Izrada Web aplikacije za pisanje online dnevnika

Lončar, Max

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

IZRADA WEB APLIKACIJE ZA PISANJE ONLINE DNEVNIKA

Diplomski rad

Max Lončar

Osijek, 2022.

SADRŽAJ

1. UVOD

Tema ovog diplomskog rada je izrada web aplikacije za pisanje *online* dnevnika. Za izradu web aplikacije se, kao *frontend* alat, koristi JavaScript biblioteka React. Web aplikacija zapravo predstavlja sustav čiji je glavni zadatak korisnicima omogućiti objavljivanje i komentiranje teksta, odnosno bloga, s ciljem povezivanja i razmjene informacija među korisnicima. Cijeli blog sustav sadrži autentifikaciju i autorizaciju. Autentifikacijom se provjera identitet korisnika, odnosno je li korisnik registriran, prijavljen, odnosno postoji li u bazi podataka. S druge strane, autorizacijom se provjera koji specifični blogovi pripadaju pojedinom korisniku jer korisnik može samo vlastite blogove i komentare uređivati i brisati. Korisnici također mogu ostaviti pojedine komentare na blogove. No osim toga, blog sustav sadrži i administracijski dio. U administracijskom dijelu, administrator upravlja korisnicima i sadržajem samog sustava.

Tijekom izrade same web aplikacije, do izražaja dolaze znanja stečena na fakultetu iz raznih kolegija kao što su „Objektno orijentirano programiranje“, „Algoritmi i strukture podataka“, „Razvoj programske podrške objektno orijentiranim načelima“, „Baze podataka“, „Osnove razvoja web i mobilnih aplikacija“. Ponajviše do izražaja dolaze znanja stečena iz kolegija „Napredno web programiranje“, gdje su se obradili MongoDB, biblioteka Mongoose te Node.js-ov *backend* aplikacijski okvir Express.js. Također su primijenjena i znanja stečena tijekom odrađivanja stručne prakse iz područja računarstva. Sama web aplikacija je namijenjena svim korisnicima koji žele podijeliti svoje proživljene trenutke i događaje, svoja razmišljanja, određena mišljenja ili znanja. Kako bi se web aplikacija što kvalitetnije izvela, korištena je skupina tehnologija MERN (engl. *MERN stack*). MERN stog predstavlja pristup u kojem se koriste već spomenute tehnologije MongoDB (baza podataka), Express.js (Node.js web okvir), React.js (okvir na strani klijenta) te Node.js (JavaScript web poslužitelj). Za testiranje API poziva (engl. *Application Programming Interface calls*), koristio se Postman – API platforma za izradu i korištenje API-ja. Za opis strukture web aplikacije koristi se JSX (engl. *JavaScript Syntax Extension*), dok se za opis prezentacije web aplikacije i njezinih komponenti koristi SASS (engl. *Syntactically Awesome Style Sheets*) varijanta SCSS te CSS (engl. *Cascading Style Sheets*). Sav programski kod je pisan u Visual Studio Code uređivaču koda.

Struktura diplomskog rada započinje uvodom gdje se ukratko opisuje problematika koja se rješava. Nakon uvoda na red dolazi teorijska podloga gdje se opisuju i uspoređuju korištene tehnologije koje su bile potrebne za uspješnu realizaciju web aplikacije. Nakon opisa svih

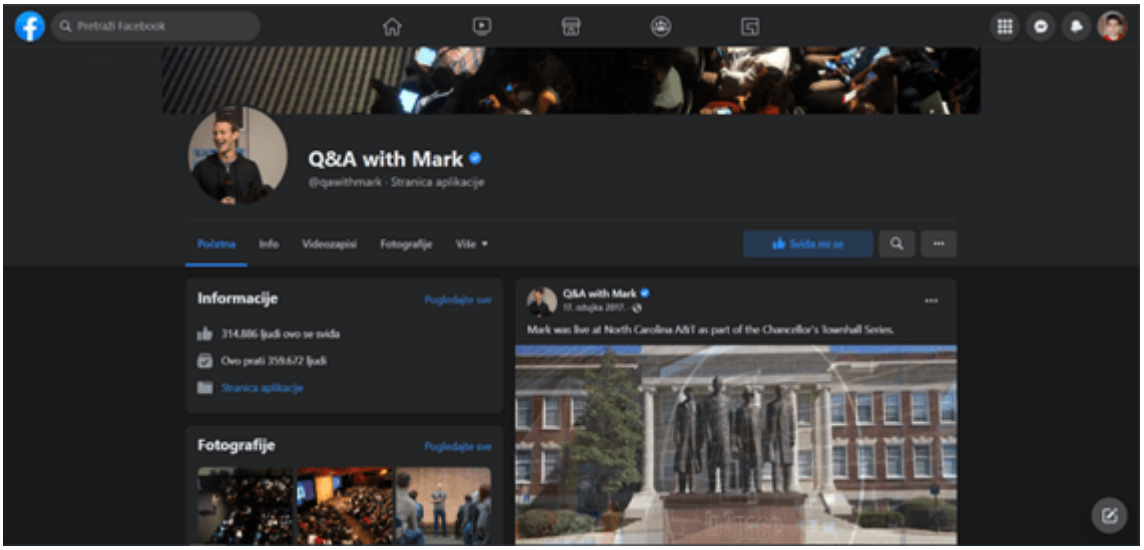
korištenih tehnologija, slijedi opis postupka izrade same web aplikacije. Postupak obuhvaća svaki dio web aplikacije uključujući *frontend, backend,* bazu podataka i API. Prije samog zaključka, slijedi prikaz rezultata, odnosno konačnog izgleda web aplikacije i njezinih komponenti.

2. PREGLED PODRUČJA I KORIŠTENE TEHNOLOGIJE

2.1. Aktualne React web aplikacije

Postoji mnogo web aplikacija kreiranih pomoću React-a. Neke od najpoznatijih su svakako Facebook, Instagram, Netflix, New York Times, Yahoo! Mail, WhatsApp, Dropbox i mnoge druge. Od svih navedenih aplikacija, najsličnija je upravo Facebook. Iako je daleko naprednija aplikacija, povezuju je poneke sličnosti. Slična je iz razloga što ima pretežito sličnu svrhu kao što je mogućnost objave i komentiranja raznih objava (u ovom slučaja bloga), odnosno omogućava nekakav stupanj komunikacije između korisnika. Osim toga, postoje i neke slične funkcionalnosti kao što su mogućnost prijave i registracije korisnika, uređivanje i brisanje vlastitih objava, odnosno blogova i mnoge druge.

Iako samo djelomično, Facebook zapravo koristi ReactJS. Njihova web stranica je zapravo izrađena s React-om kao skriptom koja je uklopljena u čitav kod same aplikacije. Njihova mobilna aplikacija je također izrađena s verzijom React-a pod nazivom React Native. React Native je dosta sličan React-u, no on je više odgovoran za prikaz komponenti iOS-a i Androida umjesto samih DOM (engl. *Document Object Model*) elemenata. Primjer izgleda Facebook stranice se nalazi na slici 2.1. [1]



Slika 2.1. *Primjer izgleda Facebook aplikacije*

Iako je manje sličnija aplikacija, Instagram je jedna od aplikacija čija je upotreba ReactJS-a ogromna. To se može vidjeti po raznim dijelovima, odnosno komponentama koji čine cjelinu aplikacije. Osim toga, tu su brojne značajke kao što su tzv. *geolokacije* te točnost pretraživanja tražilice. Sve je to zapravo u API-ju same aplikacije. Stoga se može reći da se Instagram u potpunosti temelji na ReactJS biblioteci. [1]

Osim ovih dviju popularnih društvenih mreža koje su kreirane pomoću njega, React se koristio i u svrhe *streaming* usluge pri izradi Netflix-ove platforme Gibbon koja se koristi za TV uređaje niskih performansi umjesto DOM-a koji se koristi u web preglednicima. „Netflix je čak objavio službenu objavu na blogu kojom objašnjavaju kako ReactJS biblioteka pomaže u brzini pokretanja, performansama u vremenu izvođenja, modularnosti i raznim drugim prednostima.“ [1]

Od poznatih portala vijesti, New York Times je jedan od onih koji su koristili React. Također i Yahoo-ov *mail* klijent koristi React. Koristili su React kako bi izgradili čvrstu i unificiranu arhitekturu. Trebali su obaviti puno nadogradnji te su se odlučili za React zbog mnogih razloga kao što su jednosmjeran protok podataka i renderiranje na strani klijenta i poslužitelja. [1]

WhatsApp je također jedna od aplikacija koja koristi ReactJS. Koristili su React za izgradnju korisničkih sučelja s Facebook-a. Od nedavno potpuno nova web aplikacija WhatsAapp koristi React poput već spomenutog Facebook-a. [1]

Dropbox se također prebacio na ReactJS i to u vrijeme kada je React postao vrlo popularan među programerima. Dropbox učinkovito koristi brojne resurse React okvira, što uvelike pridonosi njegovom uspjehu pohranjivanja u oblaku i ostvarivanja rješenja za sigurnosno kopiranje na mreži. [1]

Svaka od navedenih aplikacija ima poneke sličnosti s navedenom temom, neke više, a neke manje. Svaka od njih je izrađena od različitih komponenti, koje kada se zajedno grupiraju, čine jednu cjelinu. Kada se promjeni stanje nekih komponenti, React automatski ažurira DOM. Ono što je zajedničko svim ovim aplikacijama je to da su zbog smanjenog rizika, poboljšane učinkovitosti i brojnih drugih organizacijskih prednosti, odabrali ReactJS i iskoristile njegove prednosti koje nudi. [1]

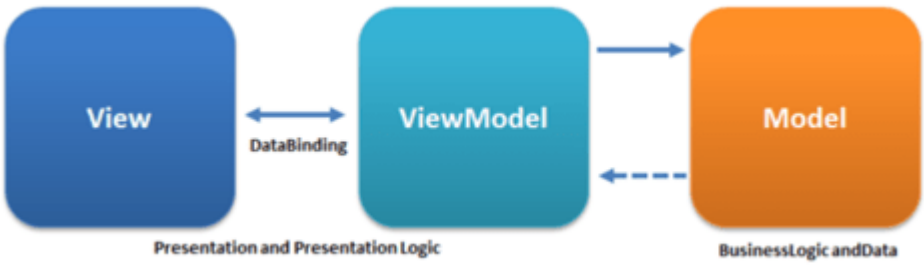
2.2. Usporedba s postojećim alatima

Za izradu diplomskog rada korištene su brojne tehnologije i alati. Za izradu web aplikacije je korištena skupina tehnologija s nazivom MERN stog. U MERN skupinu tehnologija spadaju MongoDB, Express, React i Node. Za izgradnju i testiranje API poziva, korištena je API platforma s nazivom Postman, dok se sav kod pisao unutar Visual Studio Code uređivača koda.

