnost *like*-anje, odnosno *unlike*-anje objave. U gornjem desnom kutu se nalazi gumb za brisanje objave, on je vidljiv samo u slučaju da je autor objave trenutno prijavljeni korisnik ili ako trenutno prijavljeni korisnik ima administratorske privilegije. U donjem desnom kutu postoji gumb koji služi za prikazivanje detalja objave. Klikom na taj gumb otvara se dijalog koji sadrži sve detalje o objavi, polje za unos novog komentara te sve komentare vezane uz tu objavu.

```
65      return (
66          <Card className={classes.card}>
67              <CardMedia
68              className={classes.image}
69              image={avatar}
70              title="Profile image" />
71              <CardContent className={classes.content}>
72                  <Typography
73                      variant="h5"
74                      component={Link}
75                      to={`/users/${author}`}
76                      color="primary"
77                  >{author}</Typography>
78                  {deleteButton}
79                  <Typography variant="body2" color="textSecondary">{moment(postedOn).fromNow()}</Typography>
80                  <Typography variant="body1">{body}</Typography>
81                  <LikeButton postId={id} />
82                  <span>{likeCount} {likeCount === 1 ? "Like" : "Likes"}</span>
83                  <CustomButton tip="Comments">
84                      <ChatIcon color="primary" />
85                  </CustomButton>
86                  <span>{commentCount} {commentCount === 1 ? "Comment" : "Comments"}</span>
87                  <PostDialog postId={id} username={username} />
88              </CardContent>
89          </Card>
90      )
```

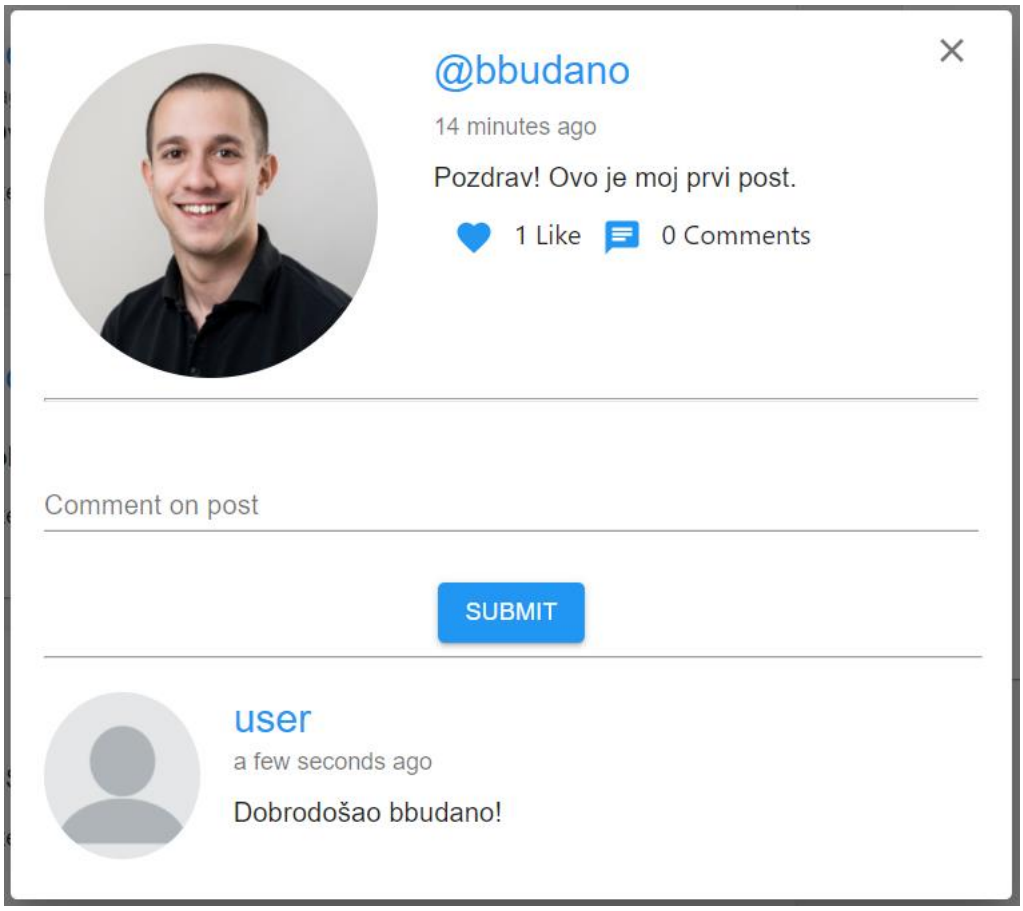
Slika 5.21. Implementacija kartice objave

