DRUŠTVENA MREŽA ZA LJUBITELJE PIVA

Hladek, Vladimir

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

DRUŠTVENA MREŽA ZA LJUBITELJE PIVA

Završni rad

Vladimir Hladek

Osijek, 2022

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 19.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Vladimir Hladek
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4352, 22.07.2019.
OIB Pristupnika:	72974204347
Mentor:	Prof. dr. sc. Krešimir Nenadić
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Društvena mreža za ljubitelje piva
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Kratko opisati osnovne funcionalnosti društvene mreže. Objasniti kako se navedene funkcionalnosti mogu realizirati web tehnologijama. Potrebno je modelirati i izraditi bazu podataka za potrebe web aplikacije. Opisati postupak izrade web aplikacije koju će koristiti ljubitelji piva. Omogućiti korisnicima pretragu pića prema zadanim kriterijima. Tema rezervirana: Vladimir Hladek
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	19.09.2022.
Datum potvrde ocjene od strane Odbora:	21.09.2022.

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

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.09.2022.

Ime i prezime studenta:	Vladimir Hladek
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4352, 22.07.2019.
Turnitin podudaranje [%]:	5

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

izrađen pod vodstvom mentora Prof. dr. sc. Krešimir Nenadić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Posljednjih godina u Hrvatskoj se sve više ljudi počinje interesirati za zanatska *(*engl. *craft)* piva. Za razliku od velikih pivovara koja prave velike količine piva koja služe samo za osvježiti i ublažiti žeđ, *craft* pivovare su male pivovare koje korištenjem tradicionalnih sastojaka nastoje napraviti piva određenih okusa i aroma kako ne bi samo ublažili žeđ, nego kako bi i ljubitelji piva mogli uživati i iskusiti razne okuse koji su često izraženiji i bolji od piva iz velikih pivovara.

Zbog sve većeg interesa za pivom, ono je sve češće i tema svakodnevnih razgovora. Kako su društvene mreže jedno od glavnih sredstava komunikacije i rasprava, cilj ovog rada je napraviti društvenu mrežu za ljubitelje piva na kojoj će glavna i jedina tema razgovora biti pivo i sve vezano za pivo. Ova web aplikacija nudi mogućnost kreiranja vlastitog korisničkog računa s kojim se može pristupiti društvenoj mreži. Omogućuje korisniku objavljivanje objava koje mogu sadržavati tekst i slike. Također omogućuje ostavljanje komentara na objavama i pozitivno ocjenjivanje objave. Svaki korisnik može urediti i svoj profil dodavanjem teksta i odabiranjem vrsta piva koja mu se sviđaju kako bi i drugi korisnici znali s kime mogu pričati o kojim vrstama piva. Za korisnike koji ne znaju kakvo bi im se pivo moglo svidjeti mogu na temelju karakteristika piva koja im se sviđaju ili na temelju vrsta piva koje već znaju da im se sviđaju pronaći one vrste koje bi im se mogle svidjeti.

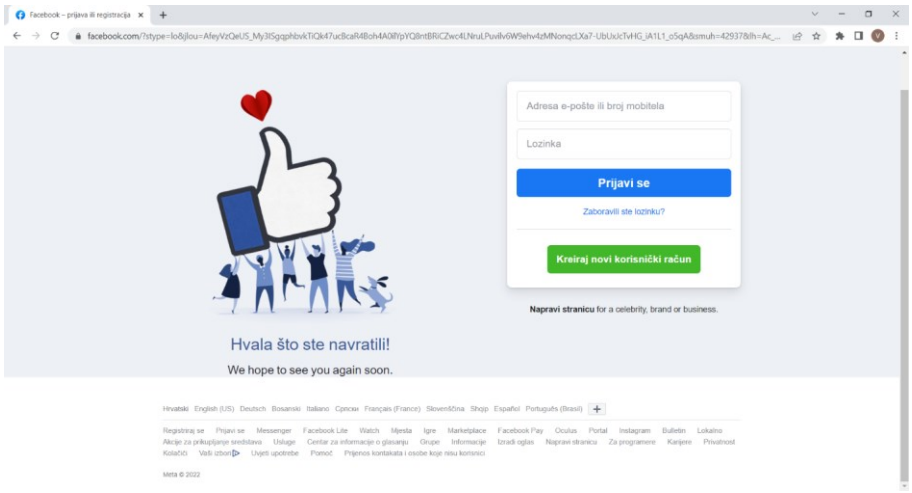
Tekst rada se sastoji od 5 poglavlja. Nakon kratkog uvoda u prvom poglavlju, u drugom su pregledane slične aplikacije, a u trećem su opisane korištene tehnologije u izradi ove web aplikacije. U četvrtom poglavlju je opisan razvoj web aplikacije te način i razlog primjene pojedinih korištenih tehnologija. Peto poglavlje prikazuje izgled napravljene web aplikacije, a šesto poglavlje je sažetak rada.

1.1. Zadatak završnog rada

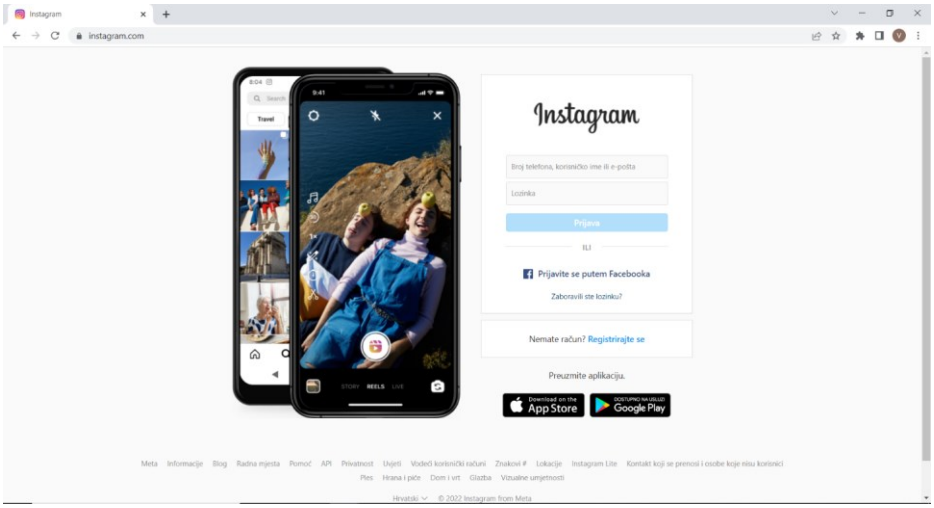
Kratko opisati osnovne funkcionalnosti društvene mreže. Objasniti kako se navedene funkcionalnosti mogu realizirati web tehnologijama. Potrebno je modelirati i izraditi bazu podataka za potrebe web aplikacije. Opisati postupak izrade web aplikacije koju će koristiti ljubitelji piva. Omogućiti korisnicima pretragu pića prema zadanim kriterijima.

2. PREGLED SLIČNIH APLIKACIJA

Na internetu je dostupno puno različitih društvenih mreža koje su većinom namijenjene širem i raznolikijem spektru korisnika. Među popularnije društvene mreže takvog tipa spadaju Facebook [1], Twitter [2] i Instagram [3], koji su prikazani na slikama 2.1., 2.2. i 2.3.. Na tim društvenim mrežama korisnici mogu dijeliti svoje mišljenje i raspravljati o raznim temama, ali te društvene mreže nisu prvenstveno usmjerene na rasprave o pivu.



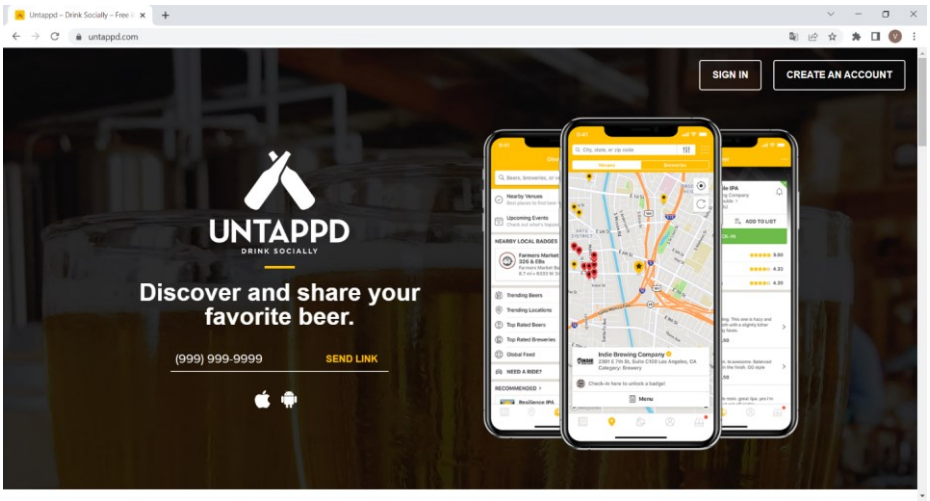
Slika 2.1. *Društvena mreža Facebook*



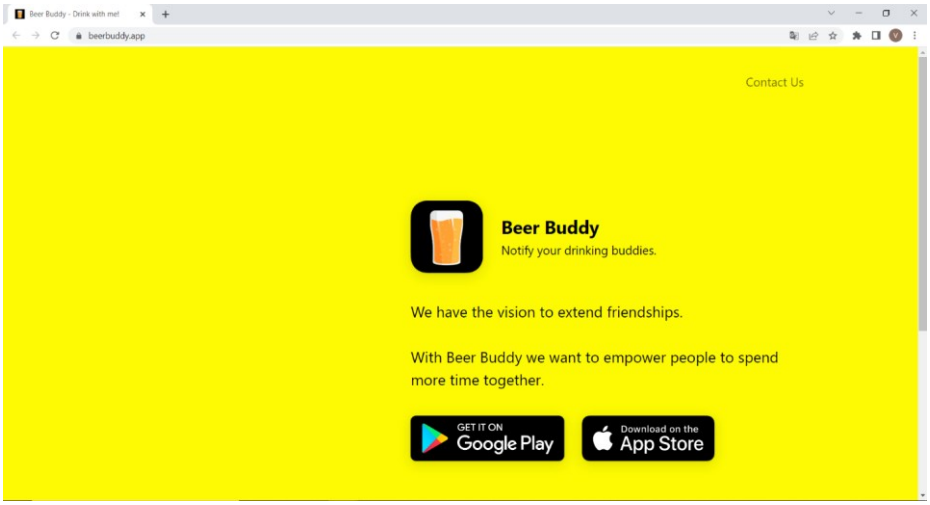
Slika 2.2. *Društvena mreža Instagram*

Slika 2.3. *Društvena mreža Twitter*

Društvene mreže koje nisu poznate kao prijašnje navedene, ali su za razliku od njih usmjerene prvenstveno na pivo su Untappd [4] i Beer Buddy [5], koji su prikazani na slikama 2.4. i 2.5.. Iako se koriste samo kao aplikacije na mobitelu i više se koriste za pronalaženje novih pivovara nego za rasprave, jedne su od rjeđih društvenih mreža koje su usmjerene na pivo.



Slika 2.4. *Društvena mreža Untappd*



Slika 2.5. *Društvena mreža Beer Buddy*

3. PREGLED KORIŠTENIH TEHNOLOGIJA

Komentirano [KN1]: Nedostaje poglavlje - Pregled sličnih rješenja/aoplikacija s najmanje 5 rješenja/aplikacija - ubaciti to novo poglavlje kao 2.

Za izradu ove web aplikacije koriste se osnovni jezici korišteni za izradu web aplikacije kao što su HTML, CSS i JavaScript. Osim navedenih bitno je navesti da se koristi React, JavaScript biblioteka za izradu korisničkih sučelja. S poslužiteljske strane se koristi MongoDB i GraphQL.

3.1. HTML

HyperText Markup Language je osnovni jezik koji upotrebljava se za izradu web stranica. HTML nije programski jezik nego jezik za označavanje koji opisuje pregledniku izgled stranice i omogućuje dosljedan prikaz web stranice klijentu. HTML dokument koji web preglednik prikazuje klijentu sastoji se od oznaka koje definiraju web pregledniku kako treba prikazati pojedine elemente HTML dokumenta [6].

Komentirano [KN2]: za označavanje

3.2. CSS

Zbog ljepšeg i detaljnijeg dizajniranja web stranice uz HTML se često koristi i stilski jezik CSS (engl. *Cascading Style Sheet*). CSS se ne mora zapisivati direktno unutar HTML dokumenta nego omogućuje odvajanje prezentacije stranice od samog sadržaja u zaseban dokument što omogućuje lakši i pregledniji pristup uređivanju pojedinih elemenata stranice [7].

Komentirano [KN3]: ovako treba biti

3.3. JavaScript

S obzirom na to da HTML i CSS određuju samo statična svojstva prikaza web aplikacije, a često je potrebno da web aplikacija ima i određene funkcionalnosti, uz HTML i CSS počinje se koristiti i JavaScript. JavaScript je skriptni jezik koji omogućuje dinamička ponašanja i interaktivnost stranice. Zbog mnogih funkcionalnosti ove web aplikacije korištenje JavaScript jezika je neizbježno. Kako bi se lakše implementirao JavaScript jezik koriste se razne biblioteke, a u slučaju ove aplikacije koristi se React [8].

3.4. React

React je JavaScript biblioteka za izradu korisničkih sučelja. React je razvio Facebook te je omogućio njegovo korištenje 2013. godine. Zbog svoje jednostavnosti, brzine pisanja, mogućnosti višestrukog korištenja pojedinih elemenata i velike podrške jedna je od najčešće korištenih biblioteka u razvoju web aplikacija. Reactom se web aplikacije izrađuju tako što se pojedini elementi i komponente izrađuju zasebno u JavaScript dokumentima koji imaju definiranu funkciju koja vraća HTML element. Taj HTML element je napisan posebnom sintaksom koja se zove JSX[1] i koja povezuje JavaScript i HTML. Ta funkcija se izvodi (engl. *export*) iz dokumenta te se uvodi

Komentirano [KN4]: ? - zamjena redoslijeda

[1] JavaScript XML

(engl. *import*) unutar *App.js* dokumenta u kojem se može pozvati kao HTML element. *App.js* je glavni dokument u kojem se sastavlja web aplikacija dodavanjem pojedinih elemenata. Kako bi se olakšalo dizajniranje pojedinih elemenata i poboljšao izgled web aplikacije koriste se još i biblioteke *semantic-ui-react* i *react-router-dom* koje su objašnjene u sljedećim potpoglavljima [9].