Za svaku korištenu tehnologiju i alate, može se pronaći par alternativa. Budući da se svaki dan generiraju ogromne količine nestrukturiranih podataka, raste i potreba za NoSQL bazama podataka. MongoDB grupira sve podatke u dokumente i zbirke i stoga se može koristiti za pohranu različitih vrsta podataka. Iako nudi visoke performanse, potrošnja podataka u MongoDB-u je velika zbog denormalizacije. No ipak postoji par dobrih alternativa za MongoDB kao što su Redis, Apache Cassandra, DynamoDB i mnogi drugi. Redis predstavlja bazu podataka otvorenog koda (engl. *open-source*) koja koristi aktivno-aktivno (engl. *Active-Active*) i aktivno-pasivne (engl. *Active-Passive*) distribuirane arhitekture. Sadrži brz sustav koji omogućuje upravljanje bazom podataka te podržava različite strukture podataka kao što su *hashovi*, *bitmape*, popisi, nizovi i druge strukture. Apache Cassandra je također jedna od boljih alternativa jer nudi visoku dostupnost i skalabilnost bez utjecaja na performanse. Podržava replikaciju u mnogim podatkovnim centrima i zonama unutar kojih je dostupna usluga oblaka (engl. *cloud service*). Replikacijom podataka na više čvorova se osigurava tolerantnost sustava na pogreške. DynamoDB se ističe time što grupira podatke u parove ključ-vrijednost i time što je potpuno upravljana baza podataka s više regija. To znači da nudi predmemoriranje (engl. *caching*) u memoriji, sigurnost, ugrađenu sigurnosnu kopiju (engl. *backup*) te mogućnost vraćanja podataka. [2]

ExpressJS je godinama standard i najpoznatija biblioteka Node.js sustava. Node je s Express *okvirom* pružio mogućnost da se po prvi puta koristi JavaScript na strani poslužitelja. Iako je na neki način započeo novu eru razvoja JavaScripta, neke druge JavaScript biblioteke se počinju izdvajati zbog niza značajki koje donose. Neke od tih biblioteka su Koa, Fastify, VueJS i mnoge druge. Koa je primjerice mnogo prilagodljivija prema postojećim NodeJS *okvirima*. Omogućuje odbacivanje povratnih poziva – što je jedna od glavnih kritika ExpresJS-a. Za razliku od ExpressJS-a, Koa dolazi bez ikakvog paketa međuprograma te zahtijeva module za usmjeravanje (engl. *routing*) i predloške. Fastify je poznat po svojoj brzini. Brži je od ExpressJS-a u gotovo svakom zahtjevu te omogućuje potpunu enkapsulaciju za razne dodatke. Osim toga, *parsira* (engl. *parses*) dolazne zahtjeve u JSON s bržim renderiranjem i omogućuje brzo usmjeravanje. VueJS je zapravo jedna od novijih alternativa ExpressJS-u, koja se temelji na MVVM (engl. *Model-View-*

ViewModel) arhitekturi. Nudi mnogo više osnovnih funkcionalnosti i poznatu sintaksu za izradu predložaka – što čini integraciju i migraciju projekata lakšom i bržom. MVVM arhitektura olakšava odvajanje *pogleda* (engl. *view*), odnosno korisničkog sučelja, od razvoja poslovne logike, odnosno modela. Slikoviti primjer MVVM arhitekture se nalazi na slici 2.2. [3]



Slika 2.2. *MVVM arhitektura [4]*

Jedan od najvećih izazova pri odabiru alternativa React-u je zapravo pronalaženje *okvira* koji nudi fleksibilnost i lakoću kao što to React čini. Zato je odabir alternative relativno težak, gdje se nekoliko stvari mora uzeti u obzir kao što su stabilnost, API pozivi, performanse i veličine paketa. Stoga se od alternativa najviše ističu Angular i VueJS. AngularJS se nekako smatra najboljom ReactJS alternativom za izradu dinamičnih web i mobilnih aplikacija. To je kompletan JavaScript *okvir* otvorenog koda koji pomaže u pojednostavljenju i strukturiranju koda. Korišten je od strane velikih tvrtki kao što su Google, Amazon i Udemy. Vrlo je popularan jer na neki način osnažuje programere s vlastitim paketom naprednih AngularJS *okvira* i alata. Postoje mnoge razlike između React-a i Angular-a [5]:

- ReactJS se više koristi za stvaranje izvrsnih UI (engl. *user interface*) komponenata, dok se AngularJS koristi za razvoj kompletnih dinamičkih web aplikacija,
- ReactJS se temelji na JavaScript programskom jeziku, dok se AngularJS temelji na Typescript programskom jeziku,
- ReactJS se temelji na virtualnom DOM-u, dok se AngularJS temelji na MVC (engl. *Model-View-Controller*) arhitekturi,
- ReactJS ima mogućnost dodavanja JavaScript biblioteka u svoj izvorni kod, dok AngularJS nema tu mogućnost,
- za testiranje i otklanjanje pogrešaka su u React-u potrebni dodaci (engl. *add-ons*), dok AngularJS to obavlja prema zadanim postavkama.

S druge strane, VueJS je također jedan od najpopularnijih frontend okvira. Otvorenog je koda i koristan je za razvoj jednostranih aplikacija. Za razliku od Angular-a, VueJS dobro rješava problem složenosti s kojim se programeri često susreću. Manje je veličine i nudi dvije glavne prednosti – vizualni DOM (engl. visual DOM) i komponentni DOM (engl. component DOM). Neke od razlika između React-a i VueJS-a su [5]:

- unutar React-a se ne može ponovno koristiti kod (osim za CSS), dok se kod VueJS-a mogu ponovno koristiti i HTML i CSS kod,
- VueJS je bolji za renderiranje i optimizaciju jer uključuje gotove funkcije renderiranja,
- ReactJS koristi JSX za dizajn JavaScript koda, dok VueJS podržava JSX kao i HTML predloške i dodatke.

Što se tiče alternativa za Node.js, ističu se ASP.NET i Ruby. ASP.NET je *okvir* web aplikacija otvorenog koda, na strani poslužitelja, koji nudi razvoj složenih API-ja u stvarnom vremenu, stvaranje dinamičkih web stranica te aplikacija temeljenih na webu. Na najbolji način koristi HTML, CSS i JavaScript za samu izradu web aplikacija. S druge strane, Ruby pokriva mnoge druge primjene kao što su izrada prototipova i analiza podataka. Njegova popularna implementacija je zapravo Ruby on Rails – razvojni *okvir* koji nudi veću brzinu te podržava .NET i Java virtualni stroj (engl. *Java Virtual Machine – JVM*). Koristi se također i kao skriptni jezik u svim vrstama razvoja aplikacija – *frontend, backend* te cjelokupni web razvoj. [6]

Postman je korišten za testiranje i praćenje API poziva. Kao nekakve alternative se ističu Testfully i Insomnia. S alatom Testfully, mogu se slati zahtjevi i lokalnim i implementiranim API-jima te provjeriti njihov odgovor. Mogu se pohraniti korisnički zahtjevi u oblaku te se mogu lagano podijeliti s drugim korisnicima. Također je omogućena i jednostavna ponovna upotreba testova te provjera ispravnosti lokalnih i postavljenih API-ja u različitim okruženjima. Insomnia je, poput Postman-a, započela kao HTTP klijent te evoluirala u alat za razvoj API-ja. Ono što Insomnia trenutno nudi su usluge HTTP klijenta, automatizirano API testiranje te Open API uređivač. Također može slati zahtjeve lokalnim i implementiranim API-jima uz mogućnost definiranja više okruženja čije se varijable mogu ugraditi u zahtjeve. [7]

U slučaju uređivača koda, najpopularniji izbor je definitivno Visual Studio Code. Od alternativa, najviše do izražaja dolaze Notepad++ te Sublime Text. Notepad++ je besplatan uređivač izvornog koda i zamjena za Notepad koja podržava nekoliko različitih jezika. Lagan je za korištenje, brz i jednostavan. No u usporedbi s Visual Studio Code-om, nema proširivo razvojno okruženje (engl. *integrated development environment – IDE*). Iako je odličan alat, nedostaje mu

mnogo značajki koje sadrži sami Visual Studio Code. Primjer komponente *BlogPost* otvorene unutar uređivača Notepad++ se nalazi na slici 2.3. [8]

Mnogi pak smatraju Sublime Text kao bolju alternativu. Sublime Text je uređivač teksta za kod, HTML te bilo koju vrstu tekstualne datoteke. Sadrži elegantno korisničko sučelje i odlične značajke. Troši manje resursa, može se proširiti raznim dodacima i proširenjima te je lako prenosiv. [8]

BlogPost.jsx

```
import { Link } from "react-router-dom";

export default function BlogPost({ blogPost }) {
  const publicFolder = "http://localhost:5000/images/";

  return (
    <div className="blog-post">
      {blogPost.image ? (
        <img
          className="blog-post__image"
          src={publicFolder + blogPost.image}
          alt="Blog post"
        />
      ) : (
        <img
          className="blog-post__image"
          src="https://media.istockphoto.com/photos/the-word-blog-arranged-
          alt="Blog post"
        />
      )}
      <div className="blog-post__info">
        <div className="blog-post__categories">
          <p className="blog-post__category">{blogPost.category}</p>
        </div>
        <p className="blog-post__date">
          {new Date(blogPost.createdAt).toDateString()}
        </p>
        <hr />
        <h2 className="blog-post__title">{blogPost.title}</h2>
        <p className="blog-post__description">{blogPost.description}</p>
        <Link to={`/post/${blogPost._id}`}>
          <button className="blog-post__button">Read more</button>
        </Link>
      </div>
    </div>
  );
}
```

Slika 2.3. *Primjer komponente BlogPost unutar Notepad++ uređivača koda*

2.3. MERN tehnologije

MERN predstavlja jednu od nekoliko varijanti MEAN stoga (MongoDB, Express, Angular, Node), gdje je Angular.js zapravo zamijenjen s React.js. Od ostalih varijanti, vrijedi istaknuti MEVN (MongoDB, Express, Vue, Node) pristup i bilo koji *frontend* JavaScript okvir. Bez obzira koja god *frontend* tehnologija da se odabere (Angular, React ili Vue), one su uvijek idealan pristup u suradnji s JavaScript-om i JSON-om. [9]

MERN stog skupina tehnologija se sastoji od četiri ključne tehnologije: MongoDB, Express, React i Node. Express i Node predstavljaju aplikacijski sloj. U tom odnosu, Express.js je zapravo web okvir (engl. *web framework*) na strani poslužitelja (engl. *server*), a Node.js je JavaScript poslužiteljska platforma (engl. *server platform*). Cijela MERN arhitektura zapravo omogućuje izgradnju web aplikacije s troslojnom arhitekturom – *frontend*, *backend* i baza podataka. Takva troslojna arhitektura uključuje razinu prikaza za *frontend* (React.js), razinu aplikacije (Express.js i Node.js) te razinu baze podataka (MongoDB). Primjer takve arhitekture se nalazi na slici 2.4. [9]

Kombinacijom svih razina, JSON podaci prirodno teku od naprijed prema natrag. Time se povećava brzina nadogradnje i jednostavnost otklanjanja pogrešaka. Također se mora znati samo jedan programski jezik i struktura JSON dokumenta da bi se razumio cijeli sustav. Zato je MERN postao učestao izbor za današnje web programere koji žele brzo napredovati i steći nekakvo iskustvo. [9]

Slika 2.4. *MERN arhitektura* [9]

2.3.1. MongoDB

MongoDB predstavlja jedan od tri sloja MERN arhitekture. Ono je zapravo program koji se bavi upravljanjem NoSQL (*Not only SQL*) bazama podataka. Koristi se kao alternativa tradicionalnim relacijskim bazama podataka i vrlo je korisna za rad s velikim skupovima raspodijeljenih podataka (engl. *Big data*). Osim toga, koristi se i kao alat za upravljanje informacijama, pohranjivanje i dohvaćanje samih informacija. [10]

MongoDB podržava različite oblike podataka. Umjesto tablica i redaka koji se koriste u relacijskim bazama podataka, MongoDB koristi arhitekturu koja se sastoji od zbirki i dokumenata. Sadrži mnoge značajke kao što su *ad-hoc* upiti, indeksiranje, balansiranje opterećenja, izvršavanje JavaScript jezika na strani poslužitelja i druge. *Ad-hoc* upiti vraćaju određena polja dokumenata i mogu uključivati korisnički definirane JavaScript funkcije. Indeksiranje se vrši primarnim i sekundarnim indeksima. Balansiranje opterećenja se vrši na način da MongoDB može pokrenuti više poslužitelja i tako održava sustav u radu i pokreće ga u slučaju kvara hardvera (engl. *hardware*). Izvršavanje JavaScript jezika na strani poslužitelja se vrši na način da se JavaScript koristi u upitima i šalje se izravno u bazu podataka na izvršavanje. [10, 11]

Što se tiče načina rada, MongoDB koristi zapise koji se sastoje od dokumenata koji sadrže strukturu podataka sastavljenu od parova polja i vrijednosti. Dokumenti zapravo predstavljaju osnovnu jedinicu podataka i slični su JavaScript objektnoj notaciji (engl. *JavaScript Object Notation*), odnosno JSON-u. Koriste varijantu JSON oblika zvanu Binarni JSON (engl. *Binary JSON*), odnosno BSON. Ono što je prednost BSON-a u odnosu na JSON je to što prihvaća više tipova podataka. Polja u dokumentima su slična stupcima u relacijskoj bazi podataka, a svaki dokument sadrži primarni ključ kao jedinstveni identifikator. [10, 11]