Ukoliko ne postoji niti jedan komentar vezan za objavu, unutar dijaloga prikazanog na slici 5.22. stoji odgovarajuća poruka sve dok netko ne komentira na tu objavu.



Slika 5.22. Dijalog koji sadrži detalje o objavi (bez komentara)

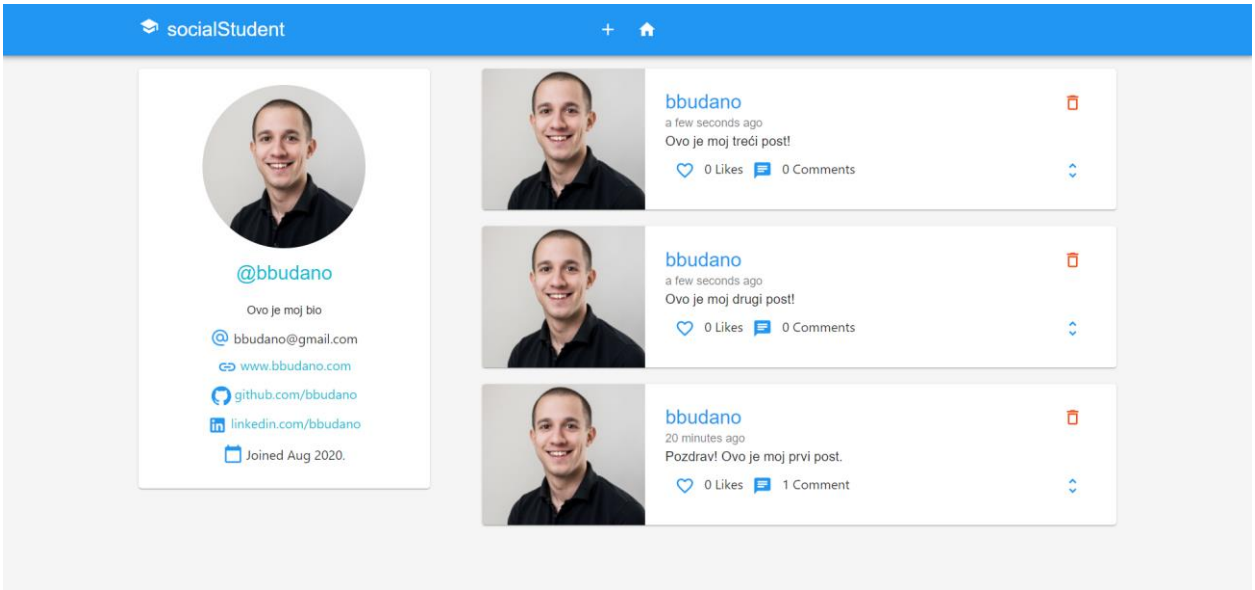
Nakon što korisnici komentiraju na objavu, njezini će komentari biti izlistani jedan ispod drugog te poredani po datumu i vremenu objave od zadnjeg prema prvom. Dijalog objave koja sadrži komentar prikazan je na slici 5.23.



Slika 5.23. Dijalog koji sadrži detalje o objavi

5.2.5. Stranica korisničkog profila

Na kartici objave, u dijalogu koji sadrži detalje objave te u komentarima, korisničko ime autora ima oblik *link*-a. Klikom na taj *link* korisnik može pristupiti stranici profila korisnika s tim korisničkim imenom koja je prikazana na slici 5.24.