Komentirano [KN5]: ?

Komentirano [KN6]: Pasiv

Komentirano [KN7]: pasiv

3.5. Semantic UI React

Semantic UI je razvojni okvir (engl. *framework*) sličan *bootstrapu*. Sadrži unaprijed izgrađene semantičke komponente uz pomoć kojih se lagano i brzu mogu napraviti web stranice. Još jedna velika prednost ovog razvojnog okvira je mogućnost integracije u druge razvojne okvire i biblioteke [10]. Za izradu ove web aplikacije koristi se njegova React integracija nazvana *Semantic UI React*.

Komentirano [KN8]: ?

Komentirano [KN9]: Ovako se piše samo ako je ispred akronim (skraćenica) sa znakom '-' (crtica)

3.6. React Router

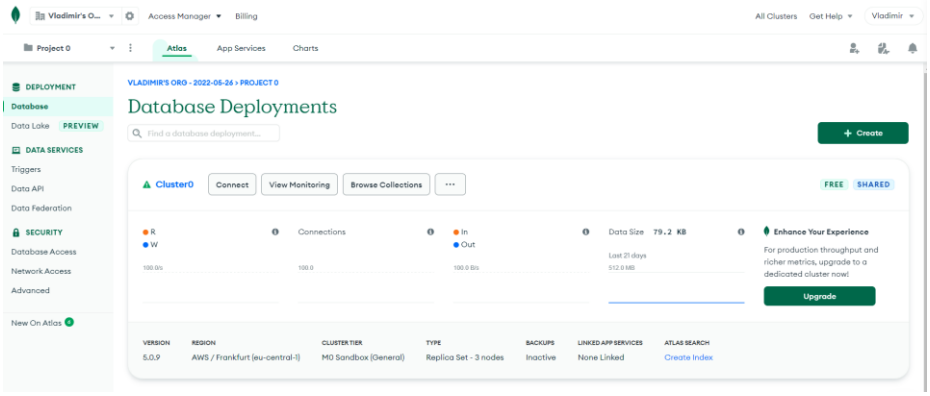
React Router je najpopularnija React biblioteka za usmjeravanje. Omogućuje mijenjanje URL[2]-a i osigurava prikaz stranice sukladan trenutnom URL-u. React Router je prikladan i pri izradi mobilnih aplikacija koristeći React, ali s obzirom na to da se u ovom slučaju radi o web aplikaciji bolje je koristiti React Router DOM, koji osim običnog React Routera sadrži i dodatne mogućnosti i poboljšanja za rad na pretraživačima [11].

Komentirano [KN10]: Veliku večinu fusnota treba prepraviti u navođenje referenci - [ ]

3.7. MongoDB

MongoDB je dokumentno orijentirana NoSQL[3] baza podataka za pohranu veće količine podataka. Za razliku od tradicionalnih relacijskih baza podataka koje koriste tablice i retke, MongoDB koristi zbirke i dokumente [12]. Osim što je jednostavna za naučiti i primijeniti ova baza podataka ima još neka svojstva i prednosti zbog čega je dobar izbor za potrebe ove web aplikacije. Nadalje, kao što slika 3.1. prikazuje, MongoDB omogućuje prikaz kretanja memorije u i izvan klastera, odabir IP[4] adresa koje će imati pristup uređivanju baze podataka, odabir korisnika i interakcije koje oni mogu imati s bazom podataka (npr. samo čitanje ili čitanje i pisanje u bazi podataka), pregled dokumenata spremljenih u bazu podataka i još neke druge funkcionalnosti koje su manje važne za izradu ove web aplikacije.

[2] Uniform Resource Locator (hrv. ujednačeni lokator resursa)
[3] No Structured Query Language (hrv. ne strukturiran jezik upita)
[4] Internet Protocol (hrv. internetski protokol)



Slika 3.1. *Kretanje podataka unutar klastera i izbornika MongoDB Atlasa*

3.8. GraphQL

GraphQL je upitni jezik (engl. *query language*) za API[5]-ove i vrijeme izvođenja za ispunjavanje upita s postojećim podacima. GraphQL omogućuje klijentu dohvaćanje točno određenih podataka koji su potrebni, bez potrebe za dohvaćanjem svih ostalih podataka. Zbog jednostavne primjene GraphQL je lagan za korištenje u izradi vanjskog sučelja (engl. *Front-end*). Jednostavnost njegove primjene je u tome što ima sintaksu sličnu JSON[6] dokumentu što je prikazano na slikama 3.2 i 3.3. na kojima se dohvaćaju podaci o dokumentu tipa *human* sa zadanim *id-om*.

```
{
  human(id: "1000") {
    name
    location
  }
}
```

Slika 3.2. *Sintaksa GQL[7] zahtjeva*

[5] Application Programming Interface (hrv. Programsko sučelje aplikacije)
[6] JavaScript Object Notation (hrv. JavaScript objektna notacija)
[7] GraphQL

Komentirano [KN11]: Format prema uputama na MAK portalu
- referirati se na shake silk u tekstu prije slike

Komentirano [KN12]: Formatirati prema uputama

Komentirano [KN13]: Za svaki akronim koje se prvi puta koristi navesti značenje - može i u fusnotu ili u zagradi

Komentirano [KN14]: JSON - akronim

Komentirano [KN15]: Sve slike (i ostali objekti u tekstu) trebaju biti referencirani u samom tesktu - pozvati se na njih - npr. 'kao što je prikazano na slici ...'

```
{
  "data": {
    "human": {
      "name": "Dorothy,
      "location": "Kansas"
    }
  }
}
```

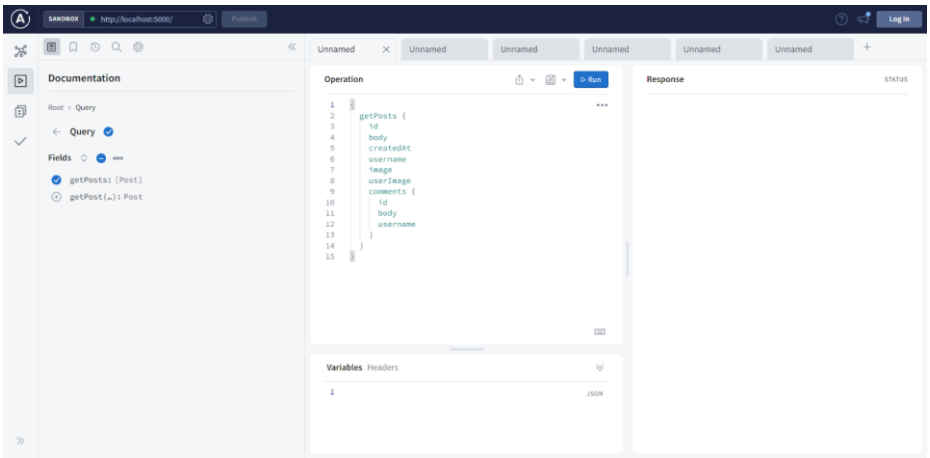
Slika 3.3. *JSON rezultat na navedeni zahtjev*

3.9. Apollo Server

Apollo Server je GraphQL poslužitelj koji je kompatibilan s bilo kojim GraphQL klijentom [13]. Prednosti Apollo Servera su jednostavno postavljanje i mogućnost testiranja upita i mutacija na lokalnoj web stranici kao što je prikazano na slici 3.4..

Komentirano [KN16]: Pregledati cijeli tekst rada i ispraviti 's' i 'sa'

Komentirano [KN17]: Slika dolje nema oznaku, opis niti referiranje na sliku



Slika 3.4. *Stranica Apollo Servera na kojoj se testiraju upiti i mutacije*

4. RAZVOJ WEB APLIKACIJE

U ovom poglavlju je opisan proces razvoja web aplikacije počevši od baze podataka i načinu izmjene podataka između baze podataka i klijenta, a zatim su prikazane pojedine komponente i elementi te je objašnjena njihova svrha i način funkcioniranja.

4.1. Dizajn baze podataka

Komentirano [VH18]: Skrati

Prvi korak u izradi web aplikacije je dizajniranje baze podataka. Baza podataka se sastoji od tri različita tipa dokumenta: Korisnik (engl. *user*), Objava (engl. *post*) i Pivo (engl. *beer*). Svaki od njih ima svoje atribute te je u nastavku objašnjeno za svaki pojedini tip koje atribute sadrži i zašto.

Komentirano [KN19]: ?

4.1.1. Tip dokumenta *User*

Dokument tipa *User* sadrži *id*, jedinstveni atribut koji se dodjeljuje korisniku pri stvaranju dokumenta tipa *User*. Također sadrži atribut *email* koji korisnik unosi pri stvaranju računa, *token* koji se korisniku dodjeljuje dok je prijavljen na račun i potvrđuje da je korisnik prijavljen u nekom trenutku, korisničko ime (engl. *username*) koje je također jedinstveno za svakog korisnika i koristi se za prikazivanje autora objave i pri prijavi na postojeći račun. Atribut koji prikazuje vrijeme nastanka računa nazvan je *createdAt*, tijelo (engl. *body*) služi kao korisnikov opis u kojem korisnik može napisati što želi. Lista koja sadrži popis *id*-ova stilova piva koje je korisnik označio da mu se sviđaju nazvana je *likedBeersId*. Pomoću *likedBeersId* atributa dohvaćaju se podaci o stilovima piva koja se sviđaju korisniku.

4.1.2. Tip dokumenta *Post*

Dokument tipa *Post* sadrži jedinstveni *id*, koji je dodijeljen svakoj objavi prilikom njenog nastanka. Atributi koji povezuju *Post* s dokumentom tipa *User*, to jest s autorom objave, su korisnički id (nazvan *userId*) i *username*. Iako bi samo *userId* bio dovoljan da se pronađe korisničko ime autora, s obzirom na to da se korisničko ime stalno prikazuje s objavom, ono je spremljeno kao atribut objave kako bi se izbjegao veliki broj poziva na bazu podataka pri prikazivanju više objava od jednom. *Body* je atribut koji sadrži tekst objave koji korisnik sam napiše, a *createdAt* je atribut koji sadrži vrijeme nastanka objave. Komentari (engl. comments) i sviđanja (engl. *likes*) su posebni atributi koji sadrže tipove podataka Komentar (engl. *Comment*) i Sviđanje (engl. *Like*) koji imaju svoje atribute o kojima više piše u nastavku. Atributi koji prikazuju koliko objava ima oznaka sviđanja i komentara nazivaju se *likeCount* i *commentCount*.

Komentirano [VH20]: Molim Vas da provjerite ispravnost ovog zapisa

Komentirano [VH21]: Molim Vas da provjerite ispravnost ovog zapisa

Atribut *Like* također sadrži svoj jedinstveni *id* koji mu je dodijeljen pri njegovom nastanku, *createdAt* koji prikazuje vrijeme nastanka oznake sviđanja i *username* koji sadrži ime korisnika koji je pozitivno označio objavu.

Atribut *Comment* sadrži jedinstveni *id* koji mu je dodijeljen pri njegovom nastanku, *createdAt* koji prikazuje vrijeme nastanka komentara i *body* koji sadrži tekst komentara koji korisnik sam unosi. Komentar je sa svojim autorom povezan atributima *userId* i *username*. Kao i u slučaju s objavama, iako se *username* autora može dohvatiti s baze podataka s *userIdom*, ipak se odmah sprema kao atribut komentara kako bi se izbjegao veliki broj poziva s baze podataka.

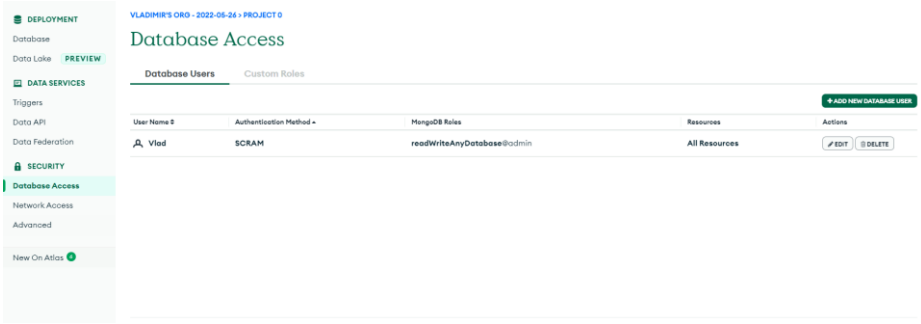
4.1.3. Tip dokumenta *Beer*

Za razliku od dokumenata tipa *User* i *Post*, dokumente tipa *Beer* može napraviti samo vlasnik baze podataka i to direktno na stranici MongoDB. *Beer* predstavlja stil piva i skoro svi njegovi atributi su karakteristike stila piva pomoću kojih se pretražuju pojedini stilovi. Dokument tipa *Beer* sadrži jedinstveni *id* i ime stila piva (atribut nazvan *styleName*). Atribut koji služi za svrstavanje pojedinih stilova u skupinu sličnih stilova, kao što su na primjer belgijski stilovi (engl. *Belgian-Styles*) piva te se koristi za opisivanje stila piva i kao pomoć pri pretraživanju stilova piva naziva se *beerClass*. Svi ostali atributi su karakteristike piva koje imaju specifične vrijednosti koje zapravo predstavljaju razine intenziteta pojedine karakteristike kao na primjer pitkost (engl. *drinkability*) koji može imati vrijednosti lagano (engl. *Easy*), srednje (engl. *Medium*) ili teško (engl. *Hard*). Ti atributi mogu imati više od jedne vrijednosti jer neki stilovi piva mogu imati malo drugačije karakteristike ovisno o proizvođaču ili se nalaze na granici između dvaju razina intenziteta.

