Web aplikacija društvene mreže dijeljenja slika

Klasiček, Antonio

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

Stručni studij

WEB APLIKACIJA DRUŠTVENE MREŽE DIJELJENJA SLIKA

Završni rad

Antonio Klasiček

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 18.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime Pristupnika:	Antonio Klasiček
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. Pristupnika, godina upisa:	AI4555, 22.07.2016.
OIB Pristupnika:	91187174047
Mentor:	Robert Šojo, mag. ing. comp.
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	mr.sc. Željko Štanfel
Član Povjerenstva 1:	Robert Šojo, mag. ing. comp.
Član Povjerenstva 2:	Marina Peko, dipl. ing.
Naslov završnog rada:	Web aplikacija društvene mreže dijeljenja slika
Znanstvena grana završnog rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak završnog rada	Web aplikacija društvene mreže koja omogućava svojim korisnicima dijeljenje i komentiranje slika. Kreirati aplikaciju koja omogućava dva tipa korisnika, administrator i korisnik mreže. Administrator jedini ima pravo brisanja i uređivanja svih komentara, kao i slika. Korisnici mreže moraju se registrirati i prijaviti, imaju mogućnost kreiranja objava postavljanjem slika, popratnog sadržaja kao što je naslov i opis. Objave mogu obrisati. Ostali prijavljeni korisnici mogu reagirati na objavu komentarima te se na iste komentare može odgovoriti novima. Aplikacija svim korisnicima omogućava kreiranje višestrukih favorita među slikama, kao i praćenje korisnika. Student treba kreirati funkcionalnu web aplikaciju. Rezervirano za studenta: Antonio
Prijedlog ocjene pismenog dijela ispita (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: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	18.09.2022.

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, 22.09.2022.

Ime i prezime studenta:	Antonio Klasiček
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4555, 22.07.2016.
Turnitin podudaranje [%]:	11

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija društvene mreže dijeljenja slika**

izrađen pod vodstvom mentora Robert Šojo, mag. ing. comp.

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

Web aplikacije su programska rješenja kojima se pristupa putem internet preglednika koristeći internet. Lako su dostupne s računala i mobilnih telefona, a rade na principu povezanosti klijent – server te klijentu pružaju grafičko sučelje kako je definirano na serveru [1].

Aplikacije društvenih mreža izbrisale su vremenska i prostorna ograničenja te su korisnicima omogućile da u kratkom vremenu dođu do željenih informacija. Danas najpoznatije društvene mreže su: instagram, facebook, twitter, itd.

Tema ovog završnog rada je izrada web aplikacije društvene mreže dijeljenja slika koja omogućava dijeljenje slika s drugim korisnicima. Vidljivi dio aplikacije izrađen je *front-end* tehnologijama: React, HTML i CSS. Za pisanje *back-end* koda korišten je Express.js, a kao baza podataka korišten je MongoDB. U ovom radu opisane su sve tehnologije korištene za izradu web aplikacije te je detaljno opisana i logika koja stoji iza samog rada aplikacije te korištenja iste.

U drugom poglavlju napravljen je pregled sličnih rješenja, u trećem poglavlju su opisane tehnologije koje su se koristile u ovom radu, u četvrtom poglavlju su opisane pojedine funkcionalnosti sa stajališta programskog koda, a u zadnjem petom poglavlju prikazane su važne slike aplikacije koje su opisane za lakše snalaženje korisniku koji po prvi puta koristi kreiranu web aplikaciju.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je izrada Web aplikacije društvene mreže dijeljenja slika koja omogućava svojim korisnicima dijeljenje i komentiranje slika. Treba kreirati aplikaciju koja omogućava dva tipa korisnika, administrator i korisnik mreže. Administrator jedini ima pravo brisanja i uređivanja svih komentara, a korisnici mreže se moraju registrirati i prijaviti te imaju mogućnost kreiranja objava postavljanjem slika, popratnog sadržaja kao što je naslov,opis.

2. PREGLED PODRUČJA

U ovom poglavlju navedene su slične web aplikacije koje služe za objavu i dijeljenje slika, videa i ostalog sadržaja.

2.1. Pinterest

Pinterest je aplikacija za pronalaženje ideja (recepti, inspiracije za dom, modu itd.). Korisnici imaju mogućnost stvaranja pinova (eng. *Pins*) pomoću kojih dijele svoje ideje s drugim korisnicima. Na te ideje korisnici mogu reagirati, komentirati te ih dodavati na svoje profile i sortirati prema idejama. Na početnom zaslonu izlistani su pinovi ljudi i tvrtka na temelju korisnikove nedavne aktivnosti i korisnika i ploča koje prate [2].

2.2. Reddit

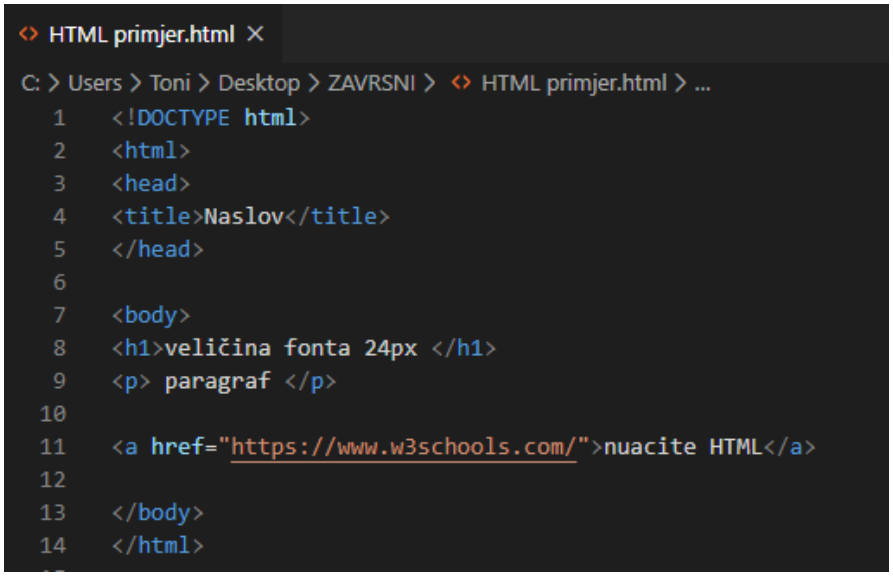
Reddit je društvena mreža na kojoj korisnici mogu dijeliti vijesti i sadržaj. Također, korisnici mogu komentirati i reagirati na objave. Reddit je podijeljen u milijune zajednica, *„subreddits“*, koje pokrivaju različite teme, a kojima se korisnici mogu pridružiti. Na početnom zaslonu korisnicima su prikazane objave iz raznih *„subreddit-a“* koje su trenutačno u trendu.[3].

3. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju će biti objašnjene sve tehnologije koje su se koristile tokom izrade web aplikacije.

3.1. HTML5

HTML (eng. *HyperText Markup Language*) je računalni jezik koji služi za kreiranje elemenata te raspoređivanje sadržaja aplikacije koji se interpretiraju na pregledniku kao web stranica (Slika 2.1.). HTML5 je najnovija verzija HTML-a u kojoj su predstavljeni novi elementi , a ima bolje postupanje s pogreškama te je mnogo jednostavniji u odnosu na starije verzije [4].



Slika 2.1. *Jednostavan primjer HTML dokumenta.*

Na slici 2.1. vidimo da se HTML dokument sastoji od više elemenata i oznaka. Osnovne oznake koji svaki HTML dokument mora imati su:

- <html>: pomoću ove oznake definiramo HTML dokument
- <head>: definira zaglavlje dokumenta u koje pišemo naslov stranice te u njemu određujemo poveznice sa CSS ili JavaScript dokumentom i
- <body>: predstavlja tijelo dokumenta u koje navodimo HTML sadržaj

3.2. CSS3

CSS (eng. *Cascaiding Style Sheets*) je jezik koji opisuje stil i izgled HTML dokumenta. Korištenjem CSS-a definira se boja, pozicija elementa, vrsta fonta, margine, veličina fonta, prozirnost i mnoga druga svojstva. CSS3 je najnovija verzija koja pruža upotrebu novijih svojstava i mogućnosti. Sintaksa CSS-a se sastoji od selektora i deklaracijskog bloka, a deklaracijski blok se sastoji od svojstva i vrijednosti (Slika 2.2.) [5].

Slika 2.2. *Prikaz CSS sintakse* [5].

Selektor se koristi za identifikaciju HTML elementa ili elemenata na koje se primjenjuje navedeni stil. Postoji čitav niz selektora kao npr.:

- Jednostavni selektori (eng. *type selectors*): najjednostavniji su od svih, a odgovaraju imenu HTML oznake i primjenjuju se na svaki istovrsni element u dokumentu
- Klasni selektori: označavaju sve elemente koji sadržavaju jednaku klasu
- Univerzalni selektori: obilježavaju se zvjezdicom (*), a označava sve elemente
- Selektor „div“: označava sve elemente [6].

3.2.1. Načini povezivanja CSS-a s HTML dokumentom

Da bi se definirani CSS mogao primijeniti na dokument, potrebno ga je povezati s HTML dokumentom. Umetanje CSS svojstava u HTML je moguće na tri načina: [7]

- Interno povezivanje: koristi se kada imamo samo jedan HTML dokument (jednu web stranicu), što znači da se rijetko koristi jer nije praktičan za izradu cjelokupne web stranice/aplikacije (Slika 2.3.).

```
<html>
<head>
  <style>
     <!-- Ovdje dolaze CSS svojstva -->
  </style>
</head>
<body>
```

Slika 2.3. *Interno povezivanje* [7].