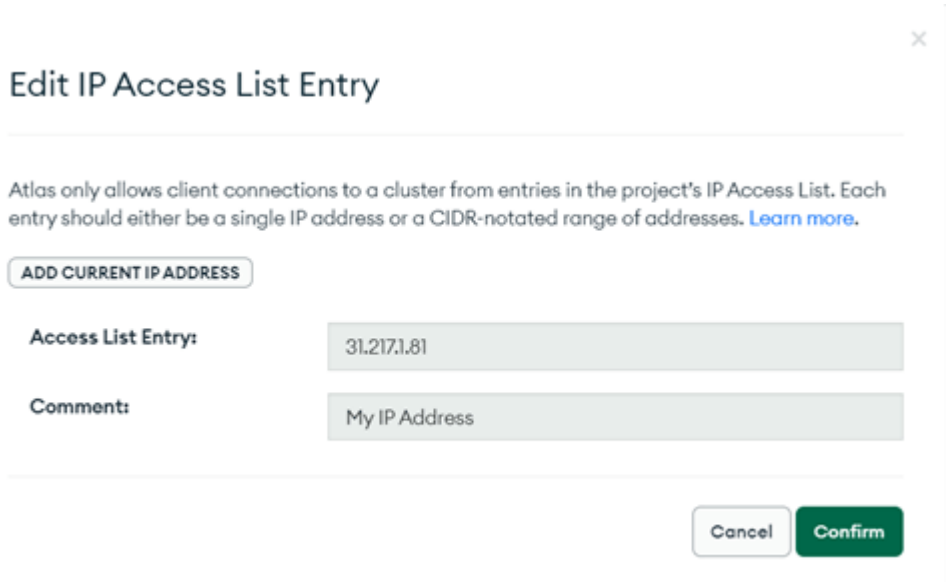
Zbirke predstavljaju skupove dokumenata i one su ekvivalentne tablicama u relacijskim bazama podataka. Također mogu sadržavati bilo koju vrstu podataka, no jedino što se ne mogu širiti po različitim bazama podataka. [10, 11]

Vidljivo je da MongoDB ima mnogo prednosti, no ima i nekoliko nedostataka. Od prednosti se može izdvojiti to da ne zahtijeva unaprijed definirane sheme, pohranjuje bilo koju vrstu podataka te sadrži fleksibilnost kreiranja bilo kojeg broja polja u dokumentu što olakšava skaliranje MongoDB baze podataka u usporedbi s relacijskim bazama podataka. Korištenje dokumenata je također prednost jer se objekti mapiraju u izvorne tipove podataka u brojnim programskim jezicima. Time se smanjuju potrebe za povezivanjem baze podataka i smanjuju se

troškovi. Budući da je njegova osnovna funkcija horizontalna skalabilnost, MongoDB je izrazito koristan za tvrtke koje koriste velike podatkovne aplikacije. [10, 11]

S druge strane, postoje i poneki nedostaci. MongoDB ne podržava višestruke glavne čvorove, što znači da se postavlja samo jedan glavni čvor u MongoDB klasteru. Taj jedan čvor ograničava brzinu upisivanja podataka. Ako *master* ne uspije, drugi čvor će se automatski pretvoriti u novog *mastera*. Ovo prebacivanje je konstantno, no nije trenutno, odnosno može malo i potrajati. Osim toga, MongoDB ne pruža potpuni referentni integritet korištenjem ograničenja stranog ključa (engl. *foreign key*), što može utjecati na konzistentnost podataka. Također se tu javljaju i problemi s autentičnošću korisnika zbog čega su se i dogodili brojni napadi koji su bili ciljani na veliki broj nezaštićenih sustava. Zbog toga su i dodane postavke koje blokiraju veze s bazom podataka ako nisu konfigurirane od strane administratora baze podataka. [10, 11]

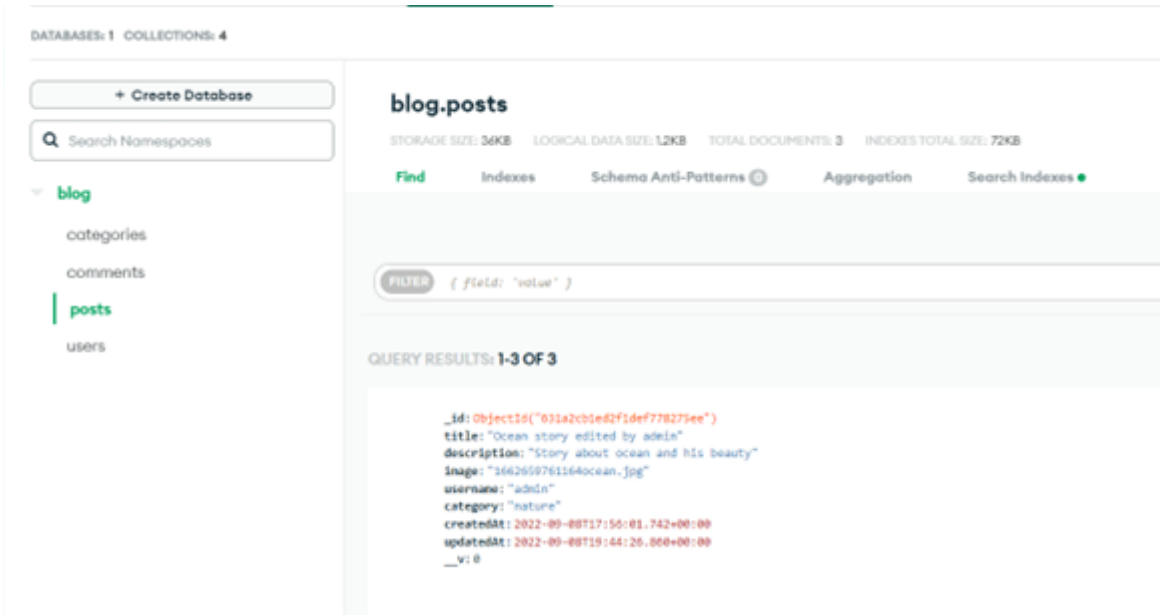
Jedna od takvih preventivnih postavki za zaštitu baze podataka je i ograničeni pristup vlastitom klasteru. Da bi se moglo pristupiti klasteru, potrebno je unijeti popis IP adresa kojima je dopušteno ostvariti klijentsku vezu s klasterom. IP adrese se unose unutar IP pristupne liste projekta, gdje bi svaki unos trebao biti ili jedna ili niz IP adresa. Primjer unosa pristupne liste se nalazi na slici 2.5.



Slika 2.5. *Primjer definiranja IP pristupne liste*

U slučaju da web aplikacija pohranjuje nekakve podatke, tada je korisno imati bazu podataka koja jednako lako radi sa React-om, Express-om i Node-om. MongoDB je jedna od takvih baza podataka. Cijeli postupak se svodi na to da se JSON dokumenti kreirani u React-u šalju na

Express.js poslužitelj, gdje se obrađuju. Nakon obrade, pod pretpostavkom da su valjani, se izravno pohranjuju u MongoDB za kasnije preuzimanje. Primjer izgleda MongoDB baze podataka se nalazi na slici 2.6. [10, 11]



Slika 2.6. *Primjer izgleda MongoDB baze podataka s detaljnijim pogledom na dokument „posts“*

2.3.2. Express.js

Express je *backend* web aplikacijski okvir na strani poslužitelja koji obavija HTTP (*Hypertext Transfer Protocol*) zahtjeve i odgovore za Node.js. Dizajniran je za izradu web aplikacija i API-ja. Kao što je već spomenuto, Express.js je *backend* komponenta razvojnih skupina tehnologija kao što su MEAN, MERN ili MEVN stog, zajedno sa MongoDB bazom podataka i bilo kojim JavaScript *frontend* okvirom. [12]

U svrhu ovog projekta, Express.js se koristi za izradu RESTful API-ja – programsko aplikacijsko sučelje koje je u skladu s ograničenjima REST arhitektonskog stila te omogućuje interakciju s RESTful web uslugama. REST je zapravo skraćenica od reprezentativnog prijenosa stanja (engl. *Representational State Transfer*) te predstavlja skup arhitektonskih ograničenja, a ne protokol ili standard. RESTful API radi na način da kada klijent podnese zahtjev, RESTful API prenosi prikaz stanja resursa samom podnositelju zahtjeva. Resursi se prenose u jednom od nekoliko formata putem HTTP protokola: JSON, HTML, PHP i mnogi drugi. Naravno od svih formata, JSON je najpopularniji. To je zato što je nevezan za sami jezik te je čitljiv i ljudima i strojevima. [13]

Express.js se također koristi za izradu jedne ili više stranica te za izradu hibridnih web aplikacija. Pomaže u upravljanju poslužiteljima i rutama te uštedi puno vremena tijekom pisanja koda. Napisan je u JavaScript programskom jeziku i time pomaže novim programerima da lakše „uđu“ u područje web programiranja. [14]

Postoje mnogi razlozi zašto je Express okvir kreiran za Node.js. Neki od tih razloga su sljedeći [14]:

- vremenska učinkovitost,
- brzina,
- ekonomičnost,
- lakoća učenja,
- asinkronost.

Neke od značajki koje bi trebalo izdvojiti su brzi razvoj na strani poslužitelja, *middleware*, odnosno rukovatelj zahtjeva (engl. *request handler*), usmjeravanje (engl. *routing*), pružanje predložaka (engl. *templating*) i otklanjanje pogrešaka (engl. *debugging*). Brzim razvojem na strani poslužitelja se uštedi puno vremena, dok rukovatelj zahtjeva ima pristup aplikacijskom zahtjev-odgovor ciklusu. Usmjeravanje se odnosi na to kako URL-ovi (*Uniform Resource Locators*) krajnjih točaka aplikacije odgovaraju na zahtjeve klijenata. Pružanje predložaka se obavlja za izgradnju dinamičkog sadržaja na web stranicama na takav način da se stvore HTML predlošci na poslužitelju. Otklanjanje pogrešaka je dosta olakšano jer Express sam identificira točan dio na kojem se nalaze pogreške. [14]

Instalacija Express.js-a je poprilično jednostavna. Prvo je potrebno instalirati Node.js te naredbom *npm install express* u terminalu instalirati Express. Kako bi Express server slušao aplikaciju, potrebno je instalirati biblioteku „nodemon“ pomoću naredbe *npm install nodemon*. Nakon instalacije, potrebno je unutar datoteke *package.json* uključiti samu „nodemon“ biblioteku. Nakon toga potrebno je unutar datoteke (u našem slučaju datoteka *index.js*) kreirati Express server i aplikaciju, koja osluškuje zadani port. Pokretanje same aplikacije se vrši s naredbom *npm start*. Takav primjer se nalazi na slici 2.7.

Slika 2.7. *Primjer kreiranja Express servera i aplikacije*

Od nekakvih prednosti Express.js-a, najviše se izdvaja to što koristi jedan jezik i za *frontend* i za *backend*, uz brzo povezivanje s bazama podataka kao što su MySQL, već spomenuti MongoDB i mnoge druge. Također bi se moglo izdvojiti i dinamičko prikazivanje HTML stranica na temelju prosljeđivanja argumenata predlošcima. [14]

Što se tiče nekakvih ograničenja ili nedostataka, izdvojiti se može to što ne postoji nekakav strukturalni način organiziranja te napisani kod može postati nerazumljiv. Također postoji dosta problema s povratnim pozivima. Kada dođe do nekakve pogreške, poruke koje obavještavaju korisnika mogu biti teško razumljive. [14]

2.3.3. React.js

React.js je JavaScript biblioteka koja služi za izgradnju korisničkih sučelja temeljenih na komponentama korisničkog sučelja. Koristi se kao baza u razvoju jednostranih (engl. *single-page*), mobilnih i poslužiteljskih aplikacija s *okvirima* kao što je primjerice Next.js. No ono što je glavno kod korištenja React-a je zapravo upravljanje stanjima i prikazivanje tih stanja u DOM-u. Stoga stvaranje React aplikacija često zahtjeva i korištenje dodatnih biblioteka za usmjeravanje kao i određene funkcionalnosti na strani klijenta. [15]

React sadrži mnoge značajke od kojih se najviše ističu deklarativno programiranje (engl. *declarative*), komponente (engl. *components*), virtualni DOM, metode životnog ciklusa (engl. *lifecycle methods*), JSX i React kuke (engl. *React hooks*). [15]

React se zapravo pridržava paradigme deklarativnog programiranja. Programeri sami dizajniraju jednostavne prikaze (engl. *views*) za svako stanje u aplikaciji, dok React učinkovito ažurira i generira komponente kada se podaci promjene. Deklarativni prikazi čine kod predvidljivijim i lakšim za otklanjanje pogrešaka. [15]

Ono od čega se React sastoji su zapravo komponente. One se mogu ponovno koristiti i moraju se formirati unutar mape *src* (*source*) prema *Pascal Case* konvenciji imenovanja. Prilikom renderiranja same komponente, moguće je proslijediti određene vrijednosti između komponenti putem svojstava, odnosno „*props*“. Dva primarna načina deklariranja komponenti u React-u su [15]:

- putem funkcionalnih komponenti,
- putem komponenti temeljenih na klasama.