Ostali atributi se mogu vidjeti u programskom kodu u prilogu.

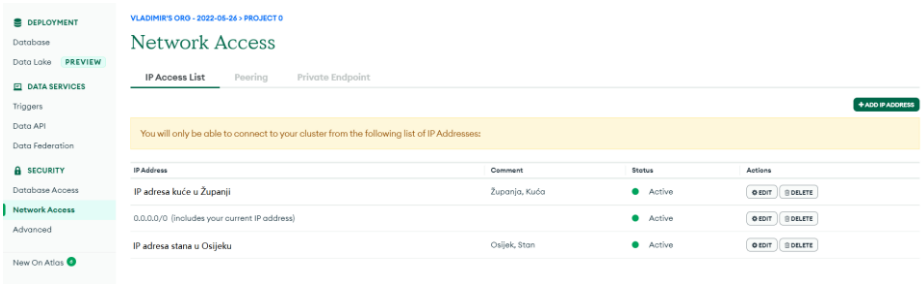
4.2. Postavljanje baze podataka i servera

Prije postavljanja baze podataka u MongoDB potrebno je napraviti račun za MongoDB Atlas. Nakon stvaranja računa stranica novog korisnika provede kroz određena pitanja preko kojih se određuje oblak (engl. *cloud*) koji će se koristiti, server i usluge koje će biti ponuđene, te na temelju odabira MongoDB napravi klaster (skupina nezavisno djelujućih elemenata povezanih nekim medijem u cilju koordiniranog i kooperativnog ponašanja [14]) po određenoj cijeni koji će se koristiti za web aplikaciju. Kako bi kao vlasniku bilo omogućeno uređivanje baze, potrebno je napraviti korisnika klastera koji može čitati i pisati podatke, što se čini ispunjavajući anketu koja se dobije pritiskom na gumb za dodavanje novog korisnika baze podataka (engl. *add new database user*) unutar dijela stranice za pristup bazama podataka (engl. *database access*) koji je prikazan na slici 4.1..



Slika 4.1. *Dio MongoDB Atlas stranice na kojoj se dodavaju korisnici baze podataka*

Osim postavljanja korisnika koji će imati pristup bazi podataka, zbog dodatne sigurnosti, potrebno je postaviti i s kojih IP adresa se može pristupiti bazi podataka. IP adrese koje imaju pristup bazi podataka postavljaju se pritiskom gumba za dodavanje IP adrese (engl. *add IP address*) unutar dijela stranice za mrežni pristup (engl. *network access*).



Slika 4.2. *Dio MongoDB Atlas stranice na kojem se prikazuju i dodaju mreže koje imaju pristup bazi podataka*

Kao što se može vidjeti na slici 4.2. jedna od adresa koje imaju pristup je adresa *0.0.0.0/0* čijim dodavanjem se omogućuje uređivanje baze podataka sa svake IP adrese.

Nakon što se odredi tko ima pristup klasteru otvori se datoteka na računalu preko Visual Studio Codea u kojoj se izradi projekt. Za početak je potrebno napraviti *package.json* i *index.js* datoteke. Za stvaranje *index.js* datoteke može se odabrati opcija za izradu nove datoteke (engl. *new file*) u izborniku Visual Studio Codea, a za stvaranje *package.json* datoteke otvori se novi terminal u Visual Studio Codeu u kojem se pokreće naredba prikazana na slici 4.3.

Komentirano [KN22]: Koristiti PASIV

```
PS C:\Users\Vladimir\Desktop\Primjer> npm init -y
Wrote to C:\Users\Vladimir\Desktop\Primjer\package.json:
```

Slika 4.3. *Naredba za stvaranje package.json datoteku*

Nadalje, uz pomoć terminala instalira se *apollo-server*, *graphql* i *mongoose* kako bi se omogućilo povezivanje (konekcija) na bazu.

```
PS C:\Users\Vladimir\Desktop\Primjer> npm install apollo-server graphql mongoose
```

Slika 4.4. *Naredba za instaliranje apollo-server, graphql i mongoose*

Kako bi se mogao pokrenuti Apollo-server u *indeks.js* datoteku unese se kod naveden na slici 4.4.

```
const { ApolloServer } = require('apollo-server');
const typeDefs = require('./graphql/typeDefs')
const resolvers = require('./graphql/resolvers')
const { MONGODB } = require('./config.js');

const server = new ApolloServer({
    typeDefs,
    resolvers,
    context: ({ req }) => ({ req })
});
```

Slika 4.5. *Kod za stvaranje novog ApolloServera*

Kao što je prikazano na slici 4.5., pri izradi novog ApolloServera dodaju se određeni parametri koji su objašnjeni kasnije u ovom poglavlju. Nakon izrade novog ApolloServera povezuje ga se s bazom podataka uz kod prikazan na slici 4.6.

```
mongoose.connect(MONGODB, { useNewUrlParser: true}).then(() => {
    console.log('MongoDB Connected');
    return server.listen({ port: 5000 });
}).then(res => {
    console.log(`Server running at ${res.url}`)
});
```

Slika 4.6. *Spajanje Apollo Servera s bazom podatak i njegovo pokretanje*

typeDefs parametar koji se prema slici 4.5. unosi pri izradi ApolloServera sadrži definicije za pojedine tipove dokumenata, mutacije i upite koji se koriste u bazi podataka. Zbog bolje preglednosti i čitljivosti odvaja ga se u zasebnu datoteku *typeDefs.js*.

```
const { gql } = require('apollo-server');

module.exports = gql`
    type User{
        id: ID!
        email: String!
        token: String!
        username: String!
        createdAt: String!
    }
```

Slika 4.7. *Prikaz dijela koda datoteke typeDefs.js u kojem je definiran tip podatka User*

Prema slici 4.7. za definiranje tipova podataka koristi se graphQL koji ima sličnu sintaksu onoj kojom JSON prikazuje podatke. Ostali definirani dokumenti u *typeDefs.js* datoteci su prikazani u nadolazećim potpoglavljima kada se budu objašnjavali tipovi dokumenata, mutacije i upiti baze podataka.

Parametar razrješivači (engl. *resolvers*) koji je prikazan na slici 4.5. sadrži definirane upite i mutacije koje je moguće provesti s određenim tipovima podataka. Više o upitima i mutacijama spominje se kasnije u ovom poglavlju.

Argument *MONGODB* koji je prikazan na slici 4.7. je izvod podataka odvojenih u zasebnoj datoteci *config.js* jer sadrži zaporku korisnika koji ima pristup bazi podataka i tajni ključ koji se kasnije koristi za kriptiranje zaporke.

```
module.exports = {
    MONGODB: 'mongodb+srv://Vlad:<password>@cluster0.shx6t.mongodb.net/<ime_datoteke>?retryWrites=true&w=majority',
    SECRET_KEY: '<ključ>'
}
```

Slika 4.8. *Prikaz izmijenjenog koda unutar config.js datoteke*

Kao što je navedeno u opisu slike 4.8. slika je izmijenjena i umjesto polja *<password>*, *<ime_datoteke>* i *<ključ>* treba se unijeti zaporka, ime zbirke u bazi podataka u kojoj se radi projekt i tajni ključ.

4.3. Tipovi dokumenta, upiti i mutacije

S obzirom na to da korisnik ove web aplikacije cijelo vrijeme izmjenjuje podatke s bazom podataka potrebno je koristiti niz upita i mutacija. U ovom potpoglavlju su objašnjeni tipovi dokumenata koji se mogu spremiti u bazi podataka, upiti i mutacije vezani za pojedine tipove dokumenata, načini na koji se pozivaju upiti i mutacije s korisničke strane te se navodi gdje i zašto su upiti i mutacije korišteni.

4.3.1. *User*, *Post* i *Beer* dokumenti

User, *Post* i *Beer* dokumenti nastaju pomoću funkcije *model* u koju se kao argument predaje tip podatka *Schema* kao što je prikazano na slici 4.9. Programski kod za izradu *Post* i *Beer* nalazi se u prilogu.

```
const { model, Schema } = require('mongoose');

const userSchema = new Schema({
    username: String,
    password: String,
    email: String,
    createdAt: String,
    body: String,
    likedBeersId: [String]
});

module.exports = model('User', userSchema);
```

Slika 4.1. *Kod za izradu modela User*

4.3.2. *typeDefs* datoteka

Prije nego što se mogu koristiti tipovi dokumenata, upiti i mutacije s *GraphQLom* potrebno ih je definirati unutar *typeDefs.js* datoteke.

```
const { gql } = require('apollo-server');

module.exports = gql`
    type Post{
        id: ID!
        body: String!
        createdAt: String!
        username: String!
        comments: [Comment]!
        likes: [Like]!
        likeCount: Int!
        commentCount: Int!
    }
```

Slika 4.2. *Definicija dokumenata Post unutar typeDefs.js datoteke*

Prema slici 4.10. prikazan je kod za definiranje tipa dokumenata *Post* za daljnje korištenje u *GraphQLu.* Sve varijable čiji tip podatka završava s uskličnikom, kao na primjer *String!*, moraju biti ispunjene pri stvaranju dokumenta. Ako se tip podatka neke varijable nalazi unutar uglatih zagrada, kao na primjer *[Like]*, to znači da se varijabla sastoji od liste koja sadrži vrijednosti navedenog tipa.

Definicije ostalih tipova dokumenta mogu se vidjeti u programskom kodu u prilogu.

Na slici 4.11. prikazan je kod za definiranje raznih upita i mutacija koje se koriste u ovoj web aplikaciji. Može se primijetiti kako se za argument mutacije *register* koristi tip unosa *RegisterInput* i za argument poziva *getFilteredBeers* koristi se tip unosa *BeerInput* koji su također definirani unutar datoteke kako bi kod bio pregledniji. Kod argumenata koji su obvezni te upita i mutacija koje moraju imati povratnu vrijednost nakon tipa podatka piše uskličnik.

```
type Query{
    getPosts: [Post]
    getPost(postId: ID!): Post
    getUser(userId: ID!): User
    sayHi: String!
    getAllBeers: [Beer]
    getBeer(beerId: ID!): Beer
    getFilteredBeers(beerInput: BeerInput): [Beer]
}

type Mutation{
    register(registerInput: RegisterInput): User!
    login(username: String!, password: String!): User!
    createPost(body: String!): Post!
    deletePost(postId: ID!): String!
    createComment(postId: ID!, body: String!): Post!
    deleteComment(postId: ID!, commentId: ID!): Post!
    likePost(postId: ID!): Post!
    editBody(userId: ID!, userBody: String!): User!
    editLikedBeers(userId: ID!, newLikedBeersId: [String]!): User!
}
```

Slika 4.3. *Definicije upita i mutacija unutar typeDefs.js datoteke*

4.3.3. Autentifikacija korisnika

Za mnoge upite i mutacije je potrebno dohvatiti podatke o prijavljenom korisniku, zato je u ovom dijelu potpoglavlja, prije opisa upita i mutacija, objašnjen način autentifikacije korisnika koji se često koristi u ovoj web aplikaciji.

```
const { AuthenticationError } = require('apollo-server');

const jwt = require('jsonwebtoken');
const { SECRET_KEY } = require('../config');

module.exports = (context) => {
    const authHeader = context.req.headers.authorization;
    if( authHeader ){

        const token = authHeader.split('Bearer ')[1];
        if(token){
            try {
                const user = jwt.verify(token, SECRET_KEY);
                return user;
            } catch(err){
                throw new AuthenticationError('Invalid/Expired token');
            }
        }
        throw new Error('Authentication token must be \'Bearer [token]');
    }
    throw new Error('Authorization header must be provided');
}
```

Slika 4.4. *Kod za funkciju checkAuth*

Slika 4.12. prikazuje kod funkcije koja se često koristi za provjeru valjanosti tokena korisnika koji je kao argument predan funkciji. Ako je token neispravan ili nevažeći funkcija izbacuje odgovarajuću poruku greške.

```
import React, { useReducer, createContext} from "react";
import jwtDecode from 'jwt-decode';

const initialState = {
    user: null
}

if(localStorage.getItem("jwtToken")){
    const decodedToken = jwtDecode(localStorage.getItem("jwtToken"));
    if(decodedToken.exp * 1000 < Date.now()){
        localStorage.removeItem("jwtToken")
    } else{
        initialState.user = decodedToken;
    }
}

const AuthContext = createContext({
    user: null,
    login: (userData) => {},
    logout: () => {}
})
```

Slika 4.5. *Kod za stvaranje varijable AuthContext*

Kao što je prikazano na slici 4.13. postavlja se početno stanje korisnika kao *null* što predstavlja da korisnik nije trenutno prijavljen na web aplikaciji. Iz lokalnog spremnika se pokušava dohvatiti token koji se dekodira i dohvaća se vrijeme njegovog isteka, te u slučaju da je token istekao briše se iz lokalnog spremnika kako se ne bi mogao koristiti i nakon što je istekao. Varijabla *AuthContext* se stvara pomoću funkcije *createContext* uvedene iz *react* biblioteke uz čiju pomoć se podaci mogu koristiti u različitim komponentama koje su unutar *AuthContext.Provider* komponente. *AuthContext* sadrži podatke o korisniku i funkcije *login* i *logout* s kojima se mijenja vrijednost od *user*.

```
function authReducer(state, action){
    switch(action.type){
        case 'LOGIN':
            return{
                ...state,
                user: action.payload
            }
        case 'LOGOUT':
            return{
                ...state,
                user: null
            }
        default:
            return state;
    }
}
```

Slika 4.6. *Kod za funkciju authReducer*

Prema slici 4.14. *authReducer* ovisno o tome jeli korisnik prijavljen ili ne vraća određene podatke o prijavljenom korisniku ili *null* ako korisnik nije prijavljen.

```
function AuthProvider(props){
    const [state, dispatch] = useReducer(authReducer, initialState);

    function login(userData){
        localStorage.setItem("jwtToken", userData.token)
        dispatch({
            type: 'LOGIN',
            payload: userData
        })
    }

    function logout(){
        localStorage.removeItem("jwtToken")
        dispatch({
            type: 'LOGOUT'
        })
    }

    return (
        <AuthContext.Provider
            value={{ user: state.user, login, logout}}
            {...props}
        />
    )
}

export { AuthContext, AuthProvider}
```

Slika 4.7. *Kod za komponentu AuthProvider*

Prema slici 4.15. komponenta *AuthProvider* je komponenta *AuthContext.Provider* koja sadrži podatke o prijavljenom korisniku i funkcije *login* i *logout* za prijavljivanje i odjavljivanje korisnika, to jest za mijenjanje podataka o korisniku. *login* funkcija dohvaća podatke o korisniku koji se prijavio i dodjeljuje lokalnom spremniku token prijavljenog korisnika, a *logout* funkcija uklanja token iz lokalnog spremnika.