- Vanjsko povezivanje: posebna CSS datoteka („stil.css“) se povezuje sa HTML dokumentom. Ovaj način je najpraktičniji jer omogućava cijelom web sjedištu korištenje iste CSS datoteke (Slika 2.4).

```
<html>
<head>
  <link rel="stylesheet" href="stil.css" />
</head>
<body>
```

Slika 2.4. *Vanjsko povezivanje* [7].

- Umetanje svojstava u HTML elemente: koristi se svojstvo elementa „style“ unutar kojeg se definira CSS kod (Slika 2.5.).

```
<p style="color:red;">Crvena boja teksta</p>
```

Slika 2.5. *Umetanje svojstava u HTML elemente* [7].

3.3. JavaScript

JavaScript je skriptni ili programski jezik koji omogućuje implementaciju složenih značajki na web stranicama [8]. U JavaScript-u nailazimo na značajke objektno orijentiranog programiranja kao što su klase, objekti enkapsulacije i nasljeđivanje. Koristi se za ostvarivanje dinamičkih i interaktivnih web stranica. JavaScript pruža mnogo mogućnosti kao što su animirani elementi, pozivanje metoda prilikom klika na gumb itd.

Načini povezivanja s HTML dokumentom:

- Pisanje JavaScript koda unutar <*Script*< </*Script*> oznaka u HTML dokumentu
- Lokalno povezivanje kao i kod CSS-a u većini slučajeva najbolja metoda <*Script src*=“ “>
- Vanjsko povezivanje zahtjeva internet adresu do vanjske JavaScript datoteke <*Script src*=“ “> [9].

3.4. MERN stack

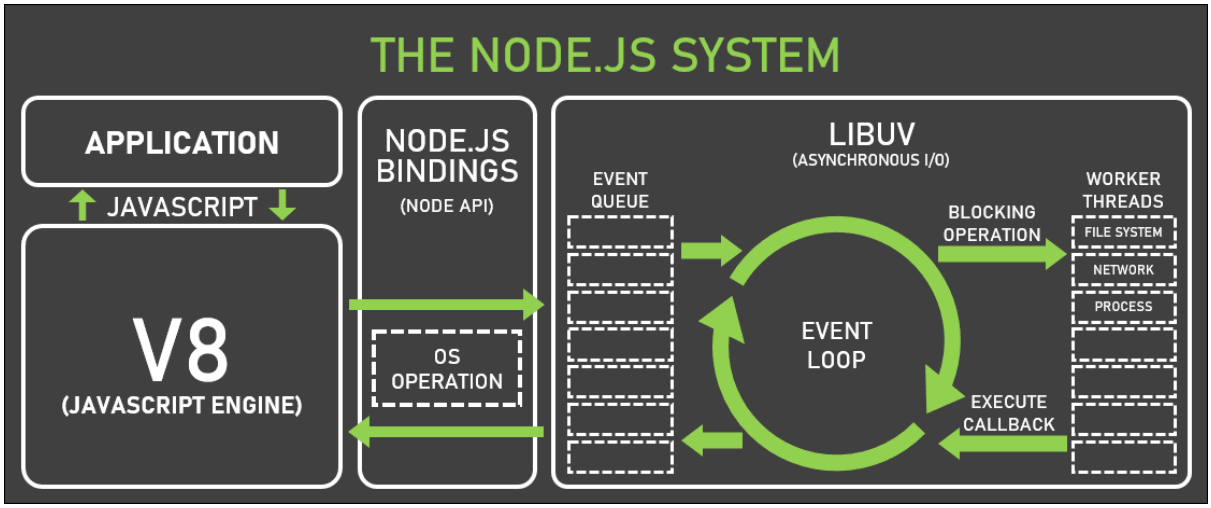
MERN je kratica za MongoDB, Express, React, Node, koja predstavlja četiri ključne tehnologije koje čine hrpu:

- MongoDB — baza podataka
- Express(.js) — Node.js web okvir
- React(.js) — JavaScript okvir na strani klijenta
- Node(.js) — JavaScript web poslužitelj

3.5. Node.js

Node.js je JavaScript okvir koji se bazira na Chrome-ovom V8 JavaScript engine-u, a omogućava da se JavaScript kod pokreće na poslužitelju, odnosno bilo kojem računalu izvan web preglednika [10]. Node.js teži ne blokirajućem i asinkronome načinu programiranja. Najveća prednost Node.js-a je što se svi događaji (eng. *events*) definirani u Node.js-u dodaju

u tzv. skup događaja (eng. *event pool*) koji se pokreću bez blokiranja daljnjeg izvršavanja koda. Na taj način se eliminira čekanje i jednostavno se nastavlja sa sljedećim zahtjevom [11] (Slika 2.6.).



Slika 2.6. *Tijek kompilacije u Node.js-u* [11].

Još jedna karakteristika Node.js-a je upravljanje paketima NPM (eng. *Node Package Manager*). Node Package Manager služi za dodavanje paketa u Node.js okruženje. Prilikom ovakvog dodavanja paketa u Node.js ne moramo razmišljati gdje se i koje datoteke uključuju. Paketi u obliku JavaScript biblioteka su dodaci Node.js okvira koji pomažu prilikom razvoja koda [12].

3.6. React

React.js je JavaScript okvir za izradu korisničkih sučelja. Zahvaljujuću deklarativnom konceptu omogućava jednostavnost prilikom izrade web aplikacija. Deklarativan koncept radi na način da, prilikom promjene podataka, React prepozna i ažurira samo promijenjene dijelove [13]. React slijedi princip arhitekture temeljene na komponentama. Komponenta u Reactu je izolirani dio koda koji se može ponovno koristiti. Dvije vrste komponenti u React su klasne komponente i funkcijske komponente. Prije React verzije 16.8, programeri su mogli rukovati stanjem (eng. *state*) i drugim React značajkama koristeći samo klasne komponente, ali s verzijom 16.8 React je uveo novi obrazac pod nazivom kuke (eng. *hooks*).

Uz React kuke (eng. *hooks*) može se koristiti stanje (eng. *state*) i druge React značajke unutar funkcijskih komponenti što omogućuje funkcijsko programiranje u Reactu [14].

3.6.1. JSX (JavaScript Syntax Extension)

JSX je sintaksa slična HTML-u/XML-u koju koristi React. JSX proširuje ECMAScript te dozvoljava pisanje HTML elemenata unutar JavaScript-a. Sintaksa je namijenjena za upotrebu od strane pred-procesora „*transpilera*“ za transformaciju tokena u ECMAScript [15].

3.6.2. Kuke (eng. *Hooks*)

Kuke omogućavaju korištenje stanja (eng. *state*) i drugih značajki Reacta bez pisanja klasa. Kuke (eng. *hooks*) korištene u ovom radu su: *useState*, *useEffect*, *useContext*, *useRef* (Slika 2.7.) [16].

```
import React, { useContext, useState, useRef,useEffect } from "react";
```

Slika 2.7. *Import Hoookova.*

UseState vraća vrijednost stanja i funkciju za ažuriranje. Tijekom početnog iscrtavanja (eng. *render*) vraćeno stanje je jednako vrijednosti proslijeđenoj kao prvi argument „*initialState*“. Funckija *setState* koristi se za ažuriranje stanja. Prihvaća novu vrijednost stanja te stavlja u red ponovno iscrtavanje komponente (Slika 2.8.) [16].

```
const [state, setState] = useState(initialState);
```

Slika 2.8. *Prikaz useState hook-a* [16].

UseEffect prihvaća dva argumenta, funkciju i ovisnost (eng. *dependency*), gdje je ovisnost neobavezna (eng. *optional*). Funkcija proslijeđena *useEffect*-u pokrenut će se nakon iscrtavanja (eng. render) (Slika2.9.) [16].

```
useEffect(() => {
  fetchPosts();
  const userData = JSON.parse(window.sessionStorage.getItem("User"));
  setUser(userData);
}, []);
```

Slika 2.9. *Primjer useEffect-a.*

UseContext prihvaća objekt konteksta i vraća trenutnu vrijednost konteksta. Trenutna vrijednost konteksta određena je propozicijom najbližeg *<MyContext.Provide>* iznad pozivajuće komponente u stablu. Kada se najbliži *<MyContext.Provider>* iznad komponente ažurira, ova kuka će pokrenuti ponovno iscrtavanje s najnovijom vrijednošću konteksta proslijeđenom pružatelju *MyContext* (Slika 2.10. i 2.11.) [16].

```
import React, { createContext, useContext } from "react";
import {
  UserRouteService,
  PostRouteService,
  CommentRouteService,
  FavoriteRouteService,
  FollowedRouteService,
} from "../services";

const services = {
  userRouteService: new UserRouteService("users"),
  postRouteService: new PostRouteService("posts/"),
  commentRouteService: new CommentRouteService("comments/"),
  favoriteRouteService: new FavoriteRouteService("favorites/"),
  followedRouteService: new FollowedRouteService("following/"),
};

export const ServiceContext = createContext({ services: null });

export const ServiceProvider = ({ children }) => {
  return (
    <ServiceContext.Provider value={services}>
      {children}
    </ServiceContext.Provider>
  );
};

export const useService = () => useContext(ServiceContext);
```

Slika 2.10. *Kreiranje i pružanje Contexta svim „child“ komponentama.*

```
const { commentRouteService } = useContext(ServiceContext);
```

Slika 2.11. *Destrukturiranje iz najbližeg „serviceContex“ konteksta.*

UseRef vraća promjenjivi (eng. *mutable*) *ref* objekt kojemu je trenutna vrijednost proslijeđena argumentom *„Initial State“*. Vraćeni objekt će postojati dok god postoji komponenta. Najčešće se koristi za izravan pristup DOM elementu (Slika 3.12.) [16].

```
const refContainer = useRef(initialValue);
```

Slika 2.12. *Primjer useRef-a.*

3.7. Express.js

Express.js je okvir (eng. *framework*) Node.js-a. Express.js omogućava lako definiranje posebnih upravljačkih događaja (eng. *event handlers*) ovisno o HTTP metodi (POST, DELETE, GET, PUT itd.) [17].