Funkcionalne komponente se deklariraju s funkcijom koja vraća neki JSX, dok se komponente temeljene na klasama deklariraju pomoću ES6 klasa. Na slici 2.8. se nalazi primjer funkcionalnih komponenti, a na slici 2.9. primjer komponente temeljene na klasama. [15]

```
const handleSocialMedia = async (link) => {
  window.open(link + window.location.href, "_blank");
};

const handleCopyClipboard = async () => {
  navigator.clipboard.writeText(window.location.href);
  setMessage(true);
  setTimeout(() => {
    setMessage(false);
  }, 2000);
};
```

Slika 2.8. *Funkcionalne komponente*

```
class ParentComponent extends React.Component {
  state = { color: 'green' };
  render() {
    return (
      <ChildComponent color={this.state.color} />
    );
  }
}
```

Slika 2.9. *Komponenta temeljena na klasi* [15]

Komponente temeljene na klasama se odnose na korištenje klasa i metoda životnog ciklusa, dok funkcionalne komponente imaju React *kuke* (engl. *React hooks*) za rješavanje problema upravljanja stanjima i rješavanje drugih problema koji nastaju prilikom pisanja programskog koda. [15]

Osim prema načinu deklariranja, komponente se mogu podijeliti na [16]:

- jednostavne komponente,
- komponente sa stanjem,
- komponente koje koriste vanjske dodatke (engl. *plugins*).

Jednostavne komponente implementiraju metodu *render()* koja prima ulazne podatke i vraća ono što se treba prikazati. Primjer na slici 2.10. koristi sintaksu sličnu XML-u (*Extensible Markup Language*) zvanu JSX. Ulaznim podacima koji se prosljeđuju u komponentu se može pristupiti metodom *render()* pomoću svojstva – *this.props*. [16]

Osim preuzimanja ulaznih podataka (pristupa im se pomoću *this.props*), komponenta može održavati unutarnja stanja podataka, kojima se pristupa putem *this.state*. Prilikom promjene stanja podataka komponente, renderirana oznaka bit će ažurirana ponovnim pozivanjem metode *render()*. Budući da React omogućuje povezivanje s drugim bibliotekama i okvirima, moguće je koristiti komponente koje koriste vanjske dodatke. [16]

```
class HelloMessage extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}

root.render(<HelloMessage name="Max" />);
```

Slika 2.10. *Primjer jednostavne komponente*

Još jedna od spomenutih značajki je virtualni model dokumenta ili skraćeno virtualni DOM. React unutar memorije stvara *cache* strukturu podataka, izračunava nastale razlike i učinkovito ažurira prikazani DOM preglednika. Takav proces se naziva „pomirenje“ (engl. *reconciliation*). Ovaj postupak omogućuje programerima da pišu programski kod kao da se cijela stranica prikazuje pri svakoj promjeni koja se dogodi. Zapravo React biblioteke prikazuju samo podkomponente koje se stvarno mijenjaju. Takvo selektivno prikazivanje povećava performanse, štedi trud ponovnog izračunavanja CSS stila, izgleda stranice i renderiranja za cijelu stranicu. [15]

Kao što je već spomenuto, komponente temeljene na klasama koriste metode životnog ciklusa. One koriste oblik spajanja koji omogućuje izvršavanje koda u zadanim točkama tijekom životnog vijeka komponente. Postoje četiri metode životnog ciklusa [15]:

- metoda *shouldComponentUpdate,*
- metoda *componentDidMount,*
- metoda *componentWillUnmount,*
- metoda *render*.

Metoda *shouldComponentUpdate* omogućuje programeru da spriječi nepotrebno prikazivanje komponente vraćanjem vrijednosti *false,* ako samo renderiranje nije potrebno. Metoda *componentDidMount* se poziva nakon što se komponenta „montira“. Obično se koristi za pokretanje učitavanja podataka s udaljenog izvora putem API-ja. Metoda *componentWillUnmount* se poziva neposredno prije nego li se komponenta sruši ili „demontira“. Obično se koristi za brisanje ovisnosti o komponentama koje zahtijevaju resurse i koje se neće jednostavno ukloniti demontažom komponente. Metoda *render* je najvažnija metoda životnog ciklusa i jedina potrebna

u bilo kojoj komponenti. Obično se poziva svaki puta kada se ažurira stanje komponente, što se odražava i na korisničkom sučelju. [15]

JSX ili *JavaScript Syntax Extension*, je proširenje sintakse JavaScript jezika. Ono pruža način strukturiranja renderiranja komponenti koristeći programerima vrlo dobro poznatu sintaksu (po izgledu slično HTML-u). React komponente se obično pišu pomoću JSX-a, no ne moraju se nužno. [15]

React *hooks*, odnosno React *kuke* su funkcije koje programerima dopuštaju da se „priključe" na React stanje i značajke životnog ciklusa iz funkcionalnih komponenti. React *hooks* ne rade unutar klasa već omogućuju korištenje samog React-a bez klasa. Neke od poznatijih kuka su [15]:

- *useState*,
- *useRef*,
- *useContext*,
- *useReducer*,
- *useLocation*,
- *useEffect*.

Najčešće korištene React *hooks* su upravo *useState* i *useEffect*, koje služe za kontrolu stanja i nuspojava. Ostale su dokumentirane na Hooks API Reference poveznici [17]. Na slici 2.11. se nalazi primjer korištenja *useEffect*, *useState* i *useLocation* React *kuke*, kojom se dohvaćaju sve blog objave i spremaju u stanje *blogPosts*.

```
const [blogPosts, setBlogPosts] = useState([]);
const { search } = useLocation();

useEffect(() => {
  const fetchBlogPosts = async () => {
    // promise based HTTP client -> axios
    const res = await axios.get("/posts" + search);
    setBlogPosts(res.data);
  };
  fetchBlogPosts();
}, [search]);
```

Slika 2.11. *useEffect, useState i useLocation React kuke*

No postoje određena pravila React *kuka* koja opisuju karakteristične uzorke koda na koje se kuke oslanjaju. Jedno od pravila je to da se React *kuku* trebaju pozivati samo na gornjoj razini, odnosno da se pozivaju unutar petlji ili grananjima. Drugo pravilo kaže da se React *kuku* trebaju pozivati samo iz funkcionalnih komponenti i prilagođenih kuka, a ne iz normalnih funkcija ili komponenti temeljenih na klasama. [15]

U konačnici, React je zapravo najviša razina MERN stog tehnologija. Njime se stvaraju dinamičke aplikacije na strani klijenta te složena sučelja grupiranjem raznih komponenti. Njegova jačina je zapravo rukovanje sučeljima vođenim podacima s minimalnim kodom, a ima sve što se očekuje od modernog web okvira.

2.3.4. Node.js

Node.js je *backend* JavaScript *runtime* okruženje koje izvršava JavaScript kod izvan web preglednika. Node.js objedinjuje razvoj web aplikacije oko jednog programskog jezika umjesto različitih jezika na strani poslužitelja i klijenta. Također ima i arhitekturu vođenu događajima (engl. *event-driven*) koja je sposobna za asinkroni I/O (engl. *input/output*). Takav dizajn ima cilj optimizirati propusnost i skalabilnost u web aplikacijama s ulazno/izlaznim operacijama. [18]

Node.js omogućuje stvaranje web poslužitelja (engl. *web server*) i mrežnih alata pomoću JavaScript programskog jezika i zbirke „modula“ koji rukuju raznim funkcionalnostima. Moduli su predviđeni za I/O datotečnog sustava, umrežavanje (engl. *networking*), binarne podatke, funkcije kriptografije, tokove podataka i za druge funkcije. Oni također koriste i API dizajniran za smanjenje složenosti pisanja poslužiteljskih aplikacija. [18]

Node.js izvorno podržava samo JavaScript programski jezik, no postoje i mnogi drugi jezici koji se mogu prevoditi u JavaScript jezik. [18]

Node.js se prvenstveno koristi za izgradnju mrežnih programa kao što su web poslužitelji. U usporedbi s PHP-om, većina funkcija u PHP-u blokira izvršavanje naredbe sve dok se prethodna naredba ne izvrši. S druge strane, Node.js funkcije ne blokiraju izvršavanje naredbi jer se naredbe izvršavaju istodobno pa čak i paralelno. Node.js funkcije također koriste povratne pozive za signaliziranje završetka ili neuspjeha izvršavanja naredbe. [18]

Unutar Node.js poslužitelja se izvršava već spomenuti Express.js. Express.js sadrži modele za usmjeravanje URL-a i rukovanje HTTP zahtjevima i odgovorima. Izradom XML HTTP zahtjeva (XHR), GET-ova ili POST-ova iz React.js *frontend*-a, korisnici se povezuju s Express.js funkcijama koje pokreću aplikaciju. Te funkcije koriste MongoDB Node.js *driver*-e za pristup i ažuriranje podataka u MongoDB bazi podataka. [18, 14]

2.4. Ostale tehnologije i alati

Od ostalih tehnologija i alata, korišteni su SCSS i CSS za stilsko uređenje kompletnog projekta, dok se već spomenuti Postman koristio za testiranje, izgradnju i praćenje API poziva. Svaka linija koda je napisana unutar već dobro poznatog uređivača koda – Visual Studio Code.

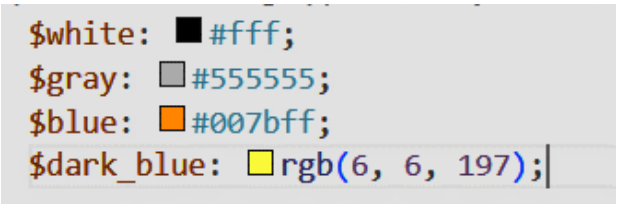
2.4.1. SASS

SASS (engl. *Syntactically Awesome Style Sheets*) predstavlja pretprocesorski skriptni jezik koji se prevodi u CSS (engl. *Cascading Style Sheets*). Sastoji se od dvije sintakse – izvorna sintaksa i novija sintaksa. Izvorna sintaksa se još naziva i „uvučena sintaksa“ (engl. *the indented syntax*), te koristi uvlačenje za odvajanje blokova koda i znakove novog retka za odvajanje pravila. Novija sintaksa se još naziva i SCSS (*Sassy CSS*). Ona koristi zagrade za označavanje i formatiranje blokova te točku-zarez za odvajanje pravila unutar samog bloka, baš poput običnog CSS-a. Datoteke uvučene sintakse imaju ekstenzije *.sass,* dok SCSS datoteke imaju ekstenziju *.scss*. [19]

SASS generalno proširuje sami CSS na način da pruža nekoliko mehanizama dostupnih u tradicionalnijim programskim jezicima. Interpretacijom stvara blokove CSS pravila za različite selektore te prevodi *SassScript* u CSS. Svakim spremanjem *.sass* ili *.scss* datoteke se prevode u izlaznu datoteku *.css*. [19]

Ukoliko se koristi Visual Studio Code za uređivanje koda, moguće je instalirati proširenje (engl. *extension*) za prevođenje SASS/SCSS datoteke unutar samog uređivača. Na taj način je omogućeno automatsko prevođenje izvorne SASS/SCSS datoteke u CSS datoteku.