4.3.4. Upiti i mutacije vezane za *User*

Kao što slika 4.16. prikazuje, pozivu *getUser* se kao argument daje *userId* te, ako postoji korisnik s takvim *id*-om, poziv vraća podatke o korisniku koji ima odgovarajući *id*.

Ovaj upit se koristi za dohvaćanje i prikaz podataka o korisniku na njegovoj profilnoj stranici *Profile* o kojoj više piše kasnije u poglavlju.

```
module.exports = {
    Query: {
        async getUser(_, { userId }){
            try{
                const user = await User.findById(userId);
                if(user){
                    return user;
                } else {
                    throw new Error('User not found')
                }
            } catch(err){
                throw new Error(err);
            }
        }
    },
```

Slika 4.16. *Kod za poziv getUser*

Mutaciji *login* se kao argumenti daju korisničko ime (engl. *username*) i zaporka (engl. *password*). Prvo se provjerava ispravnost predanih argumenata uz pomoć funkcije *validateLoginInput* koja provjerava jesu li polja za unos korisničkog imena i zaporke ispunjeni, a u suprotnom vraća odgovarajuću poruku greške. Programski kod navedene funkcije nalazi se u prilogu. Nakon provjere ispravnosti unesenih podataka ispisuju se greške vezane za unos podataka ako ih ima, a ako ih nema provjerava se postoji li uneseno korisničko ime u bazi podataka i podudara li se unesena zaporka sa zaporkom tog korisničkog imena. Nakon provjere se stvara jedinstveni token koji vrijedi dok je korisnik prijavljen te se vraćaju podaci o korisniku, njegov jedinstven *id* i token.

Poziv *login* mutacije se koristi na *Login* stranici za prijavu o kojoj više piše kasnije u ovom poglavlju.

```
async register(_, {registerInput: { username, email, password, confirmPassword, body, likedBeersId }}){

    const {valid, errors} = validateRegisterInput( username, email, password, confirmPassword );
    if(!valid){
        throw new UserInputError('Error', { errors });
    }

    const user = await User.findOne({ username });
    if(user){
        throw new UserInputError('Username is taken', {
            errors: {
                username: 'This username is taken'
            }
        });
    }

    password = await bcrypt.hash(password, 12);

    const newUser = new User({
        email,
        username,
        password,
        createdAt: new Date().toISOString(),
        body,
        likedBeersId
    });
```

Slika 4.17. *Prvi dio koda za mutaciju register*

Komentirano [VH23]: Myb

Komentirano [KN24]: zaporka

Kao što slika 4.17. prikazuje, mutaciji *register* se kao argument predaje *registerInput* koji sadrži niz podataka potrebnih za stvaranje novog korisnika *User*. Ispravnost predanih argumenata se provjerava uz pomoć funkcije *validateRegisterInput* koja provjerava jesu li svi potrebni podaci uneseni, odgovara li zapis elektroničke pošte i podudaraju li se podaci uneseni u polja za *password* i *confirmPassword*. Ako su uneseni podaci ispravni, provjerava se postoji li već korisnik s istim korisničkim imenom u bazi podataka. Zatim se kriptira unesena zaporka i stvara se novi korisnik sa svim potrebnim podacima. Jedan od podataka je *createdAt* koji predstavlja vrijeme i datum nastanka koji uz pomoć funkcije *toISOString* podatak tipa *Date* sprema u obliku podatka tipa *String*.

Programski kod za *validateRegisterInput* nalazi se u prilogu.

```
74          const res = await newUser.save();
75
76          const token = generateToken(res);
77
78          return {
79              ...res._doc,
80              id: res._id,
81              token
82          };
```

Slika 4.18. *Drugi dio koda za mutaciju register*

Na slici 4.18. prikazano je da se novi korisnik sprema u bazu podataka s jedinstvenim *id*-om i dodjeljuje mu se jedinstveni token koji vrijedi dok je korisnik prijavljen.

Poziv *register* mutacije se koristi na *Register* stranici za registriranje o kojoj više piše kasnije u ovom poglavlju.

Mutacija *editBody* koristi se za mijenjanje *body* atributa. Navedenoj mutaciji se predaju argumenti za dohvaćanje korisnika čiji atribut se mijenja i vrijednost novog atributa. Nakon što se dohvate podaci o korisniku, provjerava se jeli dohvaćeni korisnik isti kao i onaj koji je prijavljen, čime se sprječava mogućnost da korisnik može promijeniti tuđi *body* atribut. Nakon što se potvrdi da je prijavljeni korisnik isti kao i korisnik čiji atribut se mijenja, postavlja mu se nova vrijednost na atribut *body*.

Mutacija *editLikedBeers* je slična mutaciji *editBody*. *EditLikedBeers* mutaciji se predaju argumenti za dohvaćanje korisnika i nova lista stilova piva koje se sviđaju korisniku. Ako je prijavljeni korisnik isti kao i korisnik čiji atributi se mijenjaju, nova lista se sprema u atribut *likedBeersId*.

Pozivi *editBody* i *editLikedBeers* mutacija se mogu koristiti samo ako je korisnik prijavljen i nalazi se na svojoj *Profile* stranici o kojoj više piše u nastavku ovog poglavlja.

Programski kodovi za mutacije *editBody* i *editLikedBeers* nalaze se u prilogu.

4.3.5. Upiti i mutacije vezane za *Post*

Upit *getPosts* dohvaća sve objave spremljene u bazi podataka i sortira ih po vremenu nastanka tako da prikaže prvo najnovije, a upit *getPost* dohvaća samo onu objavu na bazi podataka koja ima *id* koji odgovara onom koji je kao argument predan funkciji.

GetPosts upit se koristi na *Home* stranici za prikazivanje svih objava koje se nalaze u bazi podataka, a *getPost* upit se koristi na *SinglePost* stranici za prikaz određene objave.

Programski kod za navedene upite nalazi se u prilogu.

```
31      Mutation: {
32          async createPost(_, { body }, context){
33              const user = checkAuth(context);
34
35              if(body.trim() === ''){
36                  throw new Error('Post body must not be empty');
37              }
38
39              const newPost = new Post({
40                  body,
41                  user: user.id,
42                  username: user.username,
43                  createdAt: new Date().toISOString()
44              });
45
46              const post = await newPost.save();
47
48              return post;
49          },
```

Slika 4.19. *Kod za mutaciju createPost*

Prema slici 4.19. *CreatePost* mutaciji se kao argument predaje tijelo objave i podaci o korisniku. Ako u polje za tekst tijela objave nije uneseno ništa mutacija izbacuje poruku greške, a u suprotnom stvara novi *Post* s podacima vezanim za objavu i autora objave te se sprema u bazu podataka.

Mutaciji *deletePost* se kao argumente predaju jedinstveni *id* objave koja se pokušava obrisati i podaci o korisniku. Nakon što je u bazi podataka pronađena objava s odgovarajućim *id*-om provjerava se jeli prijavljeni korisnik koji pokušava obrisati objavu autor. *DeletePost* mutacija se koristi na *Home* i *SinglePost* stranicama kako bi korisnik mogao obrisati svoju objavu.

Komentirano [VH25]: myb

Mutacija *likePost* kao argument uzima *id* objave koju korisnik želi pozitivno označiti ili s koje želi maknuti svoju pozitivnu oznaku te podatke o prijavljenom korisniku. Nakon dohvaćanja objave s odgovarajućim *id*-om provjerava se jeli prijavljeni korisnik već pozitivno označio objavu. U slučaju da je već pozitivno označio bio objavu onda će ukloniti pozitivnu oznaku, a u suprotnom će ju pozitivno označiti. Podaci o korisničkim imenima koji su pozitivno označili objavu se spremaju unutar objave *Post* i pomoću te liste se provjerava jeli prijavljeni korisnik već pozitivno označio objavu.

Mutacija *likePost* se koristi na *Home* i *SinglePost* stranicama.

Programski kod za sve *deletePost* i *likePost* mutacije nalazi se u prilogu.

4.3.6. Upiti i mutacije vezane za komentare na objavama

Mutaciji *createComment* se kao argumenti predaju *postId*, koji predstavlja *id* objave na kojoj korisnik namjerava objaviti komentar, *body*, koji predstavlja tijelo komentara i podatke o prijavljenom korisniku. U slučaju da korisnik pokuša objaviti komentar bez unošenja teksta za tijelo komentara izbacit će se odgovarajuća poruka greške, u suprotnom, prema slici 4.48., dohvaća se objava sa zadanim *id*-om te u varijablu *comments* odgovarajućeg dokumenta *Post* stvara se novi komentar sa svim potrebnim podacima. Na kraju mutacije se na bazu podataka spremaju promjene na objavi. Programski kod ove mutacije se nalazi u prilogu.

createComment mutacija se koristi na *SinglePost* stranici kako bi korisnik mogao dodati komentar na objavu.

```
32      async deleteComment(_, { postId, commentId }, context){
33          const { username } = checkAuth(context);
34
35          const post = await Post.findById(postId);
36
37          if(post){
38              const commentIndex = post.comments.findIndex((c) => c.id === commentId);
39
40              if(post.comments[commentIndex].username === username){
41                  post.comments.splice(commentIndex, 1);
42                  await post.save();
43                  return post;
44              } else{
45                  throw new AuthenticationError('Action not allowed')
46              }
47          } else {
48              throw new UserInputError('Post not found');
49          }
50      }
```

Slika 4.20. *Kod za mutaciju deleteComment*

Prema slici 4.20. mutaciji *deleteComment* kao argumenti se predaju *postId*, koji predstavlja *id* objave na kojoj se nalazi komentar kojeg korisnik namjerava obrisati, *commentId*, koji predstavlja

Komentirano [VH26]: myb

jedinstveni *id* komentara koji korisnik namjerava obrisati i podatke o prijavljenom korisniku. Nakon dohvaćanja objave kojoj odgovara zadani *postId* provjerava se postoji li objava sa zadanim *id*-om te se pronalazi indeks komentara sa zadanim *id*-om komentara. Ako se podudaraju korisničko ime prijavljenog korisnika koji pokušava obrisati komentar s korisničkim imenom autora komentara, iz varijable *comments* u dokumentu *Post* će se obrisati komentar s odgovarajućim indeksom.

Ova mutacija se koristi na *SinglePost* stranici kako bi korisnik mogao obrisati komentar na objavu ako je on autor tog komentara.

4.3.7. Upiti vezani za *Beer*

Upit *getAllBeer* dohvaća sve stilove piva koji su spremljeni u bazi podataka.

Ovaj upit se koristi na stranici *Register* kako bi se prikazali stilovi piva koje korisnik može označiti da mu se sviđaju i na *Profile* stranici kako bi se, pri mijenjanju stilova piva koja se korisniku sviđaju, mogli prikazati svi stilovi piva koje korisnik može označiti da mu se sviđaju. Programski kod za ovaj upit se nalazi u prilogu.

```
async getFilteredBeers(_, {beerInput: { beerClass, drinkability, alcoholFlavour, hopAroma, bitterness, carbonation, fruitiness, ABV, cocoaAroma}}){
    try{
        const beersInClass = await Beer.find({beerClass});
        const filteredBeers = await Beer.find({ drinkability, alcoholFlavour, hopAroma, bitterness, carbonation, fruitiness, ABV, cocoaAroma})
        return new Set([...beersInClass, ...filteredBeers]);
    } catch(err){
        throw new Error(err);
    }
},
```

Slika 4.21. *Kod za poziv getFilteredBeers*

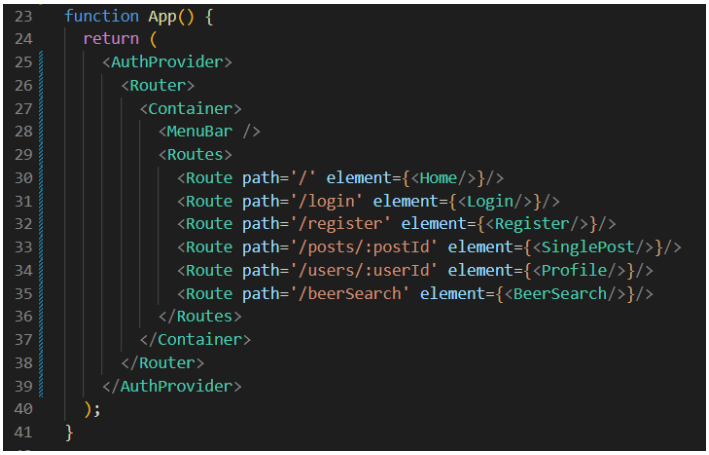
Slika 4.21. prikazuje kod za dohvaćanje piva sa zadanim karakteristikama. Kao argumenti predaju se *beerClass* i atributi koji predstavljaju karakteristike stilova piva. Prvo se pronalaze svi stilovi piva koji imaju odgovarajući *beerClass*, a zatim se dohvaćaju i stilovi piva čije karakteristike odgovaraju onim koja su predana kao argument. Povratna vrijednost poziva su svi pronađeni stilovi piva, koji se predaju kao *Set* kako bi se izbjeglo ponavljanje više istih stilova piva u listi.

Ovaj poziv se koristi na stranici *BeerSearch* kako bi korisnik ponašao stilove piva koji odgovaraju zadanom opisu. Više o ovoj stranici piše u nastavku ovog poglavlja.