Express.js omogućava:

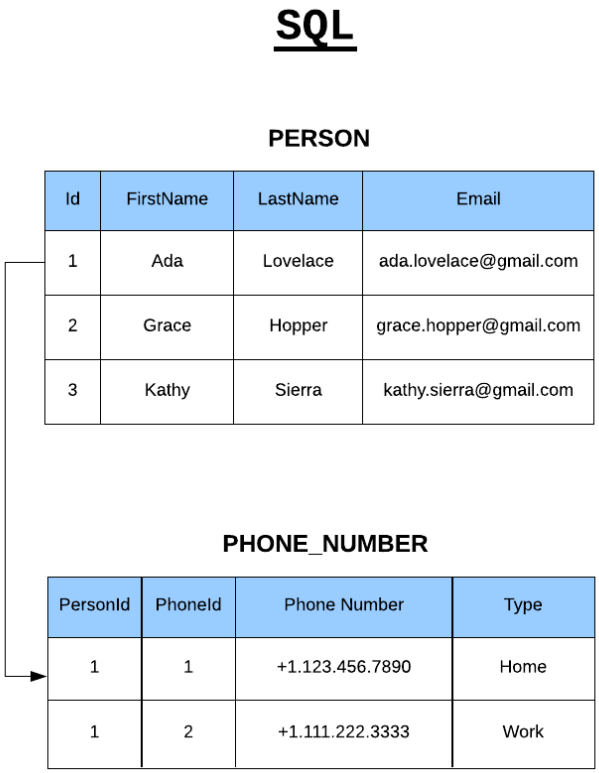
- Korištenje *midleware*-a za obradu zahtjeva
- Razvoj koda cijele aplikacije u samo jednom programskom jeziku (JavaScript)
- Brzo povezivanje sa bazom podataka (npr. MySQL, MongoDB itd.) [17].

Iako se back-end dio može pisati koristeći „čisti“ JavaScript u Node.js-u, takav način je kompliciraniji jer moduli Node.js-a nisu dovoljno prilagođeni za brzi razvoj koda. Zato se koristi Express.js koji olakšava pisanje koda.

3.8. MongoDB

SQL i NoSQL predstavljaju dva tipa baze podataka. Većinom se koriste SQl (SQL – Strukturirani jezik za upite, eng. *Structured Query Language*) baze podataka u kojima je struktura jasno i strogo definirana (Slika 2.13.). Na ovom projektu korištena je MongoDB,

NoSQL baza podataka, u kojoj je struktura podataka fleksibilna i nadogradiva. Korištenje NoSQL-a ubrzava razvoj aplikacija i smanjuje složenost postavljanja. Glavni strukturni elementi u MongoDB-u su tzv. dokumenti. Dokumenti su pisani u JSON (*JavaScript Object Notation*) obliku te su svojom strukturom slični JavaScript objektima (Slika 2.14.) [18].

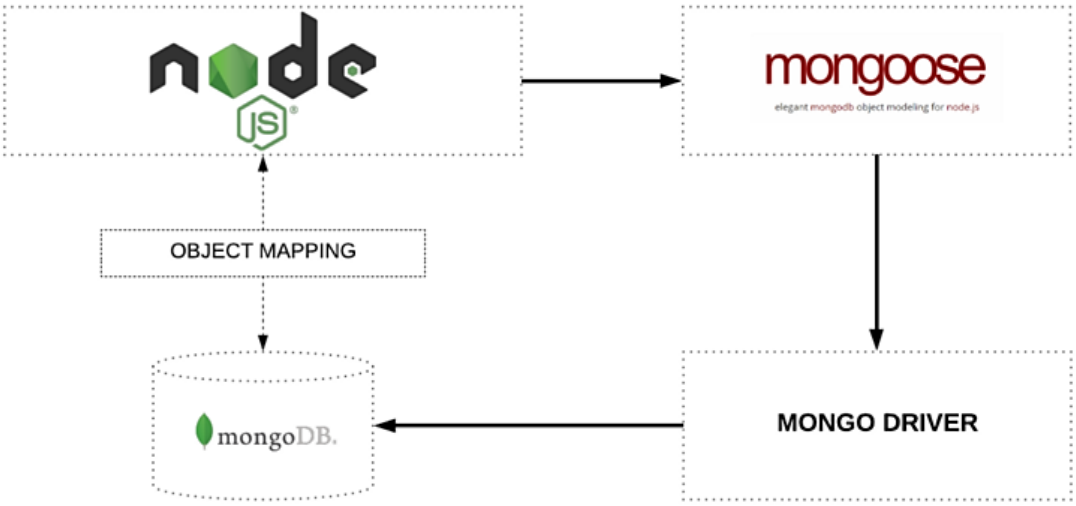


Slika 2.13. *Pohrana podataka u SQL bazi podataka* [19].

Slika 2.14. *Pohrana podataka u MongoDB (NoSQL)* [19].

3.9. Mongoose

Mongoose je biblioteka za modeliranje objektnih podataka (ODM) za MongoDB i Node.js. Upravlja odnosima između podataka, pruža provjeru valjanosti sheme i koristi se za prevođenje između objekata u kodu i reprezentacije tih objekata u MongoDB-u (Slika 2.16.) [19].



Slika 2.16. *Mapiranje objekata između Node-a i MongoDB-a upravljano putem Mongoosea* [19].

3.10. Axios

Jedna od temeljnih zadaća svake web aplikacije je komunikacija s poslužiteljima putem HTTP protokola. To se lako postiže korištenjem Fetcha ili Axiosa. U ovoj aplikaciji za komunikaciju s poslužiteljima putem HTTP protokola korišten je Axios. Axios je HTTP *„promised-based“* klijent za Node.js i preglednik. Izoforman je, što znači da se može izvoditi u pregledniku i Node.js-u s istim *„codebase-om“*. Na strani poslužitelja koristi izvorni Node.js http modul, dok na strani klijenta (pregledniku) koristi XMLHttpRequestos (Slika 2.15.) [20].

```
import axios from "axios";

class Api {
  baseUrl = "http://localhost:2000/";

  constructor(endpoint) {
    this.baseUrl = this.baseUrl + endpoint;
  }

  async get(payload) {
    return await axios.get(this.baseUrl, payload);
  }

  async put(payload) {

    return await axios.put(this.baseUrl, payload);
  }

  async post(payload) {

    return await axios.post(this.baseUrl, payload);
  }

  async delete(payload) {
    const baseUrl = this.baseUrl + payload;
    return await axios.delete(baseUrl, payload);
  }
}

export default Api;
```

Slika 2.15. *Api klasna komponenta.*

3.11. Lodash

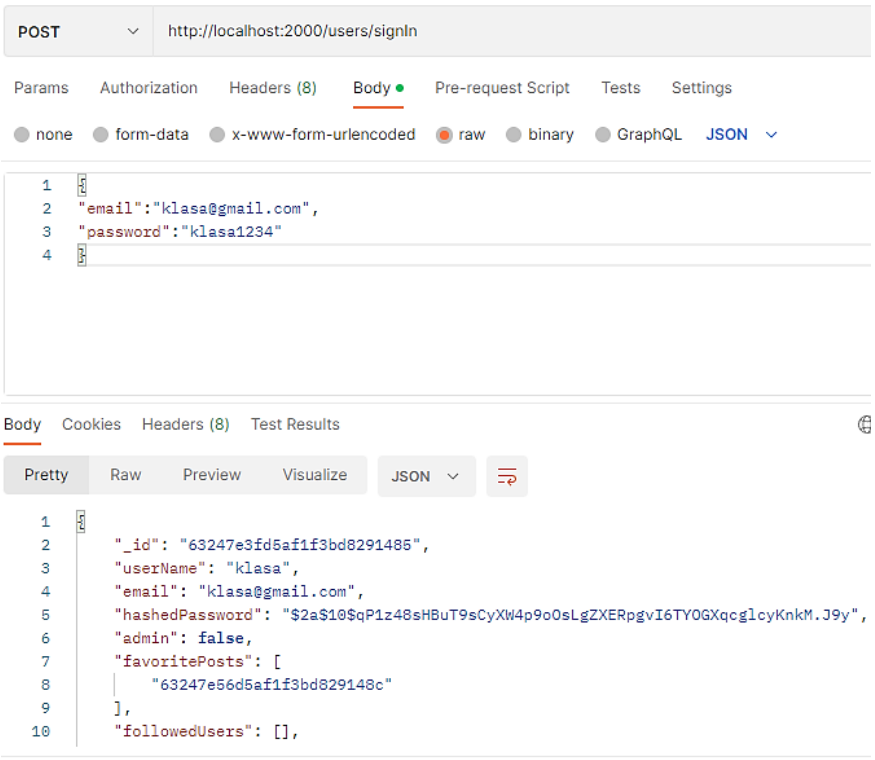
Lodash je JavaScript biblioteka koja pruža uslužne funkcije za uobičajene programerske zadatke koristeći paradigmu funkcionalnog programiranja (Slika 2.17.). Lodash ima nekoliko ugrađenih (eng. *Built-in*) uslužnih funkcija koje kodiranje u JavaScript-u čine lakšim i "čistijim". Umjesto pisanja uobičajenih funkcija, uvijek iznova, zadatak se može izvršiti s jednom linijom koda.

```
import lodash from "lodash";
const { compact } = lodash;
```

Slika 2.17. *Lodash metoda (compact).*

3.12. Postman

Postman je Api platforma koja omogućava slanje Api request-ova. Na ovom projektu korišten je za provjeru ispravnosti *back-end* koda prije pisanja *front-end* koda (Slika 2.18.).