SassScript također pruža i mehanizme varijable, gniježđenja (engl. *nesting*), *mixine* i nasljeđivanje selektora. Varijable počinju znakom dolara ($), dok se samo dodjeljivanje varijabli vrši s dvotočkom. Tipovi varijabli mogu biti brojevi, nizovi, boje i *Boolean*. Same varijable mogu biti argumenti ili rezultati dostupnih funkcija, dok se njihove vrijednosti umeću unutar izlaznog CSS dokumenta. Primjer definiranja varijabli se nalazi na slici 2.12. [19]



Slika 2.12. *Definiranje varijabli unutar SCSS datoteke*

Što se tiče ugnježđivanja, CSS podržava ugnježđivanje, no sami blokovi koda nisu ugniježđeni. SASS/SCSS dopušta da se ugniježđeni kodovi zapravo umetnu jedan u drugog. Također dopušta i iteraciju preko varijabli koristeći petlje *@for*, *@each* i *@while*. S druge strane, nasljeđivanje selektora nije dopušteno unutar CSS-a. U SASS/SCSS-u se nasljeđivanje postiže umetanjem retka unutar bloka koda koji koristi ključnu riječ *@extend* i upućuje na drugi selektor. Svi atributi proširenog selektora se primjenjuju na pozivajući selektor. Primjer ugnježđivanja i nasljeđivanja selektora se nalazi na slici 2.13. [19]

Kako bi stilovi komponenata bili što pregledniji, unutar svake SCSS datoteke je korištena BEM (engl. *Block Element Modifier*) metodologija. BEM metodologija pomaže u stvaranju komponenti za višekratnu uporabu i dijeljenje tijekom *frontend* razvoja. Više o BEM metodologiji se može pogledati na poveznici [20].

```scss
.settings {
  display: flex;
  font-family: "Roboto", sans-serif;

  &__button {
    width: 100px;
    height: 40px;
    margin-top: 10px;
    border: 2px solid $blue;
    border-radius: 4px;
    background-color: $blue;
    color: $white;
    text-align: center;
    align-self: center;
    cursor: pointer;

    &--delete {
      @extend .settings__button;
      width: 200px;
      margin-right: 100px;
    }

    &:hover {
      background-color: $dark_blue;
      transition: 0.5s ease;
    }
  }
```

Slika 2.13. *Primjer ugnježđivanja i nasljeđivanja selektora*

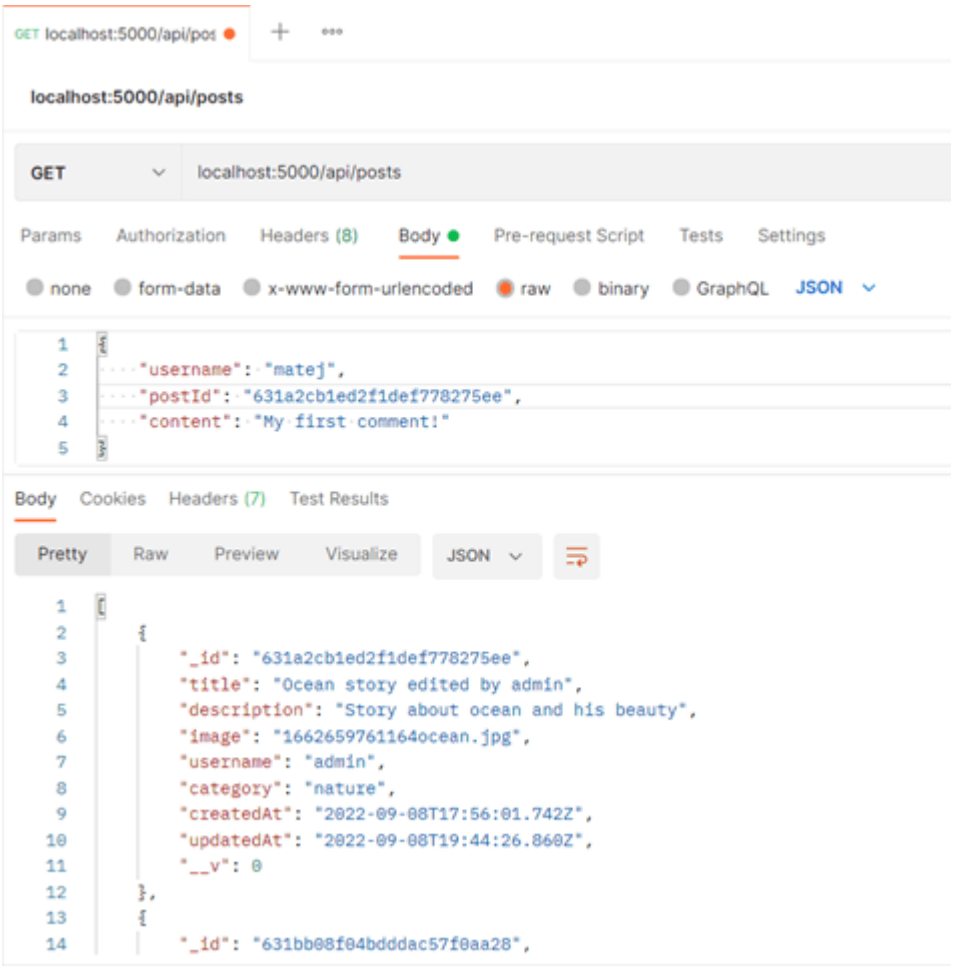
2.4.2. Postman

Postman predstavlja API platformu za izradu, korištenje, testiranje i praćenje API-ja. On pojednostavljuje svaki korak životnog ciklusa API-ja i poboljšava suradnju tako da korisnici mogu što brže stvarati sve bolje API-je. [21]

Postman omogućuje jednostavnu pohranu i suradnju oko svih API artefakata na jednoj središnjoj platformi. Ima mogućnost upravljanja specifikacijama API-ja, dokumentacijom, testnim slučajevima, rezultatima i sa svime ostalo što je povezano sa API-jima. Uključuje alate koji ubrzavaju životni ciklus API-ja od dizajniranja, testiranja, dokumentacije, izgradnje maketa (engl. *mocking*) pa sve do dijeljenja i otkrivanja API-ja. [21]

Poprilično je jednostavan za korištenje. Prvo se odabere metoda kojom će se utjecati na API te se zatim unosi URL zahtjeva u predviđeni prostor. Prilikom slanja zahtjeva se može odabrati oblik

u kojem će se podaci slati i primati, te navesti imena podataka na koje se želi utjecati. Na slici 2.14. se nalazi primjer korištenja Postman-a u kojem se dohvaćaju sve blog objave u JSON obliku.



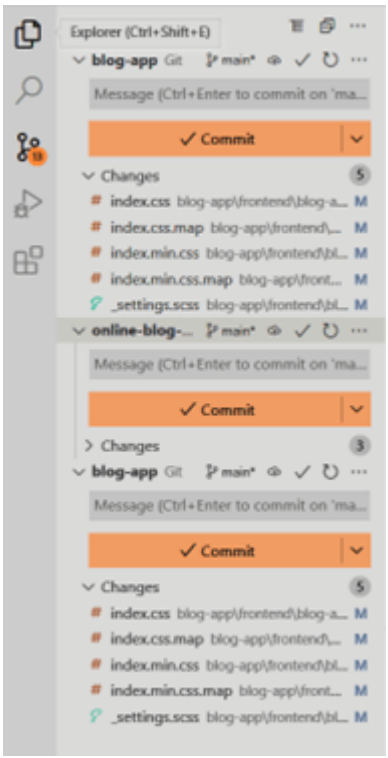
Slika 2.14. *Primjer korištenja Postman-a*

2.4.3. Microsoft Visual Studio

Visual Studio Code, skraćeno VS Code, je uređivač izvornog koda koji sadrži mnoge značajke poput podrške za otklanjanje pogrešaka, isticanje i bojanje sintakse, inteligentno dovršavanje koda, refaktoriranje i ugrađeni Git. Od ostaloga, moguće je instalirati razna proširenja, mijenjati razne tipkovničke prečace i postavke. [22]

Također VS Code uključuje i osnovnu podršku za većinu uobičajenih programskih jezika. U tu osnovnu podršku spada isticanje sintakse, podudaranje zagrada, preklapanje koda i sl. Osim toga, VS Code se može proširiti putem raznih proširenja. Proširenja dodavaju podršku za nove jezike, teme, nove programe za ispravljanje pogrešaka te za analizu samog koda. [22]

Jedna od ugrađenih značajki VS Code-a je kontrola izvora (engl. *source control*). Ono ima i zasebnu namjensku karticu unutar trake izbornika gdje korisnici mogu pristupiti postavkama kontrole verzija i vidjeti napravljene promjene na trenutnom projektu. Primjer takvog izbornika gdje korisnici mogu vidjeti napravljene promjene se nalazi na slici 2.15. Da bi se ta značajka mogla iskoristiti, potrebno je da je VS Code povezan s bilo kojim podržanim sustavom za kontrolu verzije (Git, Apache Subversion, itd.) Ovakva značajka je vrlo korisna korisnicima na način da im omogućuje stvaranje repozitorija kao i slanje i povlačenje zahtjeva izravno iz VS Code-a. [22]

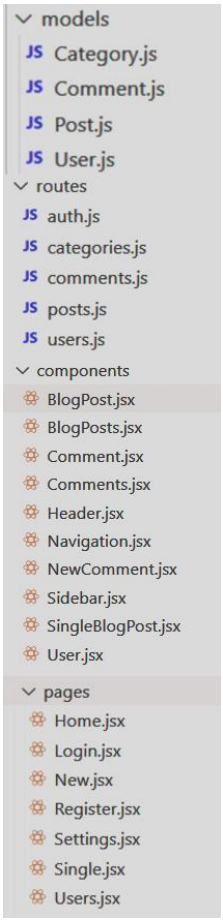


Slika 2.15. *Primjer izbornika gdje se vrši kontrola izvora*

3. IZGLED I NAČIN RADA APLIKACIJE

Za izradu web aplikacije za pisanje *online* dnevnika/ bloga, bilo je dogovoreno koristiti React *okvir* iliti *framework,* dok je za ostale tehnologije bilo potrebno dobro razmisliti kako bi se aplikacija odradila na najbolji mogući način. Odlučeno je da će se za izradu koristiti MERN tehnologije. Točnije, odlučeno je da će se za bazu podataka koristiti MongoDB, dok će se Express.js koristiti kao *backend* web aplikacijski *okvir* na strani poslužitelja s ciljem kreiranja RESTful API-ja koji prihvaća zahtjeve s *frontend*-a te povratno šalje odgovarajući odgovor. RESTful API će, kao i svaka druga linija koda, biti napisan unutar Microsoft Visual Studio Code uređivača. Unutar API-ja će biti uključen Express server, rute (engl. *routes*), modeli (engl. *models*) te povezivanje s MongoDB bazom podataka.

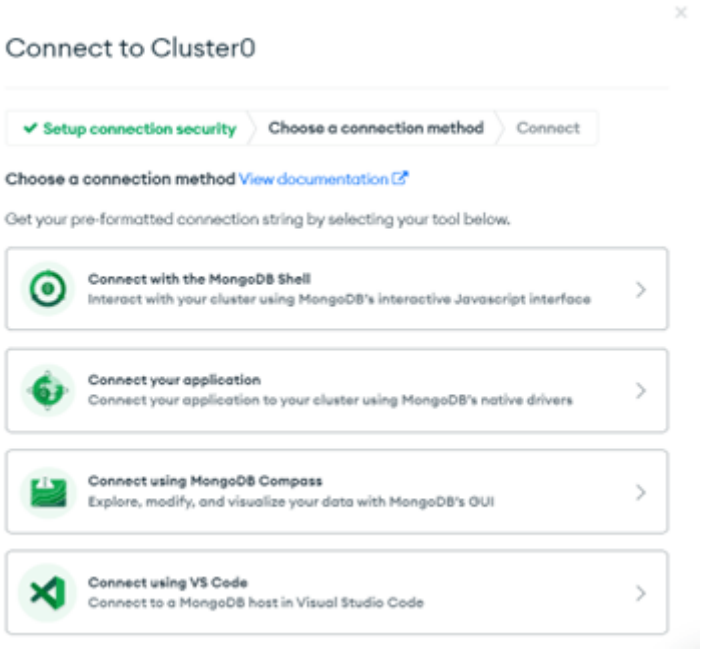
Unutar ovog poglavlja će se detaljnije opisati način na koji su primijenjene tehnologije koje su opisane u prethodnom poglavlju. Sama web aplikacija zapravo predstavlja primjer aplikacije unutar koje je moguće objaviti vlastiti blog, objavu iliti dnevnik. Služi za dijeljenje vlastitih misli, znanja i mudrosti s ostalim korisnicima. Na slici 3.1. je prikazana struktura cjelokupnog projekta.



Slika 3.1. *Struktura projekta*

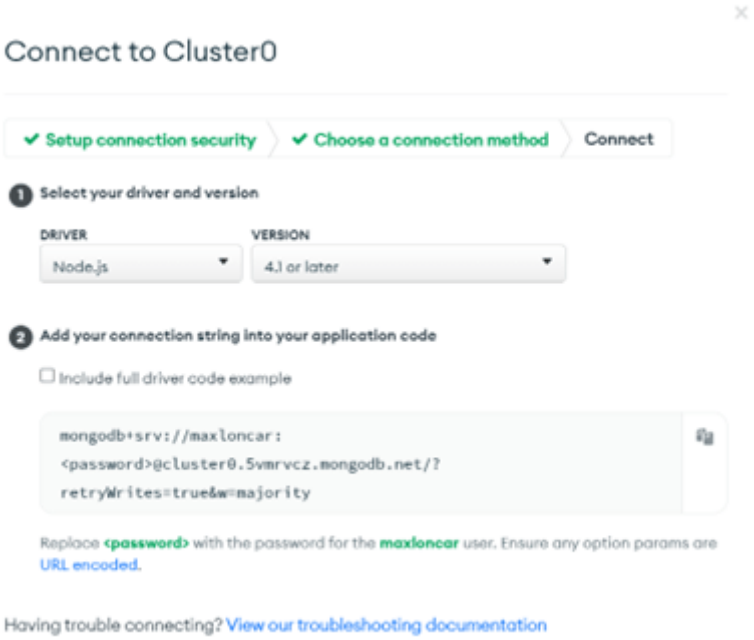
3.1. Baza podataka

Za početak je potrebno se prijaviti, odnosno registrirati unutar sustava MongoDB. Nakon toga je potrebno kreirati vlastiti klaster unutar kojeg se mogu kreirati i povezati razne kolekcije. Klikom na *Connect*, korisniku se otvara skočni prozor gdje može odabrati metodu kojom želi povezati vlastiti projekt. U ovom slučaju je korištena metoda *Connect your application* gdje se baza podataka povezuje unutar programskog koda. Primjer odabira metode se nalazi na slici 3.2.