4.4. Komponente i stranice

4.4.1. *App.js* datoteka

App.js datoteka je glavna datoteka koja se pokreće pristupanjem web aplikaciji i unutar koje se pozivaju pojedine stranice kojima korisnik može pristupiti.



Slika 4.22. *Funkcija App unutar App.js i sve njene komponente*

Prema slici 4.22. može se primijetiti kako su svi elementi unutar *AuthProvider* komponente kako bi se na svakoj stranici moglo provjeriti jeli korisnik prijavljen i koji su podaci prijavljenog korisnika. Detaljnije o *AuthProvider* komponenti piše u poglavlju *4.3.3. Autentifikacija korisnika*. Također se može primijetiti kako je većina elemenata unutar *Router* elementa te sadrži komponente naziva *Route* unutar *Routes* komponente. Prema slici 4.56. prikazano je kako se radi o komponentama koje pripadaju *react-router-dom* biblioteci koja omogućuje mijenjanje samo dijela stranice tako što će komponenta *MenuBar* uvijek biti na vrhu stranice, a ovisno o odabiru stranice bit će prikazana samo jedna komponenta od *Route* komponenti.

Contrainer komponenta unutar koje su komponente koje daju izgled web aplikaciji je prema slici 4.56. uvedena iz *semantic-ui-react* biblioteke te se brine da sadržaj stranice ne prelazi maksimalnu širinu [15].

4.4.2. *MenuBar* komponenta

MenuBar komponenta je dinamička komponenta koja korisniku omogućuje odabir stranice kojoj želi pristupiti i ovisno o odabranoj stranici *MenuBar* komponenta podcrta tekst gumba stranice koja je korisniku trenutno prikazana. Također ovisno o tome jeli korisnik prijavljen ili ne bit će mu ponuđeno da se odjavi ako je prijavljen ili da se prijavi i registrira ako nije prijavljen.

```
function MenuBar() {
  const { user, logout } = useContext(AuthContext);

  const pathname = window.location.pathname;
  const path = pathname === '/' ? 'home' : pathname.substring(1);
  const [activeItem, setActiveItem] = useState(path);

  const handleItemClick = (e, { name }) => setActiveItem(name);
```

Slika 4.23. *Postavljanje početnih vrijednosti varijabli komponente MenuBar*

Slika 4.23. prikazuje postavljanje vrijednosti varijable *user* pomoću koje se određuje jeli korisnik prijavljen. Zatim se, uz pomoć adrese stranice, određuje koja je stranica prikazana korisniku kako bi se mogao označiti odgovarajući gumb na izborniku.

```
<Menu pointing secondary size="massive" color="teal">
  <Menu.Menu position='left'>
  <Menu.Item
      name='home'
      active={activeItem === 'home'}
      onClick={handleItemClick}
      as={Link}
      to="/"
    />
    <Menu.Item
      name={user.username}
      active={activeItem === user.username}
      onClick={handleItemClick}
      as={Link}
      to={`/users/${user.id}`}
    />
    <Menu.Item
      name='beerSearch'
      active={activeItem === 'beerSearch'}
      onClick={handleItemClick}
      as={Link}
      to="/beerSearch"
    />
  </Menu.Menu>


  <Menu.Menu position='right'>
    <Menu.Item
      name='logout'
      onClick={logout}
    />
  </Menu.Menu>
</Menu>
```

Slika 4.24. *Sadržaj komponente MenuBar ako je korisnik prijavljen*

Prema slici 4.24. u slučaju kada je korisnik prijavljen bit će mu prikazan izbornik na kojem se na lijevom kraju nalazi gumb koji ga vodi na *Home* stranicu, desno od tog gumba pisat će korisnikov *username* čijim klikom će biti odveden na svoju *Profile* stranicu, a desno od tog gumba je gumb koji korisnika vodi na *BeerSearch* stranicu. Na desnom kraju je prikazan gumb preko kojeg je korisniku omogućeno odjaviti se.

Prema slici 4.25. u slučaju kada korisnik nije prijavljen bit će mu prikazan izbornik na kojem je na lijevom kraju prikazan gumb koji ga vodi na *Home* stranicu i gumb koji ga vodi na *BeerSearch* stranicu, a na desnom gumbovi koji ga vode na *Login* stranicu za prijavljivanje ili *Register* stranicu za registriranje novog korisnika društvene mreže.

```
51      <Menu pointing secondary size="massive" color="teal">
52        <Menu.Menu position='left'>
53          <Menu.Item
54            name='home'
55            active={activeItem === 'home'}
56            onClick={handleItemClick}
57            as={Link}
58            to="/"
59          />
60          <Menu.Item
61            name='beerSearch'
62            active={activeItem === 'beerSearch'}
63            as={Link}
64            onClick={handleItemClick}
65            to="/beerSearch"
66          />
67        </Menu.Menu>
68
69
70        <Menu.Menu position='right'>
71          <Menu.Item
72            name='login'
73            active={activeItem === 'login'}
74            onClick={handleItemClick}
75            as={Link}
76            to="/login"
77          />
78          <Menu.Item
79            name='register'
80            active={activeItem === 'register'}
81            onClick={handleItemClick}
82            as={Link}
83            to="/register"
84          />
85        </Menu.Menu>
86      </Menu>
```

Slika 4.25. *Sadržaj komponente MenuBar ako korisnik nije prijavljen*

4.4.3. Home komponenta

Pomoću upita getPosts dohvaćaju se podaci o objavama.

```
return (
  <Grid columns={3} divided>
    <Grid.Row className='page-title'>
      <h1>Recent data</h1>
    </Grid.Row>
    <Grid.Row>
      { user && (
        <Grid.Column>
          <PostForm/>
        </Grid.Column>
      )}
      {loading ? (
        <h1>Loading posts..</h1>
      ) : (
        <Transition.Group>
          {data &&
            data.getPosts.map((post) => (
              <Grid.Column key={post.id} style={{ marginBottom: 20}}>
                <PostCard post={post} />
              </Grid.Column>
            ))}
        </Transition.Group>
      )}
    </Grid.Row>
  </Grid>
)
```

Slika 4.26. *Sadržaj komponente Home*

Prema slici 4.26. *Home* komponenta, uz pomoć *Grid* komponente uvedene iz *semantic-ui-react* biblioteke, prikazuje mogućnost objavljivanja vlastite objave i do sad objavljene objave svih korisnika. Ovisno o tome jeli korisnik prijavljen bit će prikazana *PostForm* komponenta o kojoj više piše kasnije. Dok se objave učitavaju bit će ispisan samo tekst „Loading posts..“, a nakon što se objave dohvate pomoću funkcije *map* se prikazuju *PostCard* komponente, od kojih svaka prikazuje podatke o jednoj objavi. Više o komponenti *PostCard* piše u nastavku.

4.4.4. *PostForm* komponenta

Postavlja se početna vrijednost varijable *body* kao prazan *string* koji će se kasnije mijenjati unošenjem teksta i poslati na bazu podataka stvaranjem objave pomoću *createPost* mutacije. Za postavljanje početnih vrijednosti koristi se funkcija *useForm* koja pomaže u postavljanju početnih vrijednosti i određivanju ponašanja funkcija koje se koriste u datoteci *PostForm.js*. Nakon

stvaranja nove objave ponovno se poziva *getPosts* upit kako bi se prikazala i nova objava s ostalim objavama.

```
import { useState } from "react";

export const useForm = (callback, initialState = {}) => {
    const [values, setValues] = useState(initialState);

    const onChange = (event) => {
        setValues({ ...values, [event.target.name]: event.target.value });
    };

    const onSubmit = event => {
        event.preventDefault();
        callback();
    }

    return {
        onChange,
        onSubmit,
        values
    }
}
```

Slika 4.8. *Prikaz pomoćne funkcije UseForm*

Prema slici 4.27. useForm funkcija prima povratni poziv i početno stanje te određuje ponašanje funkcija *onChange*, *onSubmit* i vrijednost varijable *values*.

```
    return (
        <>
        <Form onSubmit={onSubmit}>
            <h2>Create a post:</h2>
            <Form.Field>
                <Form.Input
                    placeholder="Hi World!"
                    name="body"
                    type='text'
                    value={values.body}
                    onChange={onChange}
                    error={error ? true : false}
                />
                <Button type='submit' color='teal'>
                    Submit
                </Button>
            </Form.Field>
        </Form>
        { error && (
            <div className='ui error message' style={{marginBottom: 20}}>
                <ul className='list'>
                    <li>{error.graphQLErrors[0].message}</li>
                </ul>
            </div>
        )}
        </>
    )
```

Slika 4.9. *Sadržaj komponente PostForm*

Kao što je prikazano na slici 4.28. *PostForm* komponenta prikazuje *Form* komponentu uvedenu iz biblioteke *semantic-ui-react* koja prikazuje polje za unos teksta i gumb čijim pritiskom se

pokreće funkcija za spremanje objave s unesenim tekstom. U slučaju da se pokuša objaviti objava bez unesenog teksta prikazat će se element koji sadrži tekst upozorenja da objava mora imati sadržaj te objava neće biti objavljena.

4.4.5. *PostCard* komponenta

Komponenti *PostCard* se kao argument predaje *post* objekt iz kojeg se uzimaju sve vrijednosti varijabli potrebne za ispravno prikazivanje objave. Kao i u ostalim komponentama u kojima je potrebno provjeriti jeli korisnik prijavljen, ponovno se koristi *useContext* funkcija.

```
    return (
    <Card fluid>
      <Card.Content>
        <Image
          floated='right'
          size='mini'
          src='https://react.semantic-ui.com/images/avatar/large/molly.png'
        />
        <Card.Header as={Link} to={`/users/${userId}`}>{username}</Card.Header>
        <Card.Meta as={Link} to={`/posts/${id}`}>{moment(createdAt).fromNow(true)}</Card.Meta>
        <Card.Description>
            {body}
        </Card.Description>
      </Card.Content>
      <Card.Content extra>
        <LikeButton user={user} post={{ id, likes, likeCount }} />
        <Button labelPosition='right' as={Link} to={`/posts/${id}`}>
            <Button color='blue' basic>
                <Icon name='comments' />
            </Button>
            <Label basic color='blue' pointing='left'>
                {commentCount}
            </Label>
        </Button>
        { user && user.username === username && <DeleteButton postId={id} />}
      </Card.Content>
    </Card>
    )
}
```

Slika 4.29. *Sadržaj komponente PostCard*

Prema slici 4.29. komponenta *PostCard*, uz pomoć komponente *Card* uvedene iz *semantic-ui-react* biblioteke, stvori se objava koja prikazuje sliku profila korisnika koji je napravio objavu, ime korisnika, vrijeme koje je prošlo od nastanka objave, sadržaj objave i gumbove za pozitivno ocjenjivanje objave, prikazivanje objave preko cijele stranice i brisanje objave. Komponenta *LikeButton* za pozitivno ocjenjivanje objave je detaljnije objašnjena u nastavku ovog poglavlja, gumb za prikazivanje objave preko cijele stranice je prikazan kao gumb za komentiranje objave te ispod njega je prikazan broj do sada napisanih komentara na objavi. Gumb za brisanje objave prikazan je samo ako je korisnik prijavljen i ako je korisničko ime (engl. *username*) korisnika

jednako korisničkom imenu korisnika koji je napravio objavu, a više o gumbu za brisanje, to jest *DeleteButton* komponenti, piše kasnije.

4.4.6. *LikeButton* komponenta

```
function LikeButton({ user, post: { id, likeCount, likes }}){
    const [liked, setLiked] = useState(false);
    useEffect(() => {
        if(user && likes.find(like => like.username === user.username)){
            setLiked(true)
        } else setLiked(false)
    }, [user, likes]);

    const [likePost] = useMutation(LIKE_POST_MUTATION, {
        variables: { postId: id}
    });

    const likeButton = user ? (
        liked ? (
            <Button color='teal'>
                <Icon name='heart' />
            </Button>
        ) : (
            <Button color='teal' basic>
                <Icon name='heart' />
            </Button>
        )
    ) : (
            <Button as={Link} to="/login" color='teal' basic>
                <Icon name='heart' />
            </Button>
    )
```

Slika 4.30. *Postavljanje početnih vrijednosti varijabli komponente LikeButton i postavljanje sadržaja koji će biti prikazan*

Slika 4.30. prikazuje kako je postavljena početna vrijednost varijable *liked* kao *false* što predstavlja da objava nije pozitivno ocijenjena. Zatim se provjerava u bazi podataka podudara li se korisničko ime prijavljenog korisnika s jednim od korisničkih imena korisnika koji su pozitivno ocijenili objavu. U slučaju da se potvrdi da je prijavljeni korisnik ipak pozitivno ocijenio bio objavu vrijednost varijable *liked* se mijenja u *true* što utječe na izgled gumba za pozitivno ocjenjivanje objave. Kao što je prikazano dizajn gumba se mijenja ovisno o vrijednosti varijable *liked*, a u slučaju kada korisnik nije prijavljen pritiskom na gumb korisnik će otvoriti *Login* stranicu na kojoj se može prijaviti.

```
    return(
        <Button as='div' labelPosition='right' onClick={likePost}>
            {likeButton}
            <Label basic color='teal' pointing='left'>
                {likeCount}
            </Label>
        </Button>
    )
}
```

Slika 4.31. *Sadržaj komponente LikeButton*

Prema slici 4.31., uz sadržaj prikazan kodom na slici 4.30., prikazuje se broj pozitivnih ocjena i klikom na gumb se pokreće funkcija koja pokreće *likePost* mutaciju koja objavu označava kao pozitivno ocijenjenu ili joj oduzima pozitivnu ocjenu ako ju je korisnik još prije pozitivno ocijenio bio.