Slika 12.18. *Post-request (login korisnika).*

3.13. Visual Studio Code

Visual Studio Code je veoma popularan Microsoftov uređivač teksta (eng. *text editor*).

Razlozi zbog kojih je popularan:

- Jednostavan za koristit
- Besplatan
- Otvorenog koda

- Dostupan na svim OS (operacijskim sustavima)
- Podržava velik broj programskih jezika
- Automatsko prepoznavanje sintakse
- Ispravljanje grešaka
- Manipulacija nad više linija koda u isto vrijeme

4. IZRADA APLIKACIJE

U ovom poglavlju opisan je kod koji je napisan pomoću prethodno navedenih tehnologija.

Na početku je instalirana najnovija verzija Node.js okruženja. Aplikacija je podijeljena na klijent i server stranu.

Unutar klijent mape kreirana je „prazna“ React aplikacija uz pomoć naredbe (*npx create-react-app*) te su instalirane sve potrebne ovisnosti (eng. *dependecy*):

- Axios za kreiranje api request-ova
- Eeact-filebase-64 za pretvorbu (eng. *convert*) slika

Unutar servera kreiran je Index.js file koji predstavlja *„starting point“* server aplikacije te je uz pomoć naredbe (*npm init – y*) inicijaliziran prazan package.js-on (Slika 4.1. i 4.2.). Nakon inicijalizacije package.js-ona instalirane su sve potrebne ovisnosti (eng. *dependency*):

- Bodyparser omogućava slanje post request-ova
- Cors omogućava kros origine requests
- Express okvir pomoću kojeg je pisan *back-end* kod na ovoj aplikaciji
- Mongoose uz pomoć kojega se kreiraju modeli za postove
- Nodemon je alat koji pomaže u razvoju aplikacija temeljnih na Node.js-u. Kada se otkriju promjene datoteke u direktoriju, Node aplikacija se ponovno pokrene. Jednostavno rečeno prilikom izmijene koda ne mora se ponovno pokretati server kako bi se učitale promjene.
- Bcrypt.js služi za sigurno raspršivanje (eng. *hash*) i slanje lozinki

```
import express from "express";
import cors from "cors";
import mongoose from "mongoose";
import userRoutes from "./routes/user.js";
import postRoute from "./routes/post.js";
import commentRoute from "./routes/comment.js";
import favoriteRoute from "./routes/favorite.js";
import followingRoute from "./routes/following.js";
import bodyParser from "body-parser";

const app = express();

app.use(bodyParser.json({ limit: "30mb", extended: true }));
app.use(bodyParser.urlencoded({ limit: "30mb", extended: true }));

app.use(cors());

app.use("/users", userRoutes);
app.use("/posts", postRoute);
app.use("/comments", commentRoute);
app.use("/favorites", favoriteRoute);
app.use("/following", followingRoute);

app.listen(2000, () => console.log("Example app listening on port 2000!"));

mongoose.connect(
  "mongodb+srv://Klasa:Klasa159357000@cluster0.chijg.mongodb.net/Users?retryWrites=true&w=majority"
);
```

Slika 4.1. *Index.js file.*

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  ▷ Debug
  "scripts": {
    "start": "nodemon index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.0",
    "cors": "^2.8.5",
    "express": "^4.18.1",
    "lodash": "^4.17.21",
    "mongoose": "^6.4.4",
    "nodemailer": "^6.7.8",
    "nodemon": "^2.0.19"
  }
}
```

Slika 4.2. *Package.js-on server.*

4.1. Baza podataka

U ovoj aplikaciji, kao baza podataka korišten je MongoDB (NoSQL baza podataka), koja u sebi sadrži kolekcije dokumenata. Kreirana je baza podataka pod nazivom *„application“*, a konekcija na bazu izvršena je pomoću mongoose.connect() metode (Slika 4.3.). *„Application“* baza u sebi sadrži 3 kolekcije (users, posts, comments) (Slika 4.4.):

- Users - kolekcija sadrži dokumente koji u sebi imaju podatke o korisniku
- Posts – kolekcija sadrži dokumente koji u sebi imaju podatke o objavama
- Comments – kolekcija sadrži dokumente koji u sebi imaju podatke o komentarima

```
mongoose.connect(
  "mongodb+srv://Klasa:Klasa159357000@cluster0.chijg.mongodb.net/Users?retryWrites=true&w=majority"
);
```

Slika 4.3. *Konekcija na bazu.*

a)
```
_id: ObjectId('6324a87d06d841525acd87d3')
serName: "sora"
email: "sora@gmail.com"
hashedPassword: "$2a$10$TU4lV56gKziKeYnp7h9VOeGQ6Q.8yJ4cV47HI2dnYjLwrwpdYGM4u"
admin: false
> favoritePosts: Array
> followedUsers: Array
__v: 0
```

b)
```
_id: ObjectId('632482c6d5af1f3bd8293ab2'
> content: Object
> comments: Array
userId: ObjectId('63247f86d5af1f3bd82914
__v: 0
```

c)
```
_id: ObjectId('632483d3d5af1f3bd8294d78')
content: "test123"
userId: ObjectId('63247e3fd5af1f3bd8291485')
timeStamp: 2022-09-16T14:10:27.125+00:00
commentId: "632483ced5af1f3bd8294cc9"
> replies: Array
__v: 0
```

Slika 4.4. a) *User dokument* **b)** *Post dokument* **c)** *Comment dokument.*

4.2. Prijava i registracija korisnika

Kako bi korisnik imao pristup web aplikaciji potrebna je registracija i prijava.

React funkcijska komponenta (RegisterForm.jsx) sadrži polje za unos email-a, korisničkog imena, lozinke i ponovan unos lozinke čime se povećava sigurnost. Prilikom *„submita“* forme šalje se HTTP POST request na server te se kroz *„body“* šalju svi podaci iz forme (Slika 4.5.). Server radi niz provjera podataka te ukoliko podaci „prođu“ kroz sve provjere kreira se novi user dokument. Kod prijave korisnika React funkcijska komponenta (LoginForm.jsx) sadrži polje za unos email-a i lozinke. Kao i kod registracije šalje se POST request.

```
const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    await userRouteService.postSignUp(formValues);
    handleToastMessage();
    setToggle(!toggle);
  } catch (error) {
    setErrorMesage(error.response.data.message);
  }
};
```

Slika 4.5. *POST request (register).*

Slika 4.6. prikazuje provjere prilikom registracije novog korisnika. Definirane su 4 varijable koje „dohvaćaju“ podatke poslane u *„body-u“*. Provjerava se preklapaju li se lozinka i ponovan unos lozinke te postoji li user s istim email-om. Ukoliko podaci zadovolje sve provjere kreira se novi dokument u kolekciji user. Međutim, ako uvjeti nisu zadovoljeni kao odgovor (*„response“*) se šalje „error“ poruka.

```
export const signUp = async (req, res) => {
  const { userName, email, password, admin, confirmPassword } = req.body;

  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = await User.create({
      userName,
      email,
      hashedPassword,
      admin,
    });
    if (password === confirmPassword) {
      res.status(201).json(newUser);
    } else {
      return res.status(400).json({ message: "Password do not match" });
    }
  } catch (error) {
    if (error.code == 11000) {
      return res
        .status(400)
        .json({ message: "User with that email already exist!" });
    }
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.6. *Provjera podataka prilikom registracije novog korisnika.*

Još jedna bitna stavka kod sigurnosti je raspršivanje lozinke (eng. *password hash*). Funkcija raspršivanja prima vrijednost, u ovome slučaju lozinku i pretvara ju u dugački niz slova i znakova definirane veličine. Na taj način admin ne može vidjeti lozinke korisnika u bazi. Isto tako, ako napadač ima pristup bazi, ne može razotkriti raspršenu lozinku.