Slika 3.2. *Primjer odabira metode za povezivanje s bazom podataka*

Nakon toga, korisniku se prikazuje prozor s privatnim URL-om. URL je potrebno kopirati i zalijepiti negdje unutar samog projekta. Unutar URL-a je potrebno unijeti vlastito korisničko ime i zaporku, definirane unutar sustava MongoDB, točnije pod izbornikom *Database Access*. Primjer prikaza privatnog URL-a se nalazi na slici 3.3. Nakon toga, kreće se sa izradom same web aplikacije.



Slika 3.3. *Primjer prikaza privatnog URL-a*

3.2. Korisnik

Korisnik da bi vidio blog objave može biti i prijavljen i anoniman. Anoniman korisnik ima mogućnost samo pregledavanja i filtriranja blog objava pomoću tražilice te klikom na pojedinog autora, odnosno kategoriju objava. S druge strane, prijavljeni korisnik ima mogućnost dodavanja, uređivanja i brisanja blog objava, detaljnijeg pregleda te pisanja i brisanja komentara na pojedinim blog objavama. Na slici 3.4 se nalazi predložak za registraciju korisnika.

Na navedenom predlošku se vidi da je za uspješnu registraciju potrebno unijeti ime, prezime, unikatne korisničko ime, elektroničku poštu i zaporku. Za samu registraciju korisnika je korištena funkcija *handleSubmit*, koja je preko uključene biblioteke *axios* pozivala metodu POST koja služi za kreiranje novih podataka. Na slici 3.5. prikazana je POST metoda koja uzima primljene podatke i prosljeđuje ih u *User* kolekciju. Za *hashiranje* zaporke se koristila biblioteka *bcrypt*, kako se sama zaporka ne bi spremala unutar baze podataka, odnosno kako ne bi bila vidljiva.

```jsx
<div className="register">
  <h1 className="register__title">Register</h1>
  <form className="register__form" onSubmit={handleSubmit}>
    <label>First name</label>
    <input
      type="text"
      className="register__input"
      placeholder="Enter your first name..."
      onChange={(e) => setFirstname(e.target.value)}
      required
    />
    <label>Last name</label>
    <input
      type="text"
      className="register__input"
      placeholder="Enter your last name..."
      onChange={(e) => setLastname(e.target.value)}
      required
    />
    <label>Username</label>
    <input
      type="text"
      className="register__input"
      placeholder="Enter your username..."
      onChange={(e) => setUsername(e.target.value)}
      required
    />
    <label>Email</label>
    <input
      type="text"
      className="register__input"
      placeholder="Enter your email..."
      onChange={(e) => setEmail(e.target.value)}
      required
    />
    <label>Password</label>
    <input
      type="password"
      className="register__input"
      placeholder="Enter your password..."
      onChange={(e) => setPassword(e.target.value)}
      required
    />
    {error && <p className="danger">{error}</p>}
    <button className="register__button" type="submit">
      Register
    </button>
    <p className="register__need-account">
      Already have an account?{" "}
      <span>
        <Link className="link" to="/login">
          Login here
        </Link>
      </span>
    </p>
  </form>
</div>
```

Slika 3.4. *Predložak za registraciju korisnika*

```
const router = require("express").Router();
const User = require("../models/User");

// bcrypt library for hashing the password
const bcrypt = require("bcrypt");

// REGISTER
router.post("/register", async (req, res) => {
  try {
    // generate a salt and hash
    // a salt with 10 hashes per second (10 rounds)
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(req.body.password, salt);

    const newUser = new User({
      username: req.body.username,
      firstname: req.body.firstname,
      lastname: req.body.lastname,
      email: req.body.email,
      password: hashedPassword,
    });

    const user = await newUser.save();
    res.status(200).json(user);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Slika 3.5. *Metoda za registraciju korisnika*

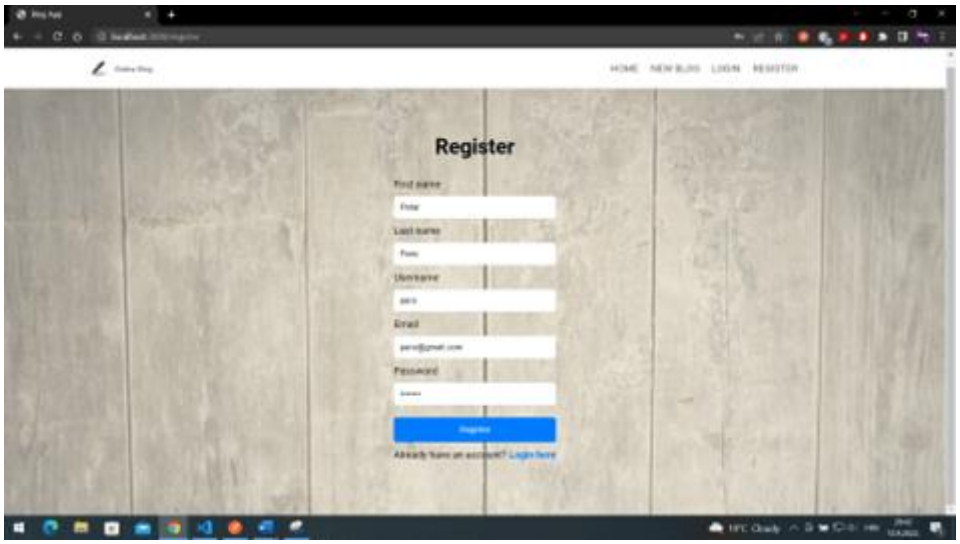
Kolekcija *User* je zapravo jedan od modela ovog čitavog projekta. Svi modeli su definirani pomoću sheme koja definira strukturu i sadržaj podataka. Primjer sheme *UserSchema* se nalazi na slici 3.6. Primjer prikaza MongoDB kolekcije s korisničkim podacima se nalazi na slici 3.7., dok se primjer konačnog izgleda *Register* stranice nalazi na slici 3.8.

```
const mongoose = require("mongoose");
const UserSchema = new mongoose.Schema(
  {
    username: {
      type: String,
      required: true,
      unique: true,
    },
    firstname: {
      type: String,
      required: true,
    },
    lastname: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    image: {
      type: String,
      default: "default-user.jpg",
    },
    isAdmin: {
      type: Boolean,
      default: false,
    },
  },
  { timestamps: true }
);
module.exports = mongoose.model("User", UserSchema);
```

Slika 3.6. *Shema modela User*

```
_id: ObjectId('6314c9a5e38275f91cd2e1e1')
username: "markomarkovic"
firstname: "Marko"
lastname: "Markovic"
email: "marko@gmail.com"
password: "$2b$10$LEDQ2.cXUtU/MgN1hJ0bG.SowwGV3WWU4c35sTKz2/dpObMtQ8jMK"
image: "1662587249408user1.jpg"
createdAt: 2022-09-04T15:52:05.810+00:00
updatedAt: 2022-09-10T16:37:13.386+00:00
__v: 0
isAdmin: false
```


Slika 3.7. *Primjer prikaza podataka unutar MongoDB kolekcije*



Slika 3.8. *Stranica za registraciju korisnika*

Korisnik prilikom registracije dobiva zadanu vrijednost za atribut „isAdmin“, a to je vrijednost *false*. To označava da je korisnik zapravo običan korisnik bez administratorskih uloga. Nakon što je korisnik registriran, usmjerava ga se na stranicu za prijavu. Na slici 3.9. se nalazi predložak za prijavu korisnika.

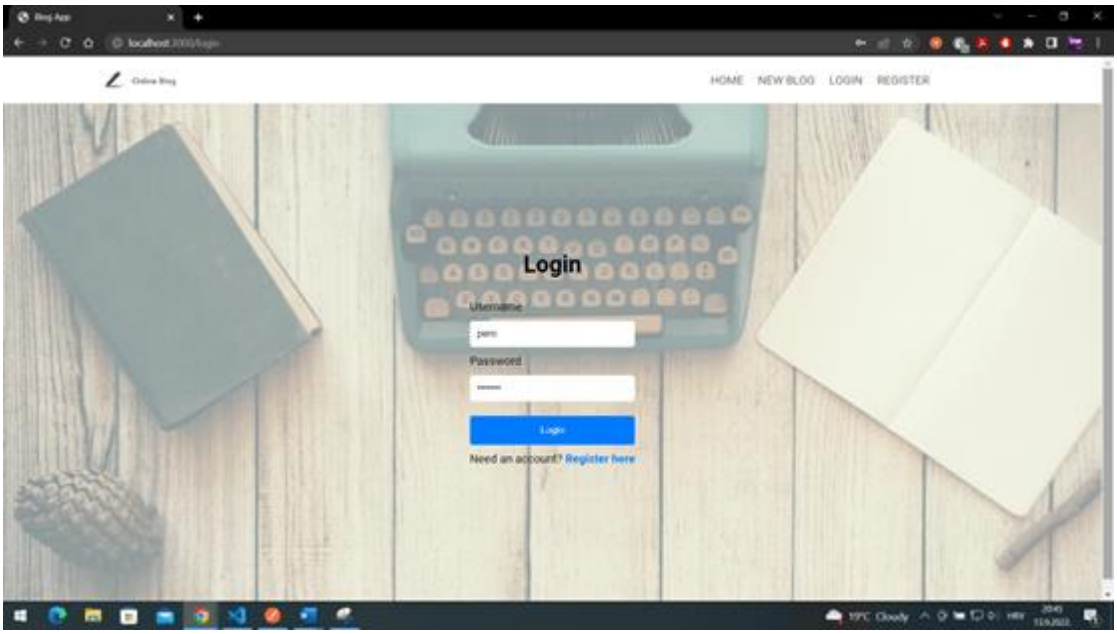
```jsx
<div className="login">
  <h1 className="login__title">Login</h1>
  <form className="login__form" onSubmit={handleSubmit}>
    <label>Username</label>
    <input
      type="text"
      className="login__input"
      placeholder="Enter your username..."
      ref={userRef}
      required
    />
    <label>Password</label>
    <input
      type="password"
      className="login__input"
      placeholder="Enter your password..."
      autoComplete="off"
      ref={passwordRef}
      required
    />
    {error && <p className="danger">{error}</p>}
    <button className="login__button" type="submit" disabled={isFetching}>
      Login
    </button>
    <p className="login__need-account">
      Need an account?{" "}
      <span>
        <Link className="link" to="/register">
          Register here
        </Link>
      </span>
    </p>
  </form>
</div>
```

Slika 3.9. *Predložak za prijavu korisnika*

Na navedenom predlošku se vidi da je za uspješnu prijavu potrebno unijeti točno korisničko ime i zaporku. Za samu prijavu korisnika je korištena funkcija *handleSubmit*, koja je također preko uključene biblioteke *axios* pozivala metodu POST. Na slici 3.10. prikazana je POST metoda unutar koje se provjeravalo postoji li korisnik s unesenim korisničkim imenom te je li *hashirana* zaporka jednaka onoj koja se nalazi unutar baze podataka. Konačan izgled *Login* stranice se nalazi na slici 3.11.

```jsx
// LOGIN
router.post("/login", async (req, res) => {
  try {
    const user = await User.findOne({ username: req.body.username });

    // if there is no user
    !user && res.status(400).json("Wrong credentials!");

    // check passwords
    const validPassword = await bcrypt.compare(
      req.body.password,
      user.password
    );

    // if password doesn't match with the database
    !validPassword && res.status(400).json("Wrong credentials!");

    // send all info except the password
    const { password, ...otherCredentials } = user._doc;
    res.status(200).json(otherCredentials);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Slika 3.10. *Metoda za prijavu korisnika*



Slika 3.11. *Izgled stranice za prijavu korisnika*

Ukoliko se korisnik poželi odjaviti, to je moguće učiniti klikom na „Logout“ unutar gornje navigacijske trake. Ukoliko korisnik poželi promijeniti vlastite korisničke podatke ili obrisati korisnički račun, to je moguće unutar *Settings* stranice na koju se dođe prilikom klika na korisničku sliku. Predložak stranice gdje je moguće promijeniti vlastite korisničke podatke se nalazi na slici 3.12.