4.4.7. *SinglePost* komponenta

Pomoću funkcije *useParams* uvedene iz biblioteke *react-router-dom* iščitava se URL u kojem se nalazi ID objave koju treba prikazati korisniku. Funkcijom *useContext* dobivaju se podaci o prijavljenom korisniku. Upitom *getPost* dohvaćaju se podaci o prikazanoj objavi, a mutacijom *createComment* omogućuje se dodavanje novog komentara na objavu. Korištenje navedenih funkcija prikazano je u programskom kodu koji se nalazi u prilogu.

```
    let postMarkup;
    if(!data){
        postMarkup = <p>Loading post..</p>
    } else{
        const { id, body, createdAt, username, comments, likes, likeCount, commentCount} = data.getPost;

        postMarkup = (
            <Grid>
                <Grid.Row>
                    <Grid.Column width={2}>
                        <Image
                        src='https://react.semantic-ui.com/images/avatar/large/molly.png'
                        size='small'
                        float='right'/>
                    </Grid.Column>
                    <Grid.Column width={10}>
                        <Card fluid>
                            <Card.Content>
                                <Card.Header>{username}</Card.Header>
                                <Card.Meta>{moment(createdAt).fromNow()}</Card.Meta>
                                <Card.Description>{body}</Card.Description>
                            </Card.Content>
                            <hr/>
```

Slika 4.32. *Prvi dio sadržaja komponente SinglePost*

Prema slici 4.32. provjerava se jesu li podaci o objavi dohvaćeni i u slučaju da nisu stranica prikazuje samo tekst „Loading Post..“. U slučaju da su podaci dohvaćeni oni se spremaju u

pojedine varijable koje se koriste za prikazati objavu. Sadržaj stranice prikazuje sliku korisnika koji je objavio objavu te pored slike podatke vezane za autora objave, detalje objave i sadržaj objave.

```
66                  <Card.Content extra>
67                      <LikeButton user={username} post={{ id, likeCount, likes }}/>
68                      <Button
69                          as='div'
70                          labelPosition='right'
71                          onClick={() => console.log('Comment on post')}
72                          >
73                              <Button basic color='blue'>
74                                  <Icon name='comments' />
75                              </Button>
76                              <Label basic color='blue' pointing='left'>
77                                  {commentCount}
78                              </Label>
79                          </Button>
80                          { user && user.username === username && <DeleteButton postId={id} callback={deletePostCallback}/>}
81                  </Card.Content>
82              </Card>
```

Slika 4.33. *Drugi dio sadržaja komponente SinglePost*

Slika 4.33. prikazuje da se ispod sadržaja opisanog kodom u slici 4.32. nalaze komponenta *LikeButton*, koja je objašnjena ranije u poglavlju, komponenta koja prikazuje broj komentara napisanih na objavi i komponenta *DeleteButton* koja je prikazana samo autoru objave i omogućuje brisanje objave i vraćanje na *Home* stranicu uz pomoć *useNavigate* funkcije.

```
83                  {user && (
84                      <Card fluid>
85                          <Card.Content>
86                              <p>Post a comment</p>
87                              <Form>
88                                  <div className="ui action input fluid">
89                                      <input
90                                          type="text"
91                                          placeholder="Comment.."
92                                          name="comment"
93                                          value={comment}
94                                          onChange={event => setComment(event.target.value)}
95                                          ref={commentInputRef}
96                                      />
97                                      <button
98                                          type="submit"
99                                          className="ui button teal"
100                                         disabled={comment.trim() === ''}
101                                         onClick={submitComment}
102                                     >
103                                         Submit
104                                     </button>
105                                 </div>
106                             </Form>
107                         </Card.Content>
108                     </Card>
109                 )}
```

Slika 4.34. *Treći dio sadržaja komponente SinglePost*

Prema slici 4.34. ispod detalja o objavi i gumbova vezanih za interakciju s objavom prikazan je formular za unos teksta komentara i gumb za objavljivanje komentara. Komponente opisane kodom u slici 4.34. su prikazane samo ako je potvrđeno da je korisnik prijavljen.

```
                    {comments.map(comment  => (
                        <Card fluid key={comment.id}>
                            <Card.Content>
                                {user && user.username === comment.username &&(
                                    <DeleteButton postId={id} commentId={comment.id}/>
                                )}
                                <Card.Header>{comment.username}</Card.Header>
                                <Card.Meta>{moment(comment.createdAt).fromNow()}</Card.Meta>
                                <Card.Description>{comment.body}</Card.Description>
                            </Card.Content>
                        </Card>
                    ))}
                </Grid.Column>
            </Grid.Row>
        </Grid>
    )
  }
  return postMarkup;
}
```

Slika 4.35. *Četvrti dio sadržaja komponente SinglePost*

Slika 4.35. prikazuje da su ispod do sada navedenog sadržaja komponente *LikeButton* prikazani sadržaji i podaci komentara objavljenih na objavi te je autorima komentara prikazana komponenta *DeleteButton* čijim pritiskom se briše komentar.

4.4.8. *DeleteButton* komponenta

DeleteButton komponenta se može koristiti ili za brisanje objave ili za brisanje komentara. Ovisno jeli komponenti kao argument predan samo *postId* koji predstavlja ID objave ili je predan i *commentId* koji predstavlja ID komentara. Ovisno o predanim argumentima pozvat će se mutacija *deletePost* ili *deleteComment.* Pri pozivu funkcije koja pokreće navedene mutacije, ako je pozvana mutacija za brisanje objave onda će se pozvati upit *getPosts* kako bi se prikazale samo objave koje su preostale.

```
    return(
        <>
        <Button
            as="div"
            color="red"
            floated="right"
            onClick={deletePostOrComment}
            >
                <Icon name="trash" style={{ margin: 0}} />
            </Button>
        </>
    )
}
```

Slika 4.36. *Sadržaj komponente DeleteButton*

Slika 4.36. prikazuje kako se *DeleteButton* sastoji od *Button* komponente čijim pritiskom se pokreće jedna od navedenih mutacija za brisanje objave ili komentara i *Icon* komponente koja na gumbu prikazuje ikonicu kante za smeće.

4.4.9. *Login* komponenta

Postavljaju se početne vrijednosti za *errors* varijablu koja će sadržavati sve greške koje program uhvati i *values* varijablu koja će čuvati podatke koje korisnik koristi kako bi se prijavio.

Funkcija *loginUser* pokreće *login* mutaciju koja će prijaviti korisnika ako je dao ispravne podatke za prijavu. Nakon uspješne prijave korisnika se uz pomoć *useNavigate* funkcije usmjerava na *Home* stranicu.

```
46    return (
47      <div className='form-container'>
48        <Form onSubmit={onSubmit} noValidate className={loading ? "loading" : ""}>
49          <h1>Login</h1>
50          <Form.Input
51            label="Username"
52            placeholder="Username.."
53            name="username"
54            type='text'
55            value={values.username}
56            error={errors.username ? true : false}
57            onChange={onChange}
58            className="form-input"
59          />
60          <Form.Input
61            label="Password"
62            placeholder="Password.."
63            name="password"
64            type='password'
65            value={values.password}
66            error={errors.password ? true : false}
67            onChange={onChange}
68            className="form-input"
69          />
70          <Button type='submit' primary>
71            Login
72          </Button>
73        </Form>
```

Slika 4.10. *Prvi dio sadržaja komponente Login*

Prema slici 4.37. *Login* komponenta se sastoji od formulara koji sadrži dvije *Form.Input* komponente za unos teksta u koje se unosi korisničko ime i zaporka. Ispod polja za unos teksta prikazana je *Button* komponenta čijim pritiskom se pokreće pokušaj prijave s unesenim podacima.

Slika 4.38. prikazuje da će, ako se pri unosu korisničkog imena ili zaporke pojavi jedna ili više greški, sve greške biti ispisane ispod formulara.

```
74          {Object.keys(errors).length > 0 && (
75            <div className='ui error message'>
76              <ul className='list'>
77                {Object.values(errors).map(value => (
78                  <li key={value}>{value}</li>
79                ))}
80              </ul>
81            </div>
82          )}
83        </div>
84      )
85    }
```

Slika 4.11. *Drugi dio sadržaja komponente Login*

4.4.10. *Register* komponenta

Prvo se postavljaju početne vrijednosti varijabli *errors* koja sadržava greške koje će program uhvatiti i *values* koja sadrži podatke koje korisnik unosi i s kojima će se registrirati i stvoriti račun. Uz pomoć upita *getAllBeers* dohvaćaju se podaci svih stilova piva i vrijednosti njihovih atributa, te se spremaju u posebnu listu objekata nazvanu *BeerOptions*.

Adduser funkcija pokreće *register* mutaciju i predaje joj kao argumente podatke za registraciju koje korisnik sam unosi. Nakon uspješne registracije u bazu podataka se dodaje novi korisnik te se korisnika uz pomoć *useNavigate* funkcije usmjerava na *Home* stranicu.

```
50    return (
51      <div className='form-container'>
52        <Form onSubmit={onSubmit} noValidate className={loading ? "loading" : ""}>
53          <h1>Register</h1>
54          <Form.Input
55            label="Username"
56            placeholder="Username.."
57            name="username"
58            type='text'
59            value={values.username}
60            error={errors.username ? true : false}
61            onChange={onChange}
62            className="form-input"
63          />
64          <Form.Input
65            label="Email"
66            placeholder="Email.."
67            name="email"
68            type='email'
69            value={values.email}
70            error={errors.email ? true : false}
71            onChange={onChange}
72            className="form-input"
73          />
```

Slika 4.39. *Prvi dio sadržaja komponente Register*

Kao što je prikazano na slici 4.39. *Register* komponenta sadrži *Form* komponentu uvedenu iz biblioteke *semantic-ui-react* koja sadrži niz polja za unos svih podataka potrebnih za stvaranje novog korisničkog računa.

```
120             <Form.Dropdown
121               control={Select}
122               options={
123                 data ? (
124                   beerOptions
125                 ) : ([])
126               }
127               label="Liked Beer Style"
128               placeholder='Beer Style Name..'
129               name="likedBeersId"
130               search
131               multiple
132               selection
133               value={values.likedBeersId}
134               onChange={onChange}
135             />
```

Slika 4.40. *Drugi dio sadržaja komponente Register*

Slika 4.40. prikazuje komponentu *Dropdown* koja je izbornik koji, ako su podaci o stilovima piva dohvaćeni, prikazuje sve objekte u listi *beerOptions*. *Dropdown* zahtjeva da objekti liste imaju samo atribute *key*, *value* i *text* zbog čega je izrađena lista *beerOptions*. Također omogućuje pretraživanje izbornika unosom teksta imena stila piva i mogućnost označivanja više različitih stilova piva koji se sviđaju korisniku.

```
136             <Button type='submit' primary>
137               Register
138             </Button>
139           </Form>
140           {Object.keys(errors).length > 0 && (
141             <div className='ui error message'>
142               <ul className='list'>
143                 {Object.values(errors).map(value => (
144                   <li key={value}>{value}</li>
145                 ))}
146               </ul>
147             </div>
148           )}
149         </div>
150       )
151     }
```

Slika 4.41. *Treći dio sadržaja komponente Register*

Prema slici 4.41. nakon niza polja za unos podataka prikazana je *Button* komponenta čijim pritiskom se pokreće pokušaj registracije korisnika. U slučaju pojavljivanja grešaka pri unosu podataka one će biti ispisane ispod formulara i neće se stvoriti novi korisnički račun.

4.4.11. *Profile* komponenta

Iz adrese stranice se uzima *id* korisnika vlasnika profilne stranice na kojoj se korisnik nalazi i podaci o prijavljenom korisniku. Zatim se stvaraju pomoćne varijable o kojima više piše kasnije.

Funkcija *submitNewBody* pomoću poziva mutacije *editBody* mijenja vrijednost atributa *body*, a funkcija *submitNewLikedBeers* pomoću poziva mutacije *editLikedBeers* mijenja vrijednost atributa *likedBeersId*. Obje funkcije nakon svojeg poziva ponovno dohvaćaju podatke o korisniku kako bi se oni prikazali bez osvježavanja stranice.

Ako se još nisu dohvatili podaci o korisniku stranica prikazuje samo tekst „*Loading profile.. “*.

```
profileMarkup = (
    <Grid>
        <Grid.Row>
            <Grid.Column width={2}>
                <Image
                src='https://react.semantic-ui.com/images/avatar/large/molly.png'
                size='small'
                float='right'/>
            </Grid.Column>
            <Grid.Column width={10}>
                <Card fluid>
                    <Card.Content>
                        {user && user.id === id &&(
                            <Button
                            as="div"
                            color="blue"
                            floated="right"
                            onClick={() => {
                                editToggle ? setEditToggle(false) : setEditToggle(true);
                            }}
                            >
                                <Icon name="edit" style={{ margin: 0}} />
                            </Button>
                        )}
                        <Card.Header>{username}</Card.Header>
                        <Card.Meta>{moment(createdAt).fromNow()}</Card.Meta>
                        <Card.Description>{body}</Card.Description>
                        <Card.Description>
                            {likedBeers[0] ? (<div>Beers I like are:</div>) : (<div>I am not sure what I like yet...</div>)}
                            {likedBeers.map((beer) => (
                                <div key={beer.id}><b>{beer.styleName}</b> which is one of the <b>{beer.beerClass}s</b></div>
                            ))}
                        </Card.Description>
                    </Card.Content>
                </Card>
```

Slika 4.42. *Prvi dio sadržaja komponente Profile*

Prema slici 4.42., ako su podaci o korisniku dohvaćeni, bit će prikazani njegovo korisničko ime, vrijeme nastanka računa, opis i piva koja je korisnik označio da mu se sviđaju. Također, ako je prijavljeni korisnik isti kao i korisnik koji je vlasnik stranice profila, bit će prikazan i gumb koji mijenja vrijednost varijable *editToggle* koja može biti *true* ili *false* i koja odlučuje hoće li biti

prikazane komponente za uređivanje korisnikovih atributa. Za prikaz stilova piva koja se sviđaju korisniku koristi se *map* funkcija koja će za svaki element u listi *likedBeers* prikazati *div* element u kojem piše *stylename* i *beerClass* svakog pojedinog stila piva.