Na ovom projektu za raspršivanje korištena je bcrypt.hash funkcija, a prilikom login-a za usporedbu lozinke sa hashanom lozinkom iz baze korištena je funkcija bcrypt.compare.sync (Slika 4.7.).

```
bcrypt.compareSync(password, user.hashedPassword);
```

Slika 4.7. *bcrypt.compare.syn funkcija.*

Nakon uspješne prijave, kreira se novi item u *„session storage-u“* User i korisnik pristupa aplikaciji. Prilikom odjave item User se briše iz *„session storage-a“* i korisnika se „odvodi“ na prijavu (Slika 4.8 i 4.9.).

```
const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    const response = await userRouteService.postSignIn(formValues);
    window.sessionStorage.setItem("User", JSON.stringify(response.data));
    navigate("/posts");
  } catch (error) {
    console.log(error);
    setErrorMesage(error.response.data.message);
  }
};
```

Slika 4.8. *POST request (login).*

```
const logOut = () => {
  window.sessionStorage.removeItem("User");
  navigate("/");
};
```

Slika 4.9. *Logout.*

Korisnik također ima opciju izmijene imena i lozinke.

PUT request, na server kroz *„body“*, šalje podatke: userId, updateUserName/updatePassword ili updateUserName+updatePassword (Slika 4.10.).

```
    setReadOnly(!readOnly);
    const response = await userRouteService.updateUser({
      userId: user._id,
      updatedUserName: updatedUsername,
      updatedPassword: updatedPassword,
    });
    setSuccessMsg(response.data.message);
  };
```

Slika 4.10. *PUT request za ažuriranje imena/lozinke.*

Server, kroz „body“, „dohvaća“ podatke pomoću userId-a i pronalazi odgovarajućeg korisnika te ga ažurira (Slika 4.11.).

```
export const changeUserInfo = async (req, res) => {
  const { userId, updatedUserName, updatedPassword } =
    req.body;
  const updatedHashedPassword = await bcrypt.hash(updatedPassword, 10);
  try {
    await User.findOneAndUpdate(
      { _id: userId },
      { userName: updatedUserName, hashedPassword: updatedHashedPassword }
    );
    return res.status(200).json({ message: "Succesfuly changed user info" });
  } catch (error) {
    res.status(500).json(error);
  }
};
```

Slika 4.11. *Ažuriranje imena/lozinke.*

4.3. Kreiranje i brisanje objava

Nakon uspješne prijave korisnik pristupa aplikaciji. Glavna značajka aplikacije je kreiranje objava.

Komponenta (PostForm.jsx) sadrži polje za unos naslova (eng. *Title*) i polje tipa *„file“* gdje se *„uploada“* slika. Nakon *„submit-a“* forma šalje HTTP POST request te se serveru kroz *„body“* šalju podaci: Content (title i slika), UserId i TimeStamp (konstruktor new Date() vraća niz koji predstavlja trenutno vrijeme) (Slika 4.12.).

```
const handleSubmit = async (e) => {
  e.preventDefault();
  const userId = JSON.parse(window.sessionStorage.getItem("User"))._id;
  try {
    await service.post({
      content: content,
      userId,
      timeStamp: new Date(),
    });
    fetchPosts();
    setContent(intialState);
    fileInputRef.current.value = "";
  } catch (error) {
    console.log(error);
  }
};
```

Slika 4.12. *POST request (objava).*

Server kroz koji „body“ „dohvaća“ poslane podatke i kreira novu objavu (Slika 4.13.).

```
export const createPost = async (req, res) => {
  const { content, userId, timeStamp } = req.body;
  try {
    const newPost = await Post.create({
      content,
      comments: [],
      userId,
      timeStamp,
    });
    res.status(201).json(newPost);
  } catch (error) {
    return res.status(500).json({ message: error.message });
  }
};
```

Slika 4.13. *Kreiranje objave.*

Bitna stavka prilikom brisanja objava je brisanje pripadajućih komentara. Za razliku od POST i PUT request-ova koji podatke šalju kroz *„body“*, HTTP DELETE request serveru podatke šalje putem *„params-a“*. Jednostavnije rečeno, serveru se podaci šalju kroz rutu.

Detaljan opis HTTP DELETE request-a: klikom na delete (*„trash“* ikonicu) pozove se deletePost funkcija. deletePost funkcija poziva delete metodu te joj kao parametar šalje id objave (Slika 4.14). Metoda delete prima *„params“* vrijednosti i pridodaje ih baseUrl-u. Na taj način se putem *„params-a“* serveru šalje id objave koju je potrebno obrisati (Slika 4.15.).

```
const deletePost = async (id) => {
  const response = await service.delete(id);
  fetchPosts();
};
```

Slika 4.14. *Delete request (brisanje objave).*

```
  async delete(params) {
    const baseUrl = this.baseUrl + params;
    return await axios.delete(baseUrl, params);
  }
}
```

Slika 4.15. *Definiranje delete request rute.*

Na serveru je također potrebo definirati rutu kako bi se kroz „params“ uspješno dohvatio id objave (Slika 4.16.).

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

Slika 4.16. *Server ruta.*

Brisanje objave (Slika 4.17.):

- Server dohvaća podatak (id objave) kroz *„params“*
- findOne pronalazi odgovarajuću objavu
- provjera sadrži li objava komentare
- ukoliko objava sadrži komentare, brišu se svi komentari
- findByIdAndRemove briše objavu

```
export const deletePost = async (req, res) => {
  const deletePost = await Post.findOne({ _id: req.params.id });
  if (deletePost.comments.length > 0) {
    await Comment.remove({
      _id: {
        $in: deletePost.comments,
      },
    });
  }
  Post.findByIdAndRemove({ _id: req.params.id }).then(function (post) {
    res.send(post);
  });
};
```

Slika 4.17. *Brisanje objave.*

4.4. Kreiranje komentara

Korisnik ima mogućnost komentiranja objava i odgovara na komentare.

HTTP POST request komentara i odgovora *„pucaju“* na istu rutu. U tom slučaju komentari i odgovori se spremaju u istu kolekciju. Ovisno o poslanim podacima, server uz pomoć definiranih provjera, razlikuje komentare od odgovora te komentare pridružuje odgovarajućoj objavi, a odgovore pridružuje odgovarajućem komentaru (Slika 4.18.).

a)

```
const submitComment = async () => {
  await commentRouteService.post({
    content: comment,
    userId: user._id,
    timeStamp: new Date(),
    postId: data._id,
  });
  await fetchPosts();
  setComment("");
};
```

b)

```
const submitReply = async () => {
  await commentRouteService.post({
    content: reply,
    userId: user._id,
    timeStamp: new Date(),
    commentId: comment._id,
  });
  handleReplyToggle();
  await fetchPosts();
};
```

Slika 4.18. a) *POST request (comment)* **b)** *POST request (reply).*

Server kroz *„body“* dohvaća podatke i kreira novi *„comment“* dokument. Nakon kreiranja novog dokumenta slijedi provjera je li kroz *„body“* poslan commentId. Ukoliko je, id novog komentara se *„push-a“* u polje *replies* odgovarajućeg komentara. Ukoliko commmentId nije poslan, id novog komentara se *„push-a“* u polje *comments* odgovarajuće objave (Slika 4.19.).

```
export const addComment = async (req, res) => {
  const { content, userId, timeStamp, postId, commentId } = req.body;

  try {
    const newComment = await Comment.create({
      content,
      userId,
      timeStamp,
      commentId,
    });

    if (commentId) {
      await Comment.findOneAndUpdate(
        { _id: commentId },
        { $push: { replies: newComment.id } },
        { new: true }
      );
      res.status(201).json();
    }

    await Post.findOneAndUpdate(
      { _id: postId },
      { $push: { comments: newComment.id } },
      { new: true }
    );

    res.status(201).json();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.19. *Kreiranje komentara.*

HTTP PUT request na server kroz *„body“* šalje podatke: updatedContent, commentId i updatedTimeStamp. Kod slanja request-a za uređivanje komentara, ključno je poslati id komentara koji se uređuje (Slika 4.20.).

```
const editComment = async () => {
  setReadOnly(!readOnly);
  await commentRouteService.put({
    updatedContent: updatedComment,
    commentId: data._id,
    updatedTimeStamp: new Date(),
  });
  await fetchPosts();
};
```

Slika 4.20. *PUT request (comment).*

Server kroz *„body“* „dohvaća“ podatke te uz pomoć metode findOneAndUpdate filtrira komentare po Id-u, pronalazi odgovarajući komentar i ažurira podatke (Slika 4.21.).

```
export const editComment = async (req, res) => {
  const { updatedContent, updatedTimeStamp, commentId } = req.body;

  try {
    const editedComment = await Comment.findOneAndUpdate(
      { _id: commentId },
      { content: updatedContent, timeStamp: updatedTimeStamp },
      { upsert: true, new: true }
    );

    res.status(201).json(editedComment);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.21. *Ažuriranje komentara.*

Kao što je prilikom brisanja objave bitno obrisati pripadajuće komentare, tako je i prilikom brisanja komentara bitno obrisati sve pripadajuće odgovore.