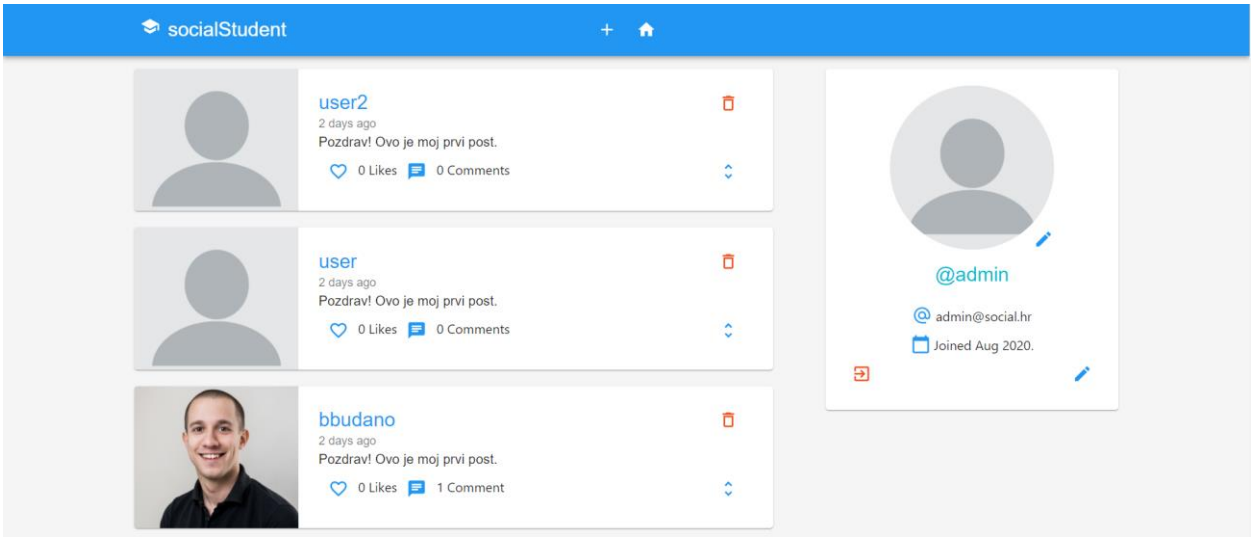


Slika 5.24. Stranica korisničkog profila

Nakon klika na korisničko ime određenog korisnika, korisnik je preusmjeren na novu stranicu u kojoj se s lijeve strane nalazi kartica koja predstavlja profil tog korisnika, a s desne su strane izlistane objave samo tog korisnika. Korisnik ovdje također ima mogućnost *like*-anja i komentiranja objava, a ako se nalazi na svom profilu ima mogućnost i brisanja objava.

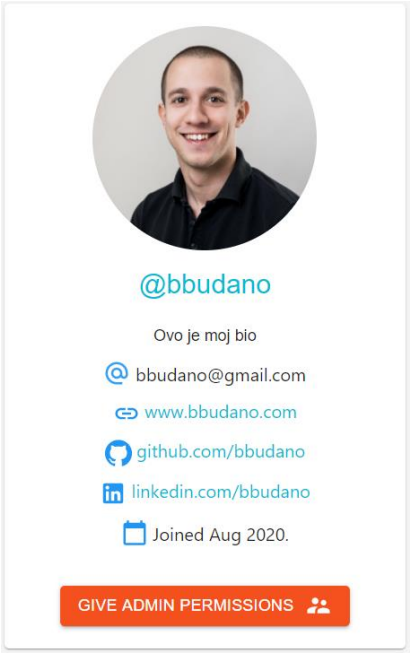
5.2.6. Administratorske mogućnosti

Ukoliko se u sustav prijavi korisnik koji ima administratorske privilegije, otvorene su mu dodatne mogućnosti.