Ako je prijavljeni korisnik isti kao i korisnik koji je vlasnik profilne stranice na kojoj se nalazi i vrijednost varijable *editToggle* je *true*, onda će korisniku biti omogućen unos novog teksta za atribut *body* i pritiskom na gumb taj atribut će se promijeniti pozivnom funkcije *submitNewBody*.

```
{user && user.id === id && editToggle && (
    <Card fluid>
        <Card.Content>
            <p>Change your liked beers</p>
            <Form>
                <Form.Dropdown
                    control={Select}
                    options={
                        beers.data ? (
                        beerOptions
                        ) : ([])
                    }
                    label="Liked Beer Style"
                    placeholder='Beer Style Name..'
                    name="likedBeersId"
                    search
                    multiple
                    selection
                    defaultValue={users.data.getUser.likedBeersId}
                    onChange={onChange}
                />
                <button
                        type="submit"
                        className="ui button teal"
                        onClick={submitNewLikedBeers}
                    >
                        Submit
                </button>
            </Form>
        </Card.Content>
    </Card>
```

Slika 4.43. *Drugi dio sadržaja komponente Profile*

Također, prema slici 4.43., ako je prijavljeni korisnik isti kao i korisnik koji je vlasnik profilne stranice na kojoj se nalazi i vrijednost varijable *editToggle* je *true*, onda je prikazana *Dropdown* komponenta na kojoj se mogu označiti piva koja se sviđaju korisniku i gumb koji mijenja vrijednost atributa *likedBeersId* na vrijednosti koje su odabrane *Dropdown* komponentom. Kako korisnik ne bi morao iznova odabirati sve stilove piva koje mu se sviđaju, unaprijed su označeni stilovi koji su do sada bili označeni u *likedBeersId* atributu.

4.4.12. *BeerSearch* komponenta

Prvo se postavljaju početne vrijednosti objekta *values* koji sadrži varijable koje će biti argumenti za upit *getFilteredBeers* koji dohvaća stilove piva sa zadanim vrijednostima atributa.

```
44      return (
45          <div className='form-container'>
46              <Form onSubmit={onSubmit} noValidate>
47                  <h1>Search Beer</h1>
48                  <Form.Dropdown
49                      control={Select}
50                      options={[
51                          {key: "any", text: "I Do Not Prefer Any Styles", value: "" },
52                          {key: "belgian-style", text: "Belgian-Styles", value: "Belgian-Style" },
53                          {key: "bock", text: "Bock", value: "Bocks" },
54                          {key: "brown ale", text: "Brown Ales", value: "Brown Ale" },
55                          {key: "dark lager", text: "Dark Lagers", value: "Dark Lager" },
56                          {key: "india pale ale", text: "India Pale Ales", value: "India Pale Ale" },
57                          {key: "pale ale", text: "Pale Ales", value: "Pale Ale" },
58                          {key: "lager", text: "Lagers", value: "Lager" },
59                          {key: "stout and porter", text: "Stout and Porters", value: "Stout and Porter" },
60                          {key: "wheat beer", text: "Wheat Beers", value: "Wheat Beer" }
61                      ]}
62                      label="What Styles Do You Like?"
63                      placeholder='I Do Not Prefer Any Styles'
64                      name="filteredBeerClass"
65                      selection
66                      value={values.filteredBeerClass}
67                      onChange={onChange}
68                  />
```

Slika 4.44. *Prvi dio sadržaja komponente BeerSearch*

Kao što je prikazano na slici 4.44. stranica se sastoji od niza *Dropdown* komponenti koje su izbornici koji nude odabir jedne vrijednosti svake pojedine karakteristike stila piva. Te vrijednosti se spremaju u atribute objekta *values* i koriste se kao argumenti za pretraživanje stilova piva. Svaki *Dropdown* također nudi mogućnost odabira vrijednosti praznog teksta čijim odabirom se pri pretraživanju stilova piva neće uzimati u obzir vrijednost tog argumenta.

```
188                 <h2>Beers You May Like Are:</h2>
189                 {data ? (
190                     data.getFilteredBeers[0] ? (
191                         data.getFilteredBeers.map((beer) => (
192                             <p key={beer.styleName}><b>{beer.styleName}</b> which is one of the <b>{beer.beerClass}s</b></p>
193                         )
194                     )) : (
195                         <p>Well... I'm Sorry But I Don't Know A Beer That Fits That Description</p>
196                     )
197                     ) : (
198                         <div>Loading Beers...</div>
199                     )}
200             </Form>
201         </div>
202     )
203 }
```

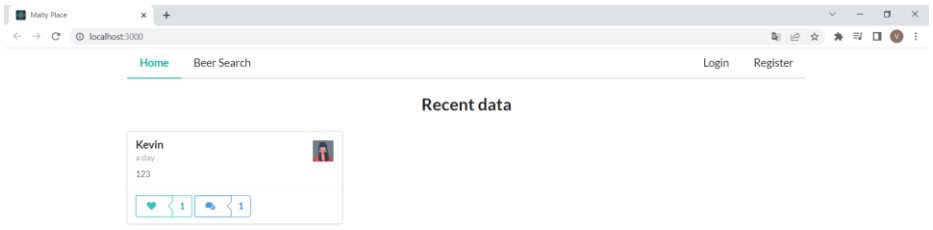
Slika 4.45. *Drugi dio sadržaja komponente BeerSearch*

Na slici 4.45. prikazan je dio koda koji, ako su dohvaćeni stilovi piva, prikazuje stilove piva koji odgovaraju zadanom opisu, to jest vrijednosti njihovih *stylename* i *beerClass* atributa uz pomoć funkcije *map*. U slučaju da podaci o stilovima piva još nisu dohvaćeni ili ne postoji ni jedan stil piva koji odgovara opisu prikaže se odgovarajući tekst.

Komentirano [VH27]: Ostavit jednu od ovih

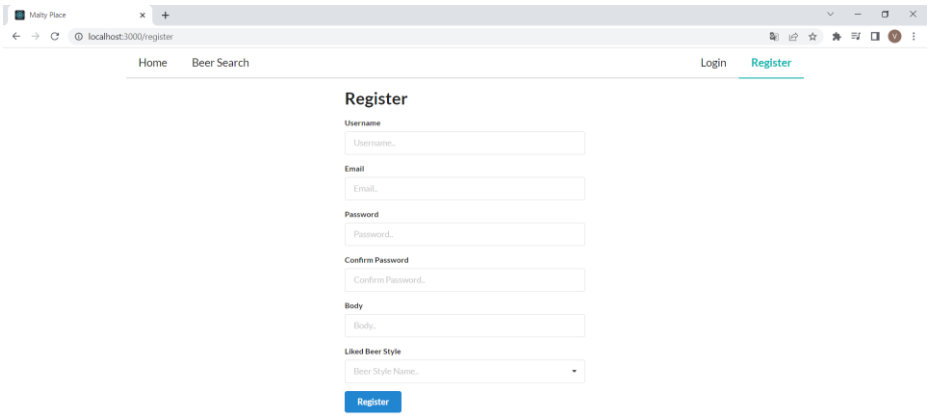
5. RAD S APLIKACIJOM

Nakon pokretanja web aplikacije korisniku je prikazan izbornik na kojem može birati stranicu i *Home* stranica.



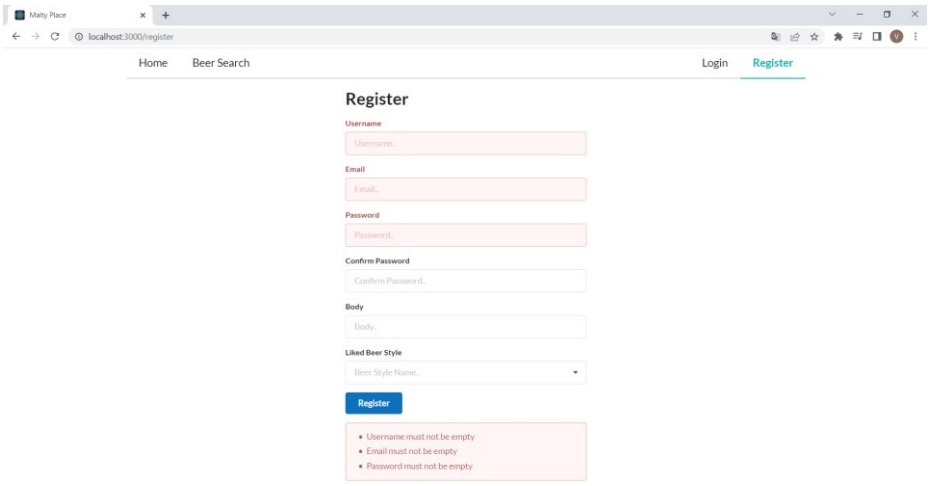
Slika 5.1. *Izgled Home stranice dok korisnik nije prijavljen*

Slika 5.1. prikazuje da se korisniku koji nije prijavljen na *Home* stranici prikazuju sve dohvaćene objave koje se nalaze u bazi podataka. Na izborniku, koji je uvijek prikazan korisniku, nalaze se gumbovi koji korisnika vode na stranice *BeerSearch, Login* ili *Register* na kojima može pretraživati stilove piva te prijaviti se ili registrirati na web aplikaciju. Korisniku su na objavama prikazani podaci o objavi, korisničko ime autora objave i gumbovi za pozitivno označiti objavu ili komentirati objavu. Pritiskom na gumb za pozitivno označiti objavu aplikacija će neprijavljenog korisnika odvesti na *Login* stranicu, a pritiskom na gumb za komentiranje aplikacija vodi korisnika na stranicu objave o kojoj više piše kasnije u ovom poglavlju. Pritiskom na korisničko ime autora objave, aplikacija vodi korisnika na *Profile* stranicu korisnika autora objave, kojoj također više piše u nastavku.

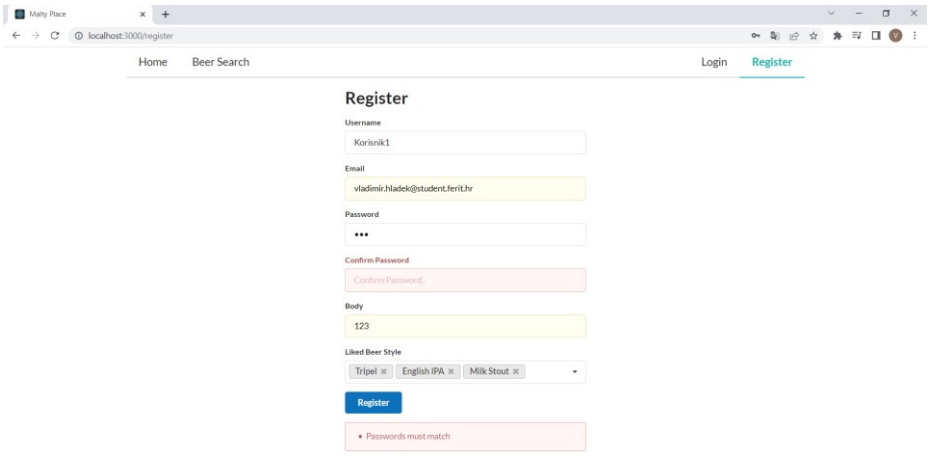


Slika 5.2. *Izgled Register stranice*

Prema slici 5.2. na *Register* stranici korisniku je prikazan niz polja za ispunjavanje tekstom u koje treba unijeti tražene podatke za stvaranje novog korisničkog računa i izbornik koji prikazuje stilove piva koje korisnik može označiti da mu se sviđaju. Izbornik se može pretraživati unosom teksta i može se odabrati više različitih stilova piva.



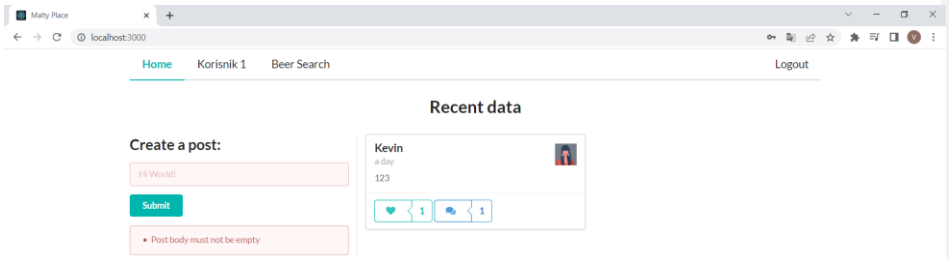
Slika 5.3. *Prvi izgled Register stranice u slučaju nastanka greške*



Slika 5.4. *Drugi izgled Register stranice u slučaju nastanka greške*

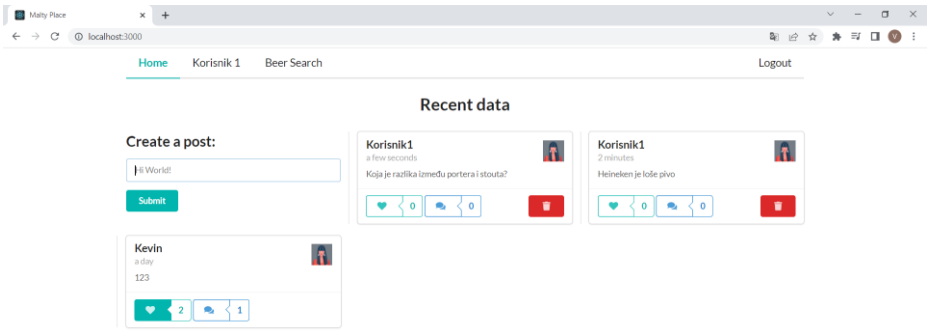
Prema slikama 5.3. i 5.4., u slučaju da nisu sva obvezna polja ispunjena ili unesena zaporka nije potvrđena ponovnim unošenjem u *confirmPassword* polju, prikazuju se odgovarajuće poruke upozorenja.