HTTP DELETE request putem *„params-a“* serveru šalje jednu od dvije kombinacije podataka (Slika 4.22.i 4.23.):

- commentId i postId (brisanje komentara)
- replyId i commentId (brisanje odgovora)

```
const deleteComment = async ({ replyId, commentId, postId }) => {
  await commentRouteService.delete({ replyId, commentId, postId });
  fetchPosts();
};

async delete(params) {
  const { commentId, replyId, postId } = params;
  const baseUrl = `${this.baseUrl}${commentId}/${replyId}/${postId}`;
  return await axios.delete(baseUrl, params);
}
```

Slika 4.22. *DELETE request za brisanje komentara.*

```
router.delete("/:commentId/:replyId/:postId", deleteComment);
```

Slika 4.23. *Definiranje rute na serveru.*

Server kroz *„params“* dohvaća poslane podatke. Vrši se provjera je li poslan replyId. Ukoliko nije poslan, znači da je komentar tipa komentar. Komentar i njegovi pripadajući odgovori se brišu te se komentar *„pulla“* iz odgovarajućeg *„comments“* polja. Ukoliko je „replyId“ poslan, znači da je komentar tipa odgovor. Odgovor se briše i *„pulla“* iz odgovarajućeg *„replies“* polja (Slika 4.24.).

```
export const deleteComment = async (req, res) => {
  const { commentId, replyId, postId } = req.params;
  try {
    const deleteComment = await Comment.findOne({
      _id: commentId,
    });
    if (replyId == "undefined") {
      await Comment.remove({
        _id: {
          $in: deleteComment.replies,
        },
      });
      await Post.findOneAndUpdate(
        { _id: postId },
        { $pull: { comments: { $in: [commentId] } } },
        { new: true }
      );

      Comment.findByIdAndRemove({ _id: commentId }).then(function (comment) {
        res.send(comment).json();
      });
    } else {
      await Comment.findOneAndUpdate(
        { _id: commentId },
        { $pull: { replies: { $in: [replyId] } } },
        { new: true }
      );

      Comment.findByIdAndRemove({ _id: replyId }).then(function (reply) {
        res.send(reply).json();
      });
    }
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.24. *Brisanje komentara.*

4.5. Dodavanje i brisanje favorit objava

Prilikom dodavanja favorit postova, serveru je potrebno poslati id trenutno prijavljenog korisnika i id posta koji se dodaje u favorite.

HTTP POST request kroz „body“ serveru šalje postId i userId (Slika 4.25.).

```
const addToFavorites = async () => {
  await favoriteRouteService.post({
    postId: post._id,
    userId: user._id,
  });
  await fetchFavoriteList();
};
```

Slika 4.25. *POST request (kreiranje objave).*

Server kroz *„body“* dohvaća userId i postId te pomoću userId-a pronalazi odgovarajućeg korisnika i unutar polja *„favoritePosts“ „pusha“* id objave. Na taj način se u bazi Useru postavljaju favorit postovi (Slika 4.26.).

```
export const addFavoritePost = async (req, res) => {
  const { userId, postId } = req.body;

  try {
    await User.findOneAndUpdate(
      { _id: userId },
      { $push: { favoritePosts: postId } },
      { new: true }
    );
    res.status(201).json();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.26. *Dodavanje favorit objava.*

Prilikom brisanja favorit objava HTTP DELETE request na server kroz *„params“* šalje podatke: userId i postId (Slika 4.27.).

```
const deleteFromFavorites = async ({ userId, postId }) => {
  await favoriteRouteService.deletePostFromFavorites({ userId, postId });
  await fetchFavoriteList();
  await fetchFavoritePosts();
};
```

Slika 4.27. *DELETE request (brisanje favorit objava).*

Server kroz *„params“* dohvaća userID i postID. Pronalazi odgovarajućeg korisnika te *„pulla“* postId iz „favoritePosts“ polja (Slika 4.28.).

```
export const deleteFavoritePost = async (req, res) => {
  const { userId, postId } = req.params;

  try {
    await User.findOneAndUpdate(
      { _id: userId },
      { $pull: { favoritePosts: postId } },
      { new: true }
    );
    res.status(201).json();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.28. *Brisanje favorit objava.*

4.6. Praćenje korisnika

Prilikom praćenja i prestanka praćenja korisnika, serveru je potrebno poslati id trenutno prijavljenog korisnika te id korisnika kojeg se želi pratiti.

Praćenjem korisnika HTTP POST request na server kroz *„body“* šalje podatke: userId i followedUserId (Slika 4.29).

```
const addToFollowedUsers = async () => {
  await followedRouteService.post({
    userId: user._id,
    followedUserId: post.userId._id,
  });

  await fetchFollowedUsersPosts();
};
```

Slika 4.29. *POST request (praćenje korisnika).*

Server kroz *„body“* dohvaća podatke te pomoću userID-a pronalazi korisnika i *„push-a“* novi followedUserID u polje followedUsers (Slika 4.30.).

```
export const addFollowedUser = async (req, res) => {
  const { userId, followedUserId } = req.body;

  try {
    await User.findOneAndUpdate(
      { _id: userId },
      { $push: { followedUsers: followedUserId } },
      { new: true }
    );
    res.status(201).json();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.30. *Praćenje korisnika.*

Prilikom prestanka praćenja korisnika HTTP DELETE request na server kroz *„params“* šalje podatke: userId i followedUserId (Slika 4.31.).

```
const deleteFollowedUser = async () => {
  await followedRouteService.deleteFollowedUser({
    userId: user._id,
    followedUserId: post.userId._id,
  });

  await fetchFollowedUsersPosts();
};
```

Slika 4.31. *DELETE request (brisanje praćenog korisnika).*

Server kroz „body“ dohvaća podatke te pomoću userID-a pronalazi korisnika i *„pulla“* followedUserID iz polja followedUsers (Slika 4.32.).

```
export const removeFollowedUser = async (req, res) => {
  const { userId, followedUserId } = req.params;

  try {
    await User.findOneAndUpdate(
      { _id: userId },
      { $pull: { followedUsers: followedUserId } },
      { new: true }
    );
    res.status(201).json();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.32. *Brisanje praćenog korisnika.*

4.7. Dohvaćanje objava

Prilikom dohvaćanja objava iz baze podataka, ne preporuča se dohvaćanje svih objava odjednom, jer to utječe na performanse aplikacije. Zbog toga je, u ovoj aplikaciji, uvedena paginacija. Pomoću paginacije objave su raspodijeljene na stranice (3 objave po stranici).

GET request putem query-a serveru šalje podatke: pageSize i PageNumber (Slika 4.33.).

```
const fetchPosts = async () => {
  const response = await postRouteService.get({
    pageSize: 3,
    page: pageNumber,
  });

  setPosts(response.data.posts);
  setNumOfPages(response.data.numOfPages);
};
```

Slika 4.33. *GET request (dohvaćanje objava).*

Server iz querya dohvaća poslane podatke te sortira objave po datumu, postavlja broj objava na 3 objave te ovisno o stranici kalkulira koje točno objave je potrebno dohvatiti iz baze.

Populate() je Mongoose metoda koja služi za povezivanje dokumenata preko kolekcija. Objave su povezane sa korisnikom i komentarima, komentari su povezani sa korisnikom i odgovorima, a odgovori su povezani samo sa korisnikom (Slika 4.34.).

```
export const getPosts = async (req, res) => {
  const { page, pageSize } = req.query;

  try {
    const posts = await Post.find({})
      .sort({ timeStamp: "desc" })
      .limit(pageSize)
      .skip((page - 1) * pageSize)
      .populate("userId")
      .populate("comments")
      .populate({ path: "comments", populate: { path: "userId" } })
      .populate({ path: "comments", populate: { path: "replies" } })
      .populate({
        path: "comments",
        populate: { path: "replies", populate: { path: "userId" } },
      });
    const numberOfPosts = await Post.countDocuments();
    const numOfPages = Math.ceil(numberOfPosts / pageSize);

    res.status(200).json({ posts, numOfPages });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Slika 4.34. *Dohvaćanje objava*

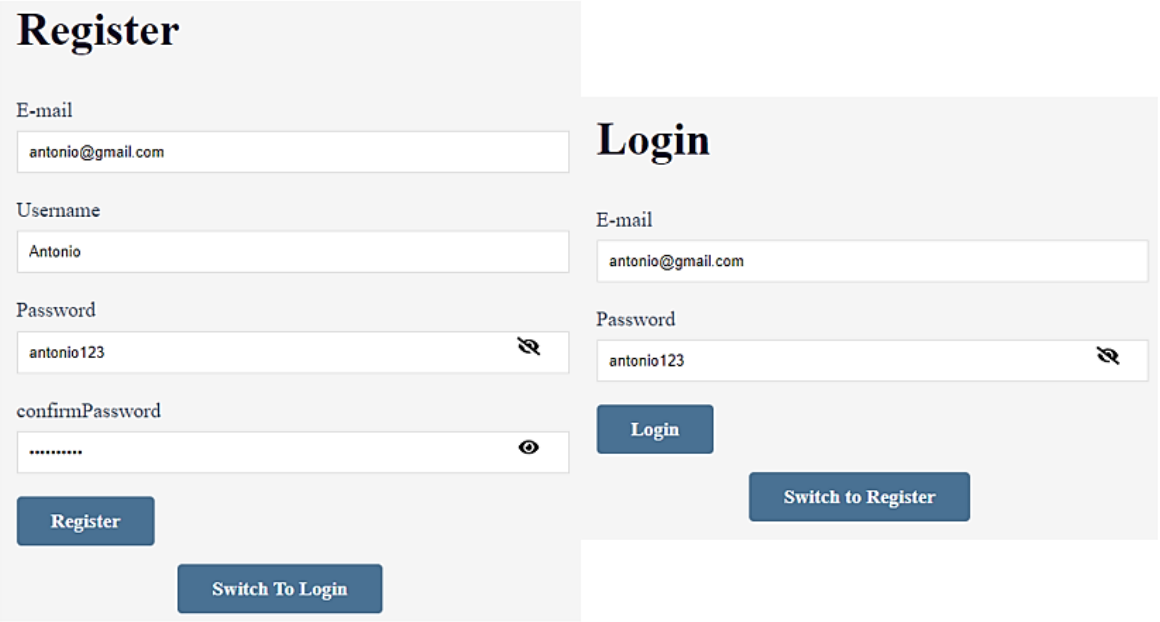
Funkciju fetchPosts je potrebno pozvati na iscrtavanju (eng. *render*) stranice, ali i prilikom brisanja, uređivanja i dodavanja novih postova, komentara i reply-ova kako se stranica ne bi morala svaki puta osvježiti (eng. *refresh*) (Slika 4.36.).

```
useEffect(() => {
  fetchPosts();
  const userData = JSON.parse(window.sessionStorage.getItem("User"));
  setUser(userData);
}, [pageNumber]);
```