Unutar predloška se nalaze podaci koje je moguće promijeniti – korisnička slika, ime, prezime, elektronička pošta te zaporka. Klikom na dugme „Update“ se ažuriraju uneseni podaci pozivanjem metode PUT kojoj se predaju podaci te se unutar nje provjerava je li predani ID jednak korisničkom ID-u. Također se prilikom promjene zaporke *hashira* tek unesena zaporka i pohranjuje unutar baze podataka. S druge strane, klikom na dugme „Delete Your Account“ korisnik briše vlastiti korisnički račun pozivanjem metode DELETE, gdje se provjerava je li korisnik vlasnik korisničkog računa ili ima status administratora. Na slikama 3.13. i 3.14. se nalaze metode PUT i DELETE.

```jsx
<div className="settings">
  <div className="settings__wrapper">
    <div className="settings__header">
      <h1 className="settings__title">Update Your Account</h1>
      {!user.isAdmin && (
        <button className="settings__button--delete" onClick={handleDelete}>
          Delete Your Account
        </button>
      )}
    </div>
    <form className="settings__form" onSubmit={handleUpdate}>
      <label>Profile Image</label>
      <div className="settings__profile">
        <img
          src={
            file
              ? URL.createObjectURL(file)
              : publicFolder + user.image || default_user
          }
          alt="User"
          className="settings__image"
        />
        <label htmlFor="input__file">
          <i className="settings__icon fa-solid fa-upload"></i>
        </label>
        <input
          type="file"
          id="input__file"
          onChange={(e) => setFile(e.target.files[0])}
        />
      </div>
      <label>First name</label>
      <input
        type="text"
        value={firstname}
        className="settings__input"
        onChange={(e) => setFirstname(e.target.value)}
      />
      <label>Last name</label>
      <input
        type="text"
        value={lastname}
        className="settings__input"
        onChange={(e) => setLastname(e.target.value)}
      />
      <label>Email</label>
      <input
        type="email"
        value={email}
        className="settings__input"
        onChange={(e) => setEmail(e.target.value)}
      />
      <label>Password</label>
      <input
        type="password"
        className="settings__input"
        autoComplete="on"
        onChange={(e) => setPassword(e.target.value)}
      />
      {success && <p className="success">Your profile has been updated!</p>}
      <button className="settings__button" type="submit">
        Update
      </button>
    </form>
  </div>
  <Sidebar />
</div>
```

Slika 3.12. *Predložak za izmjenu korisničkih podataka*

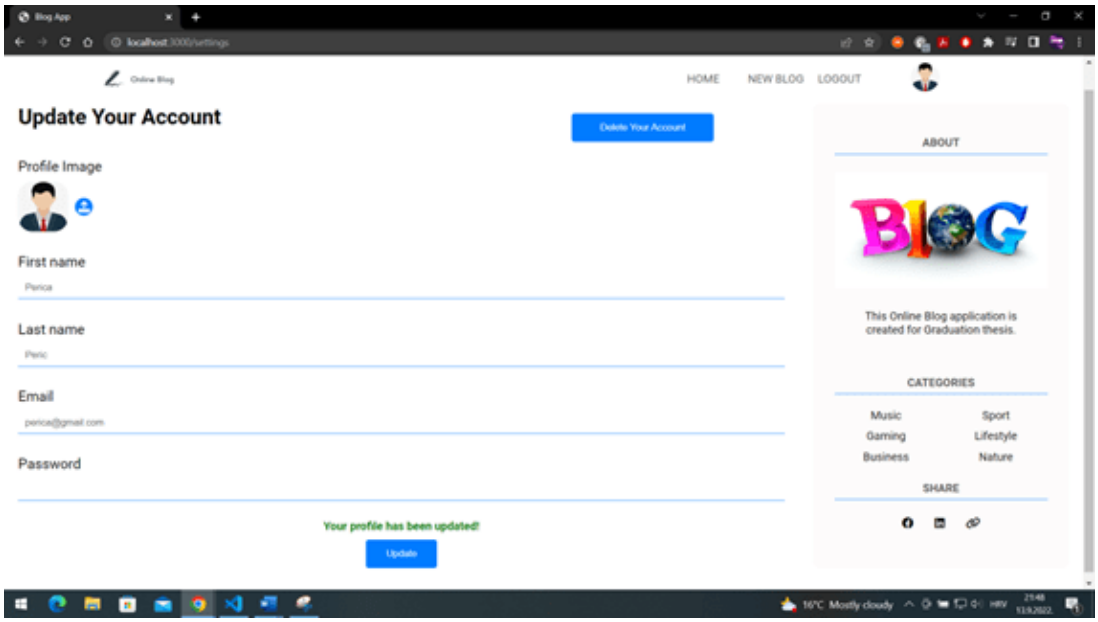
```
// UPDATE
router.put("/:id", async (req, res) => {
  // check if the user is owner of the account
  if (req.body.userId === req.params.id) {
    if (req.body.password) {
      // generate a salt and hash
      // a salt with 10 hashes per second (10 rounds)
      const salt = await bcrypt.genSalt(10);
      req.body.password = await bcrypt.hash(req.body.password, salt);
    }
    try {
      const updatedUser = await User.findByIdAndUpdate(
        req.params.id,
        {
          // set new proeprties inside request and body
          $set: req.body,
        },
        // send the new user
        { new: true }
      );
      res.status(200).json(updatedUser);
    } catch (err) {
      res.status(500).json(err);
    }
  } else {
    res.status(401).json("You can only update your account!");
  }
});
```

Slika 3.13. *Metoda za izmjenu korisničkih podataka*

```
// DELETE
router.delete("/:id", async (req, res) => {
  // check if the user is owner of the account
  if (req.body.userId === req.params.id || req.body.isAdmin === true) {
    const user = await User.findById(req.params.id);
    // check if user exists
    if (user) {
      try {
        // delete blog posts of the deleted user
        await Post.deleteMany({ username: user.username });
        await User.findByIdAndDelete(req.params.id);
        res.status(200).json("User has been deleted!");
      } catch (err) {
        res.status(500).json(err);
      }
    } else {
      res.status(404).json("User not found!");
    }
  } else {
    res.status(401).json("You can only delete your account!");
  }
});
```

Slika 3.14. *Metoda za brisanje korisnika*

Primjer cjelokupnog izgleda *Settings* stranice, unutar koje je moguće uređivati korisničke podatke te obrisati vlastiti korisnički račun, se nalazi na slici 3.15.



Slika 3.15. *Izgled Settings stranice*

3.3. Blog objave

Nakon prijave, korisnika se usmjerava na *Home* stranicu unutar koje su prikazane sve blog objave te bočna traka na kojoj se nalaze informacije o projektu te kategorije blog objava. Korisnik može otići na pojedinu blog objavu tako da klikne na dugme „Read more“. Blog objave je moguće filtrirati pomoću tražilice te klikom na pojedinu kategoriju i autora. Klikom na pojedinu kategoriju blog objava, korisniku se prikazuju samo blog objave te kategorije. Isti je slučaj i sa imenom autora gdje se klikom na korisničko ime prikazuju samo blog objave koje je napisao baš taj autor. Primjer predloška jedne blog objave se nalazi na slici 3.16.

```jsx
<div className="blog-post">
  {blogPost.image ? (
    <img
      className="blog-post__image"
      src={publicFolder + blogPost.image}
      alt="Blog post"
    />
  ) : (
    <img
      className="blog-post__image"
      src="https://media.istockphoto.com/photos/the-word-blog-arranged
      alt="Blog post"
    />
  )}
  <div className="blog-post__info">
    <div className="blog-post__categories">
      <p className="blog-post__category">{blogPost.category}</p>
    </div>
    <p className="blog-post__date">
      {new Date(blogPost.createdAt).toDateString()}
    </p>
    <hr />
    <h2 className="blog-post__title">{blogPost.title}</h2>
    <p className="blog-post__description">{blogPost.description}</p>
    <Link to={`/post/${blogPost._id}`}>
      <button className="blog-post__button">Read more</button>
    </Link>
  </div>
</div>
```

Slika 3.16. *Predložak blog objave*

Na navedenom predlošku se vide osnovne informacije blog objave kao što su slika objave, kategorija, naslov, opis te datum kreiranja same blog objave. Blog objave se dohvaćalo pomoću funkcije *fetchBlogPosts*, koja je preko uključene *axios* biblioteke pozivala metodu GET koja dohvaća pojedine podatke. Na slici 3.17. je prikazana GET metoda koja dohvaća podatke iz kolekcije *Post* prema unesenim upitima (engl. *queries*) za korisničko ime i kategoriju.

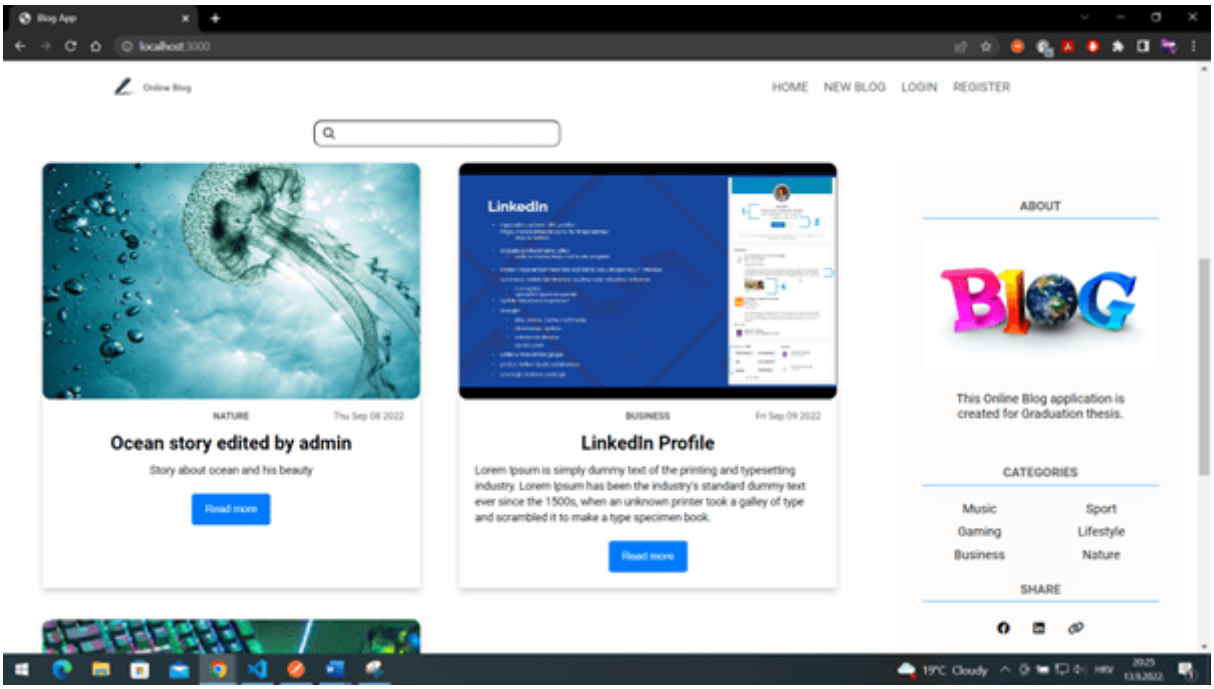
```jsx
// GET ALL BLOG POSTS
router.get("/", async (req, res) => {
  // queries inside URL (for example ?user=john, ?category=music)
  const username = req.query.user;
  const categoryName = req.query.category;

  try {
    let posts;
    if (username) {
      // get all blog posts which are written by this username
      posts = await Post.find({ username });
    } else if (categoryName) {
      posts = await Post.find({ category: categoryName });
    } else {
      // get all posts
      posts = await Post.find();
    }
    res.status(200).json(posts);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Slika 3.17. *Metoda za dohvaćanje blog objava*

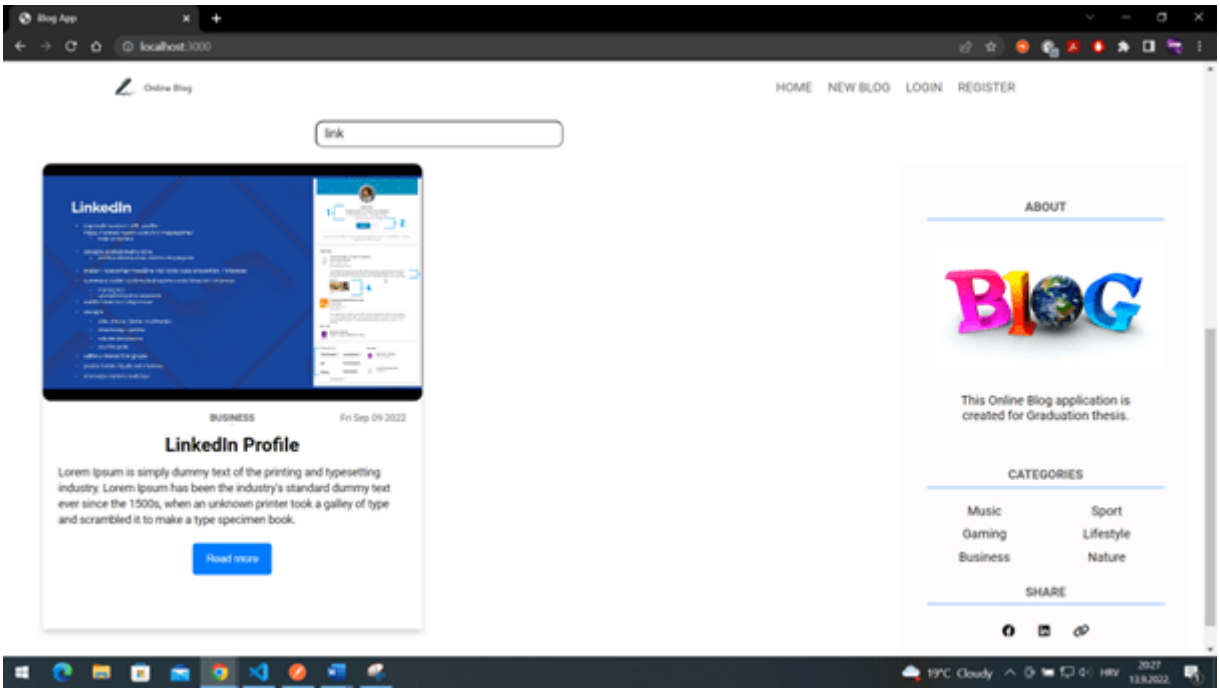
Kolekcija *Post* je također jedna od modela koja je definirana pomoću sheme. Primjer *PostSchema* se nalazi na slici 3.18. Primjer *Home* stranice s dohvaćenim blog objavama nalazi na slici 3.19., dok se primjer filtriranja blog objava pomoću tražilice nalazi na slici 3.20.