Komentirano [KN28]: Zaporka - promijeniti u cijelom tekstu



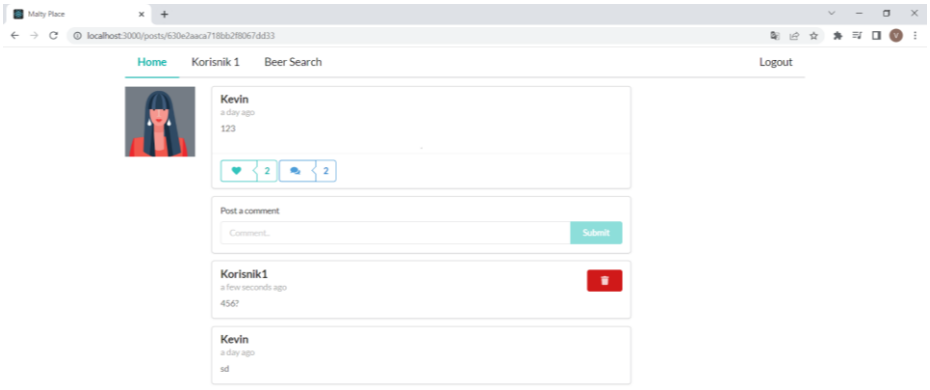
Slika 5.5. *Prvi izgled Home stranice nakon prijavljivanja*

Slika 5.5. prikazuje da nakon uspješnog prijavljivanja na izborniku više nisu prikazani gumbovi za prijavljivanje i registraciju, ali prikazan je gumb *Logout* čijim pritiskom će se korisnik odjaviti s prijavljenog računa. Korisnik sada može pozitivno označiti objavu, a klikom na gumb promijenit će se izgled gumba. Također je korisniku ponuđena opcija objavljivanja vlastite objave, ali pritiskom na gumb *Submit* bez dodavanja teksta objave bit će prikazana poruka greške.



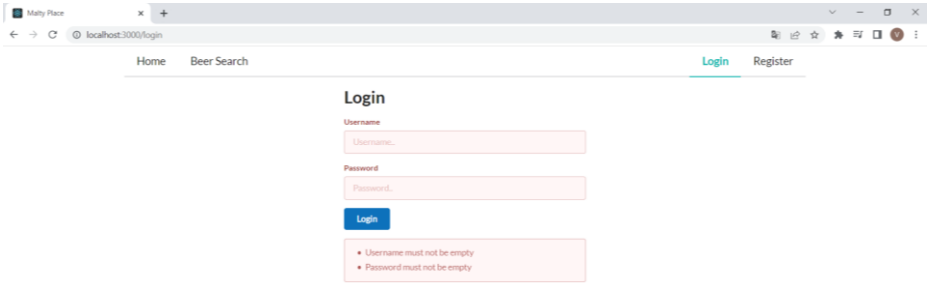
Slika 5.6. *Drugi izgled Home stranice nakon prijavljivanja*

Kao što je prikazano na slici 5.6. nakon uspješnog objavljivanja objave korisniku će uz objavu koju je on objavio biti prikazan gumb za brisanje objave čijim pritiskom se objava briše.



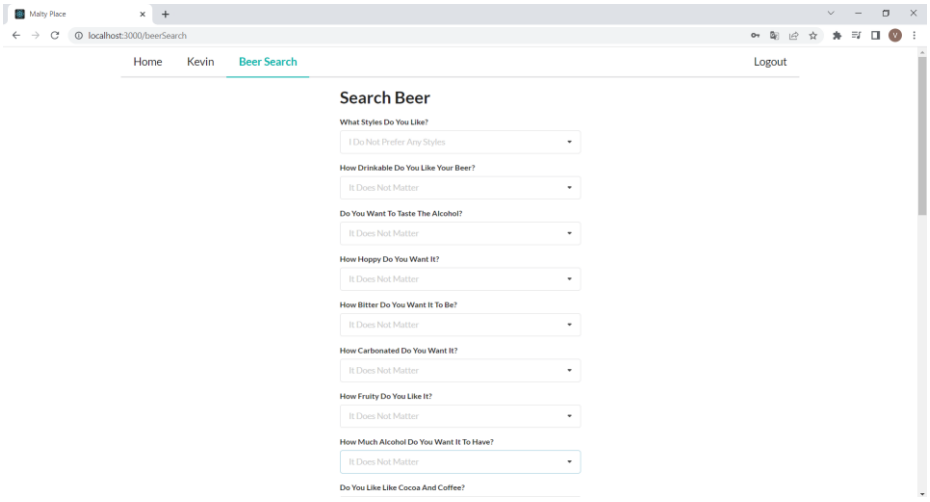
Slika 5.7. *Izgled stranice objave*

Kao što je prikazano na slici 5.7. korisniku su na stranici objave prikazani podaci o objavi i komentari na objavu. Također mu je prikazano polje za pisanje i objavljivanje vlastitih komentara na objavi. Uz komentare koje je prijavljeni korisnik napisao prikazan je gumb za brisanje komentara čijim pritiskom se komentar briše.



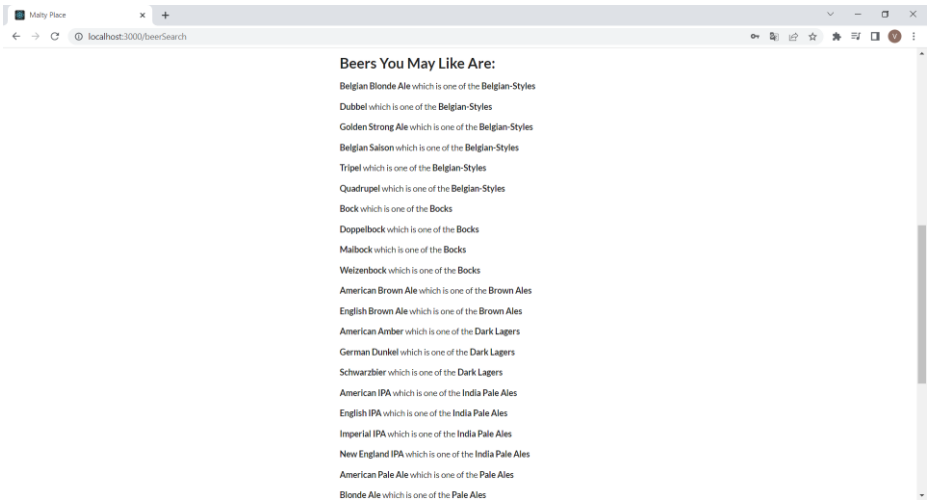
Slika 5.8. *Izgled Login stranice*

Prema slici 5.8. *Login* stranica prikazuje polja za unos korisničkog imena i zaporke. U slučaju da korisničko ime ili zaporka nisu uneseni ili unesena zaporka ne odgovara korisničkom imenu, prikazat će se odgovarajuće poruke upozorenja.

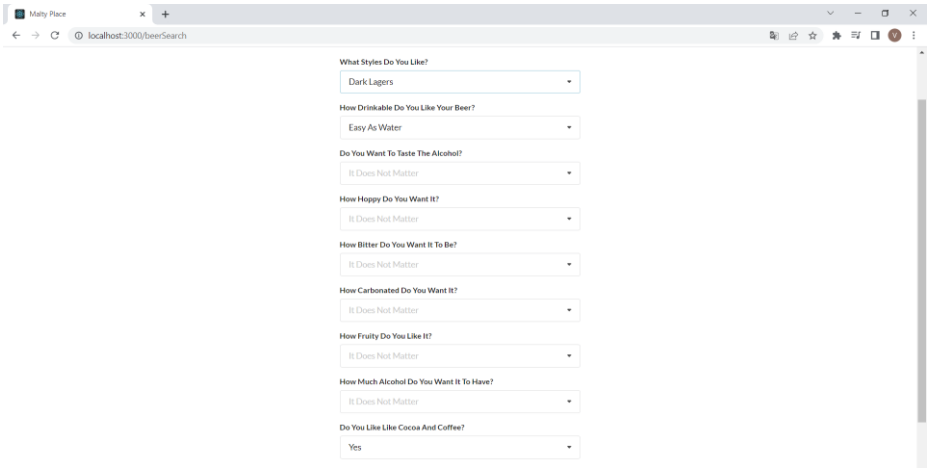


Slika 5.9. *Prvi izgled stranice BeerSearch*

Kao što je prikazano na slici 5.9., na stranici *BeerSearch* prikazan je niz pitanja i izbornika u kojima korisnik može odabrati odgovor na pitanje ili ostaviti ih praznim. Ovisno o odabranim odgovorima na stranici će biti ispisana imena stilova piva koja odgovaraju opisu i ime skupine stilova kojoj oni pripadaju. U slučaju da su sva polja prazna ispisat će se svi stilovi piva kao što je prikazano na slici 5.10.

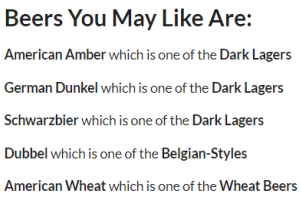


Slika 5.10. *Drugi izgled stranice BeerSearch*



Slika 5.11. *Treći izgled stranice BeerSearch*

Ako je korisnik odgovorio na neka od pitanja kao što je prikazano na slici 5.11. gdje je korisnik rekao da mu se sviđaju tamni lageri (engl. *dark lagers*) stilovi piva i stilovi piva koja su jako pitka i sadrže aromu kakaa onda će se prikazati samo stilovi koji pripadaju tamnim lagerima ili imaju odgovarajuće karakteristike kao što se može vidjeti na slici 5.12.



Slika 5.12. *Četvrti izgled stranice BeerSearch*

6. ZAKLJUČAK

Iako postoje brojne društvene mreže, rijetko koja je usmjerena na rasprave o pivu. Ovaj završni rad pruža društvenu mrežu koja je prvenstveno namijenjena raspravama o pivu i dijeljenju svojeg mišljenja s drugim ljubiteljima piva.

Pomoću web tehnologija kao što su HTML, CSS, React, frameworkova i MongoDB baze podataka korisniku su omogućene sve uobičajene mogućnosti društvene mreže kao što su registriranje, prijavljivanje, pisanje i brisanje objava, pisanje i brisanje komentara i mogućnost uređivanja vlastite stranice profila. Osim navedenih uobičajenih mogućnosti koje nude većina društvenih mreža, korisnik također ima mogućnost odabiranja i prikazivanja stilova piva koja mu se sviđaju i pronalazak novih stilova piva uz unos traženih karakteristika u pivu.

Web aplikacija je dizajnirana vrlo jednostavno i pregledno kako bi ju svaki korisnik brzo naučio koristiti.

Dohvaćanje podataka s baze podataka se odvija vrlo brzo i u pravilu neprimjetno, ali u rijetkom slučajevima naglog izazivanja velikog broja poziva na bazi podataka može doći do neispravnog dohvaćanja podataka koje stranica prikazuje, što znači da prostora za poboljšanje i napredak ima.

LITERATURA

[1] Facebook.com, https://www.facebook.com/ [stranica posjećena 13.09.2022.]

[2] Twitter.com, https://twitter.com/ [stranica posjećena 13.09.2022.]

[3] Instagram.com, https://www.instagram.com/ [stranica posjećena 13.09.2022.]

[4] Untappd.com, https://untappd.com/ [stranica posjećena 13.09.2022.]

[5] BeerBuddy.com, https://beerbuddy.app/ [stranica posjećena 13.09.2022.]

[6] HTML & CSS – W3C, https://www.w3.org/standards/webdesign/htmlcss#whatcss [stranica posjećena 13.09.2022.]

[7] Cascading Style Sheets, level 1, https://www.w3.org/TR/CSS1/ [stranica posjećena 13.09.2022.]

[8] What is JavaScript?, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript [stranica posjećena 13.09.2022.]

[9] What is React?, https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs [stranica posjećena 13.09.2022.]

[10] Semantic UI Guide, https://www.freecodecamp.org/news/semantic-ui-guide/ [stranica posjećena 13.09.2022.]

[11] React Router DOM: How to handle routing in web apps, https://blog.logrocket.com/react-router-dom-tutorial-examples/ [stranica posjećena 13.09.2022.]

[12] What is MongoDB?, https://www.guru99.com/what-is-mongodb.html [stranica posjećena 13.09.2022.]

[13] Introduction to Apollo Server, https://www.apollographql.com/docs/apollo-server/ [stranica posjećena 13.09.2022.]

[14] Računalni klasteri, https://wiki.srce.hr/pages/viewpage.action?pageId=8488260 [stranica posjećena 13.09.2022.]

[15] Container – Semantic UI React, https://react.semantic-ui.com/elements/container/ [stranica posjećena 13.09.2022.]

SAŽETAK

U ovome je projektu napravljena društvena mreža za ljubitelje piva. Svrha ove web aplikacije je ponuditi korisnicima društvenu mrežu koja je prvenstveno namijenjena raspravama vezanim za pivo i stilove piva. Komunikacija među korisnicama se odvija putem objava, komentara i oznakama sviđanja objave. Također je korisnicima omogućeno pretraživanje stilova piva s određenim karakteristikama koje sami biraju. Korištenjem React biblioteke stvorene su sve komponente stranice koje se prikazuju korisniku web aplikacije. Korištenjem biblioteke Semantic UI React samo dizajniranje komponenata je dodatno pojednostavljeno iako se biblioteka Semantic UI React nije pokazala idealnom zbog greške u njezinom izvornom kodu koju mora ispraviti sam programer. MongoDB je omogućio jednostavnu pohranu podataka i pokazao se kao baza podataka koja je vrlo jednostavna za koristiti, pogotovo ako se poveže s kodom putem GraphQLa koji je znatno pojednostavio dohvaćanje i mijenjanje podataka u bazi putem koda.

Ključne riječi: društvena mreža, GraphQL, komunikacija, MongoDB, React

ABSTRACT

A social network for beer lovers

This graduate paper describes development of a web page to serve as a social network for beer lovers. The purpose of this web application is to offer users a social network that is primarily intended for discussions related to beer and beer styles. Communication between users takes place through posts, comments and post likes. It is also possible for users to search for beer styles with specific characteristics that they prefer. All page components that are displayed to the user were created using the React library. By using the Semantic UI React library, the designing of the components itself is further simplified, although the Semantic UI React library did not prove to be ideal due to an error in its source code that must be corrected by the developer himself. MongoDB provided simple data storage and proved to be a very easy-to-use database, especially when connected with the code via GraphQL which greatly simplified retrieving and modifying data in the database via code.

Keywords: social network, GraphQL, communication, MongoDB, React

PRILOZI

Sav programski kod nalazi se na CD-u