Slika 5.25. Početna stranica (administrator)

Administrator ima dozvolu brisati sve objave, bez obzira je li on autor tih objava ili ne. Iz tog će razloga administratoru, na početnoj stranici ili na korisničkim profilima, unutar svake objave biti prikazan gumb za brisanje objave, što je prikazano na slici 5.25. Administrator također ima mogućnost dodijeljivanja administratorskih privilegija drugim korisnicima te oduzimanje istih. To može postići odlaskom na korisnički profil željenog korisnika. Ukoliko željeni korisnik nema administracijske privilegije, na dnu njegovog profila nalazi se gumb na kojem piše „Daj administracijska prava“ (engl. *Give admin permissions*), prikazan na slici 5.26.



Slika 5.26. Gumb za dodijeljivanje administracijskih prava

Ukoliko željeni korisnik ima administracijske privilegije, na dnu njegovog profila se nalazi gumb na kojem piše „Oduzmi administracijska prava“ (engl. *Take admin permissions*), prikazan na slici 5.27.

Slika 5.27. Gumb za oduzimanje administracijskih prava

6. ZAKLJUČAK

U ovom su završnom radu opisane i objašnjene sve funkcionalnosti i metode izrade web aplikacije Portal za studente. Portal za studente poboljšava komunikaciju među studentima te znatno olakšava postavljanje upita uz brzo dobivanje odgovora. Ova web aplikacija također demonstrira jednostavno korištenje i laku izradu poslužiteljskog dijela aplikacije korištenjem *Spring* programskog okvira te klijentske strane aplikacije korištenjem *React JavaScript* programskog okvira. Razvoj aplikacije znatno su olakšali mnogi korišteni alati. *IntelliJ IDEA* razvojno okruženje pruža odličan pregled strukture projekta, dovršavanje linija koda te ispravljanje grešaka. *Postman* alat olakšao je dizajniranje i razvoj kompletnog API-ja mogučnošću testiranja API-ja prije razvoja klijentskog dijela aplikacije. API dokumentacija je izgrađena zajedno s aplikacijom korištenjem *Swagger* sučelja. *PgAdmin 4* korisničko sučelje pruža vizualnu reprezentaciju baze podataka i podataka unutar nje te omogućava testiranje upita na bazu prije nego se implementiraju u poslužiteljski dio aplikacije. Aplikacija je funkcionalna i radi u lokalnom okruženju.

LITERATURA

[1] Introducing Facebook Campus, https://about.fb.com/news/2020/09/introducing-facebook-campus/, rujan 2020.

[2] IntelliJ IDEA Overview, https://www.jetbrains.com/help/idea/discover-intellij-idea.html#IntelliJ-IDEA-supported-languages, rujan 2020.

[3] What is Java technology and why do I need it?, https://www.java.com/en/download/faq/whatis_java.xml, rujan 2020.

[4] Java Documentation, https://docs.oracle.com/en/java/, rujan 2020.

[5] Gradle User Manual, https://docs.gradle.org/current/userguide/userguide.html, rujan 2020.

[6] Hibernate ORM, https://hibernate.org/orm/, rujan 2020.

[7] Hibernate (framework), https://en.wikipedia.org/wiki/Hibernate_(framework) , rujan 2020.

[8] The Java EE 6 Tutorial, https://docs.oracle.com/javaee/6/tutorial/doc/bnacj.html#bnadb, rujan 2020.

[9] Spring Framework, https://en.wikipedia.org/wiki/Spring_Framework, rujan 2020.

[10] Spring Boot, https://spring.io/projects/spring-boot, rujan 2020.

[11] A Comparison Between Spring and Spring Boot, https://www.baeldung.com/spring-vs-spring-boot, rujan 2020.

[12] Web MVC framework, https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference /html/mvc.html, rujan 2020.

[13] Spring Data, https://spring.io/projects/spring-data, rujan 2020.

[14] Spring Data JPA, https://spring.io/projects/spring-data-jpa, rujan 2020.