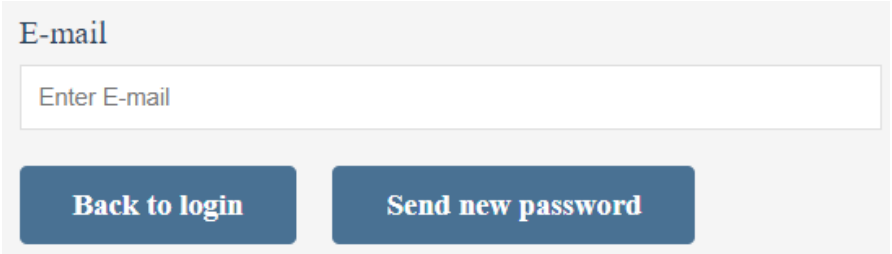
Slika 4.36. *useEffect (dohvaćanje objava prilikom iscrtavanja).*

5. KORIŠTENJE APLIKACIJE

Za pristup aplikaciji, korisnik se treba registrirati i prijaviti (Slika 5.1.). Ukoliko je korisnik zaboravio lozinku, postoji mogućnost slanja nove, privremene, lozinke na njegov email (Slika 5.2.).



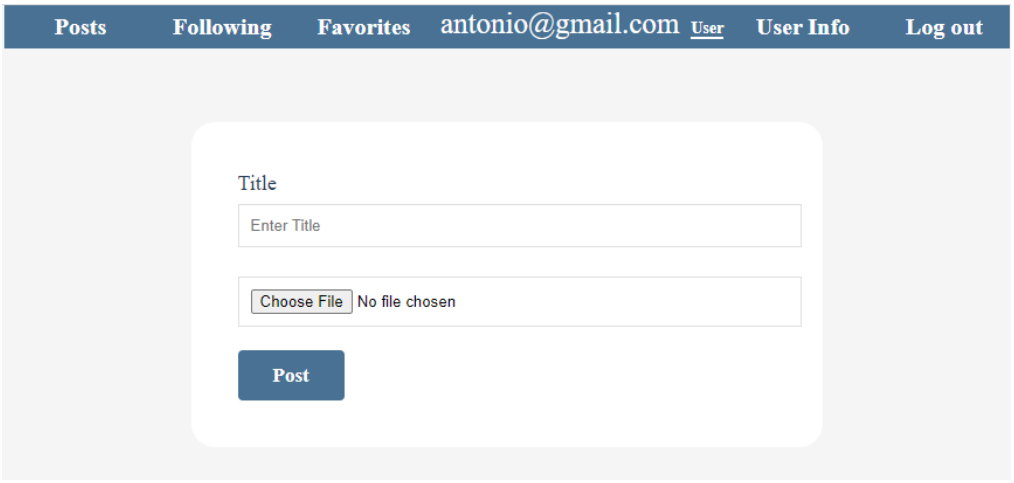
Slika 5.1. *Registracija i prijava korisnika.*



Slika 5.2. *Kreiranje privremene lozinke.*

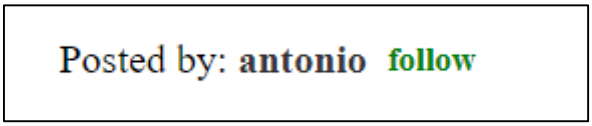
Nakon uspješne prijave, korisnik pristupa web aplikaciji te mu se na početnom zaslonu prikazuju objave svih korisnika. Korisnik ima mogućnost kreiranja objava postavljanjem slika i naslova slike, a objave može i obrisati (Slika 5.3.). Ostali prijavljeni

korisnici mogu reagirati na objavu komentarima te se na iste komentare može odgovoriti novima (Slika 5.4.). Aplikacija svim korisnicima omogućava kreiranje višestrukih favorita među slikama te se korisnici međusobno mogu pratiti (eng. *follow*) (Slika 5.5.).



Slika 5.3. *Kreiranje objave.*

Slika 5.4. *Prikaz komentara i odgovora*

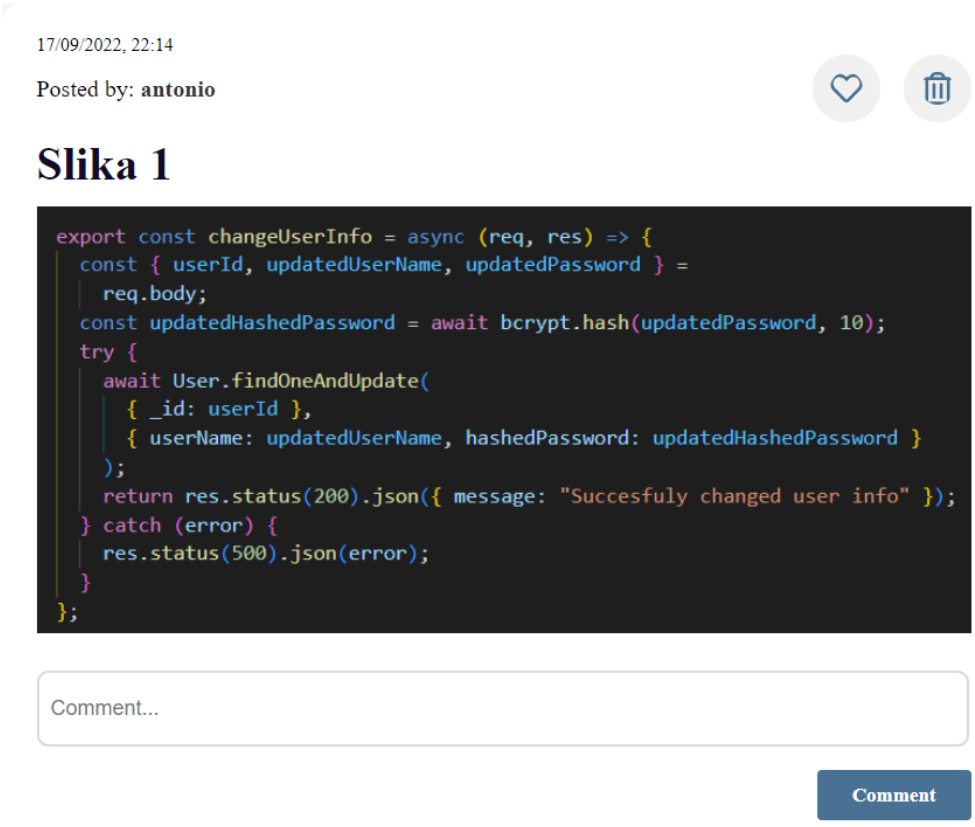


Slika 5.5. *Prikaz tipke za praćenje korisnika*

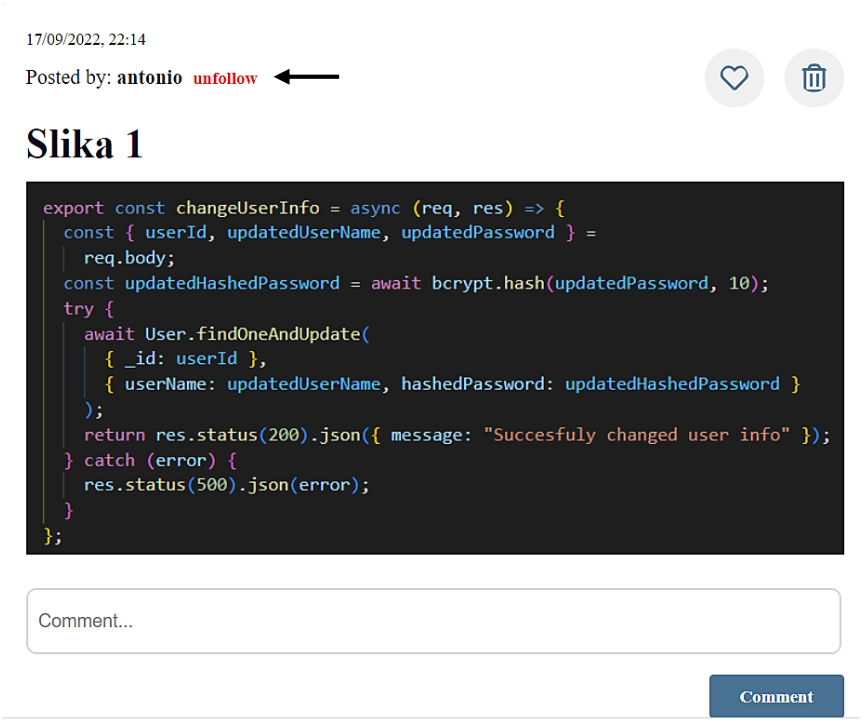
Korisnik preko navigacijske trake može putem poveznica (eng. *link*) pristupiti stranicama:

- Posts – na kojoj se mogu kreirati nove objave te su izlistane objave svih korisnika (Slika 5.6.)
- Following – stranica na kojoj su izlistane objave praćenih korisnika (Slika 5.7.)
- Favorites – stranica na kojoj su izlistane favorit objave (Slika 5.8.)
- userInfo – korisnik može mijenjati informacije (ime i šifra) (Slika 5.9.).