[15] Spring Security, https://spring.io/projects/spring-security, rujan 2020.

[16] Introduction to JSON Web Tokens, https://jwt.io/, rujan 2020.

[17] What is Swagger?, https://swagger.io/docs/specification/2-0/what-is-swagger/, rujan 2020.

[18] Getting Started, https://code.visualstudio.com/docs, rujan 2020.

[19] What is JavaScript?, https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript, rujan 2020.

[20] Introduction, https://yarnpkg.com/getting-started, rujan 2020.

[21] React (web framework), https://en.wikipedia.org/wiki/React_(web_framework), rujan 2020.

[22] Getting Started with Redux, https://redux.js.org/introduction/getting-started, rujan 2020.

[23] About, https://www.postgresql.org/about/, rujan 2020.

[24] pgAdmin 4, https://www.pgadmin.org/docs/pgadmin4/4.23/index.html, rujan 2020.

[25] Defining JPA Entities, https://www.baeldung.com/jpa-entities, rujan 2020.

[26] Spring Boot @Repository, http://zetcode.com/springboot/repository/, rujan 2020.

[27] Službena Java dokumentacija, https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html, rujan 2020.

[28] Službena Spring Data dokumentacija, https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Pageable.html, rujan 2020.

[29] Službena Spring Data dokumentacija, https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Page.html, rujan 2020.

[30] @Component vs @Repository and @Service in Spring, https://www.baeldung.com/spring-component-repository-service, rujan 2020.

[31] Guide to Spring @Autowired, https://www.baeldung.com/spring-autowire, rujan 2020.

[32] Quick Guide to Spring Controllers, https://www.baeldung.com/spring-controllers, rujan 2020.

[33] The Spring @Controller and @RestController Annotations, https://www.baeldung.com/spring-controller-vs-restcontroller, rujan 2020.

[34] Representational state transfer, https://en.wikipedia.org/wiki/Representational_state_transfer, rujan 2020.

SAŽETAK

Web aplikacija Portal za studente omogućuje korisnicima jednostavno postavljanje pitanja svim drugim registriranim korisnicima te lako odgovaranje na pitanja drugih korisnika. Aplikacija je intuitivna i jednostavna za korištenje. Korisnici mogu imati ulogu standardnog korisnika ili ulogu administratora. Uloga administratora korisniku pruža dodatne mogućnosti na web aplikaciji Portal za studente. Za izradu poslužiteljskog dijela aplikacije korišten je *Java* programski jezik i *Spring* programski okvir. Za izradu klijentskog dijela aplikacije korišten je *React JavasScript* programski okvir. Aplikacija koristi *PostgreSQL* bazu podataka.

Ključne riječi: Portal za studente, *Java, Spring, JavaScript, React*

ABSTRACT

Web application – Portal for students

The Portal for students web application allows users to easily ask questions to all other registered users and easily answer other user's questions. The application is intuitive and easy to use. Users can have a standard user role or an administrator role. The administrator role provides the user with additional features on the Portal for students web application. The Java programming language and the Spring framework were used to create the server side of the application. The React Javascript programming framework was used to create the client side of the application. The application uses a PostgreSQL database.

Keywords: Portal for students, Java, Spring, JavaScript, React

ŽIVOTOPIS

Zovem se Bernard Budano. Rođen sam 8.4.1996. u Osijeku. Prebivalište mi je na adresi Zagorska 5, 31000 Osijek. Godine 2015. završio sam Opću gimnaziju u Osijeku. Godine 2017. upisao sam stručni studij, smjer Informatika na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku na kojem trenutno pohađam 3. godinu. Od 7. mjeseca 2019. do 2. mjeseca 2020. godine bio sam zaposlen u Adcon Group d.o.o. tvrtci kao student, a od 2. mjeseca do 2020. godine zaposlen sam u tvrtci Base58 kao student. Tečno govorim i razumijem engleski jezik te poznajem i osnove njemačkog jezika. Posjedujem vozačku dozvolu B kategorije.