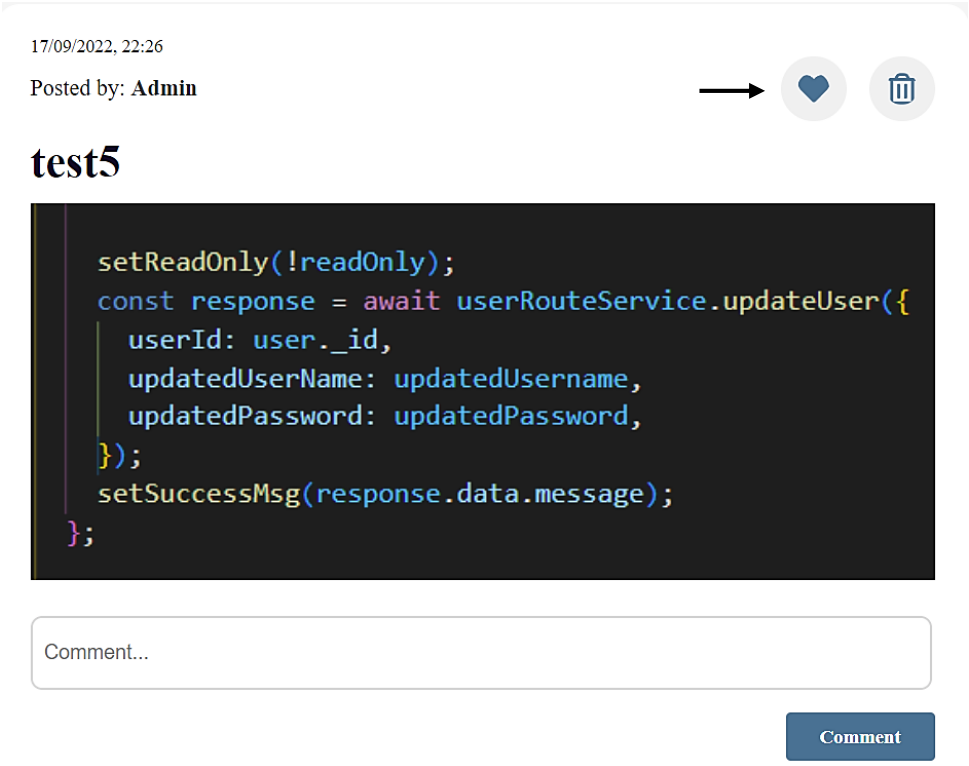
Aplikacija također ima korisnika Administratora koji ima mogućnost brisanje, editiranje svih objava, komentara i odgovora



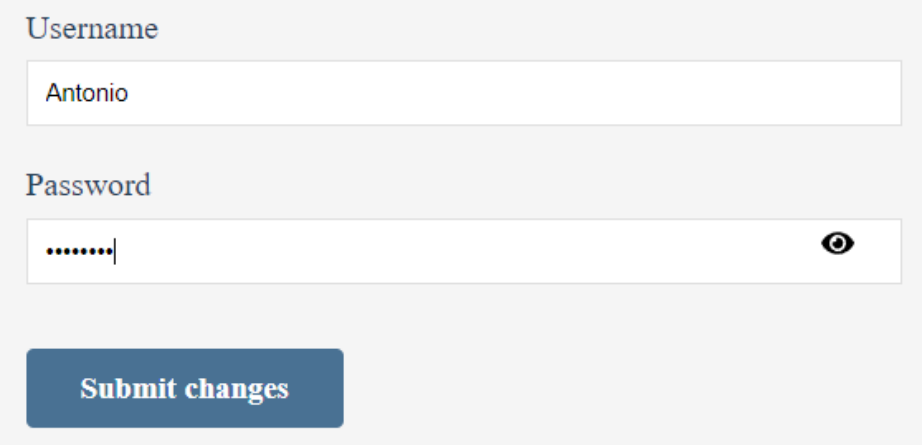
Slika 5.6. *Primjer objave u aplikaciji.*



Slika 5.7. *Prikaz praćenog korisnika.*



Slika 5.8. *Prikaz favorit objave.*



Slika 5.9. *Izmjena imena i lozinke korisnika.*

6. ZAKLJUČAK

U sklopu ovog završnog rada, napravljena je web aplikacija društvene mreže koja omogućava svojim korisnicima dijeljenje i komentiranje slika. Za pristup aplikaciji korisnik se mora registrirati i prijaviti. Prilikom registracije korisnika, u bazi podataka lozinka se šifrira kako bi se spriječila zloupotreba korisničkih informacija.

Nakon uspješne prijave korisnik ima mogućnost kreiranja objava (slika) koje dijeli s ostalim korisnicima na početnom zaslonu. Korisnik može i komentirati, pratit i spremati objave među favorite. Također, postoji opcija izmijene korisničkog imena i lozinke. Web aplikacija društvene mreže korisnicima može poslužiti za dijeljenje i pronalaženje ideja.

Budućim unaprjeđenjem aplikacije poželjno je poboljšati prikaz komentara. U slučaju velikog broja komentara ili odgovora dodala bi se značajka „prikaži više“ koja bi prikazala sve prvobitno skrivene komentare i odgovore.

LITERATURA

[1] https://geek.hr/pojmovnik/sto-je-web-aplikacija/ (pristup 08.07.2020.)

[2] https://help.pinterest.com/en/guide/all-about-pinterest (pristup 17.09.2022.)

[3] https://www.digitaltrends.com/computing/what-is-reddit/ (pristup 18.09.2022.)

[4] https://www.educba.com/html5-vs-html4/ (pristup 08.07.2020.)

[5] Bugarić, M., Rezo, F. 2019. PROGRAMIRANJE ZA Internet, Upute za laboratorijske vježbe, Osnove CSS-a

[6] https://tesla.carnet.hr/mod/book/view.php?id=5397&chapterid=828 (pristup 08.07.2020.)

[7] https://www.mojwebdizajn.net/skriptni-jezici/vodic/css/uvod-u-css.aspx (pristup 08.07.2020.)

[8] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (pristup 09.07.2020.)

[9] https://www.w3schools.com/js/js_whereto.asp (pristup 09.07.2020.)

[10] Priyesh Patel, „What exactly is Node.js?“. 2018. URL: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/ (pristup 10.09.2022.)

[11] https://www.webprogramiranje.org/uvod-u-node-js/ (pristup 10.09.2022.)

[12] W3Schools, „What is npm?“, (bez dat.). URL: https://www.w3schools.com/whatis/whatis_npm.asp (pristup 10.09.2022.)

[13] Vipul A. M., Prathamesh Sonpatki „ReactJS by Example - Building Modern Web Applications with React“. Birmingham, Publishing Place. 2016.

[14] https://www.freecodecamp.org/news/react-hooks-fundamentals/ (pristup10.09.2022.)

[15] https://www.reactenlightenment.com/react-jsx/5.1.html (pristup10.09.2022.)

[16] https://reactjs.org/docs/hooks-reference.html (pristup10.09.2022.)

[17] Stack Overflow (2015.), *What are the benefits of Express over plain Node.JS?*, URL: https://stackoverflow.com/questions/28823253/what-are-the-benefits-of-express-over-plain-node-js (pristup10.09.2022.)

[18] Guru99.com (bez dat.), *SQL vs NoSQL: What's the difference?*, URL: https://www.guru99.com/sql-vs-nosql.html (pristup 10.09.2022.)

[19] https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/ (pristup 12.09.2022.)

[20] https://axios-http.com/docs/intro (pristup 12.09.2022.)

SAŽETAK

Naslov: Web aplikacija društvene mreže dijeljenja slika

Cilj ovog rada je izrada web aplikacije društvene mreže koja omogućava svojim korisnicima dijeljenje i komentiranje slika Na početku su opisane back-end i front-end tehnologije koje su korištene prilikom razvoja aplikacije. Front-end i back-end kod pisani su JavaScript programskim jezikom uz pomoć okvira React i Express.js. Svi podaci se spremaju u MongoDB (NoSQL bazu podataka). Kako bi korisnik pristupio aplikaciji mora se registrirati i prijavit. Nakon prijave korisniku se izlistaju objave svih korisnika te sam ima mogućnost kreiranja novih. Korisnik, također, ima mogućnost komentirati objave i odgovarati na već postojeće komentare, pratiti druge korisnike i spremati objave u favorite.

Ključne riječi: baza podataka, društvene mreže, Express.js, React .js, web aplikacija

ABSTRACT

Title: Image sharing social network web application

The goal of this paper is the creation of a web application of a social network that enables its users to share and comment on images. At the beginning, the backend and frontend technologies that were used during the development of the application were described. The frontend and backend code are written in the JavaScript programming language with the help of the React and Express.js frameworks. All data is stored in MongoDB (NoSQL database). To access the application, the user must register and log in. After logging in, the user is shown the posts of all users and has the option of creating new ones. The user also can comment on posts and reply to existing comments, follow other users, and save favourite posts.

Key words: data base, Express.js, React.js, social network, web application