```
const mongoose = require("mongoose");
const PostSchema = new mongoose.Schema(
  {
    title: {
      type: String,
      required: true,
      unique: true,
    },
    description: {
      type: String,
      required: true,
    },
    image: {
      type: String,
      required: false,
    },
    username: {
      type: String,
      required: true,
    },
    category: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Post", PostSchema);
```

Slika 3.18. *Shema modela Post*



Slika 3.19. *Prikaz svih blog objava na Home stranici*



Slika 3.20. *Prikaz filtriranja blog objava pomoću tražilice*

Što se tiče bočne trake, u njoj su, kao što je već rečeno, prikazane informacije o projektu te kategorije blog objava. Kategorije su dohvaćene funkcijom *getCategories* koja pomoću biblioteke *axios* s metodom GET dohvaća sve kategorije iz kolekcije *Category*. Primjer GET metode se nalazi na slici 3.21.

```
// GET ALL CATEGORIES
router.get("/", async (req, res) => {
  try {
    const categories = await Category.find();
    res.status(200).json(categories);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Slika 3.21. *Dohvaćanje kategorija*

Kolekcija *Category* predstavlja također jedan od modela koji su definirani pomoću sheme. Primjer *CategorySchema* se nalazi na slici 3.22.

```
const mongoose = require("mongoose");
const CategorySchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    value: {
      type: String,
      required: true,
    },
    label: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Category", CategorySchema);
```

Slika 3.22. *Shema modela Category*

Kreiranje nove blog objave se vrši klikom na „New blog“ unutar navigacijske trake. Tada se korisniku otvara prozor unutar kojeg je moguće kreirati novu blog objavu. Za kreiranje blog objave je potrebno unijeti sliku, naslov, tekst te kategoriju. Nakon unesenih podataka, blog objava se kreira klikom na dugme „Publish“. Predložak za kreiranje nove blog objave se nalazi na slici 3.23.

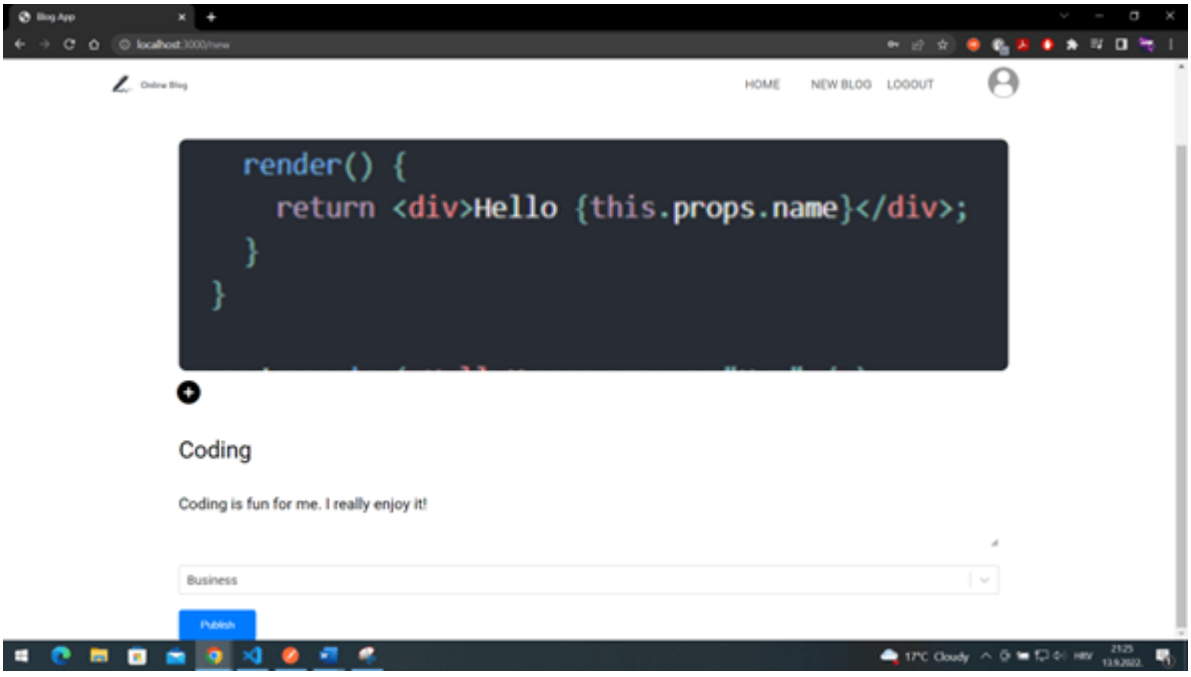
```
<Header title="Write a new blog post" />
<div className="new">
  {file && (
    <img
      src={URL.createObjectURL(file)}
      alt="New blog"
      className="new__image"
    />
  )}
  <form className="new__form" onSubmit={handleSubmit}>
    <div className="new__group">
      <label htmlFor="input__file">
        <i className="new__icon fa-solid fa-circle-plus"></i>
      </label>
      <input
        type="file"
        id="input__file"
        onChange={(e) => setFile(e.target.files[0])}
      />
      <input
        type="text"
        placeholder="Blog title"
        className="new__input new__input-title"
        autoFocus={true}
        onChange={(e) => setTitle(e.target.value)}
      />
    </div>
    <div className="new__group">
      <textarea
        placeholder="Write your story..."
        type="text"
        className="new__input new__input-description"
        onChange={(e) => setDescription(e.target.value)}
      ></textarea>
    </div>
    <Select
      options={categories}
      onChange={(choice) => setCategory(choice.value)}
    />
    <button className="new__button" type="submit">
      Publish
    </button>
  </form>
</div>
```

Slika 3.23. *Predložak za kreiranje nove blog objave*

Navedeni predložak sadrži već prethodno navedene podatke. Od svih podataka, komponenta *Select* je gotova komponenta koja se dobije iz biblioteke *react-select*. Objave su kreirane na način da se preko funkcije *handleSubmit* (koja se pokreće klikom na dugme „Publish“) pomoću *axios* biblioteke poziva metodu POST koja uzima primljene podatke i prosljeđuje ih kolekciji *Post*. Primjer metode za kreiranje nove blog objave se nalazi na slici 3.24., a konačan izgled stranice za dodavanje nove blog objave se nalazi na slici 3.25.

```
// CREATE NEW BLOG POST
router.post("/", async (req, res) => {
  const newPost = new Post(req.body);
  try {
    const savedPost = await newPost.save();
    res.status(200).json(savedPost);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Slika 3.24. *Metoda za kreiranje nove blog objave*



Slika 3.25. *Primjer kreiranja nove blog objave*

Nakon što se kreira nova blog objava, korisniku se otvara detaljniji prikaz novokreirane objave. Unutar detaljnijeg prikaza blog objave, korisnik može uređivati ili obrisati samo vlastitu objavu,

osim ako je administrator koji može brisati i uređivati sve blog objave. Klikom na ikonu olovke, korisnik uključuje tzv. *update mode*, gdje se naslov i tekst blog objave pretvaraju u polje za unos teksta. Korisnik unosi željene izmjene te ih klikom na dugme „Update“ potvrđuje i sprema. S druge strane, klikom na ikonu koša za smeće, korisnik briše vlastitu blog objavu. Obje vrste radnje se vrše unutar funkcija koje pomoću biblioteke *axios* pozivaju metode PUT i DELETE. Na slikama 3.26. i 3.27. se nalaze metode PUT i DELETE.

```
// UPDATE BLOG POST
router.put("/:id", async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);

    // check if it is user's post
    if (post.username === req.body.username || req.body.isAdmin === true) {
      try {
        const updatedPost = await Post.findByIdAndUpdate(
          req.params.id,
          {
            // set new proeprties inside request and body
            $set: req.body,
          },
          // send the new user
          { new: true }
        );
        res.status(200).json(updatedPost);
      } catch (err) {
        res.status(500).json(err);
      }
    } else {
      res.status(401).json("You can only update your blog post!");
    }
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Slika 3.26. *Metoda za uređivanje blog objave*

```
// DELETE BLOG POST
router.delete("/:id", async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);

    // check if it is user's post or if it's admin
    if (post.username === req.body.username || req.body.isAdmin === true) {
      try {
        await post.delete();
        res.status(200).json("Blog post has been deleted!");
      } catch (err) {
        res.status(500).json(err);
      }
    } else {
      res.status(401).json("You can only delete your blog post!");
    }
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Slika 3.27. *Metoda za brisanje blog objave*

Metodom PUT se ažuriraju podaci blog objave. Iako je vrlo slična metodi POST, izdvaja ju to što se prilikom uređivanja podataka provjerava je li korisnik vlasnik te blog objave ili korisnik možda ima status administratora. Metodom DELETE se brišu blog objave prema jedinstvenom ID-u kojeg svaka blog objava sadrži. No prije samog brisanja, prvo se provjerava je li korisnik vlasnik same blog objave ili ima status administratora.

3.4. Komentari

Unutar svake blog objave, moguće je ostaviti vlastiti komentar. Da bi korisnik mogao ostaviti komentar, mora biti prijavljen. Korisnik vlastiti komentar unosi unutar forme. Forma za unos komentara se sastoji od trenutne slike korisnika, prostora za unos teksta te dugmeta „Add“ kojim se dodaje sami komentar. Predložak forme za unos komentara se nalazi na slici 3.28.

```
<div className="new-comment">
  <img
    className="new-comment__image"
    src={publicFolder + user.image}
    alt="User"
  />
  <form className="new-comment__form" onSubmit={handleSubmit}>
    <textarea
      placeholder="Write your comment..."
      type="text"
      className="new-comment__content"
      value={content}
      onChange={(e) => setContent(e.target.value)}
    ></textarea>

    <button className="new-comment__button" type="submit">
      Add
    </button>
  </form>
</div>
```

Slika 3.28. *Predložak forme za unos komentara*

Za kreiranje novog komentara se pomoću funkcije *handleSubmit* (koja se pokreće klikom na dugme „Add“) s *axios* bibliotekom također koristi POST metoda, na identičan način kao i kod kreiranja blog objave. POST metoda uzima primljene podatke i prosljeđuje ih kolekciji *Comment*. Kolekcija *Comment* je jedan od modela definiranih pomoću sheme. Primjer *CommentSchema* se nalazi na slici 3.29.

```
const mongoose = require("mongoose");
const CommentSchema = new mongoose.Schema(
  {
    username: {
      type: String,
      required: true,
    },
    postId: {
      type: String,
      required: true,
    },
    content: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Comment", CommentSchema);
```

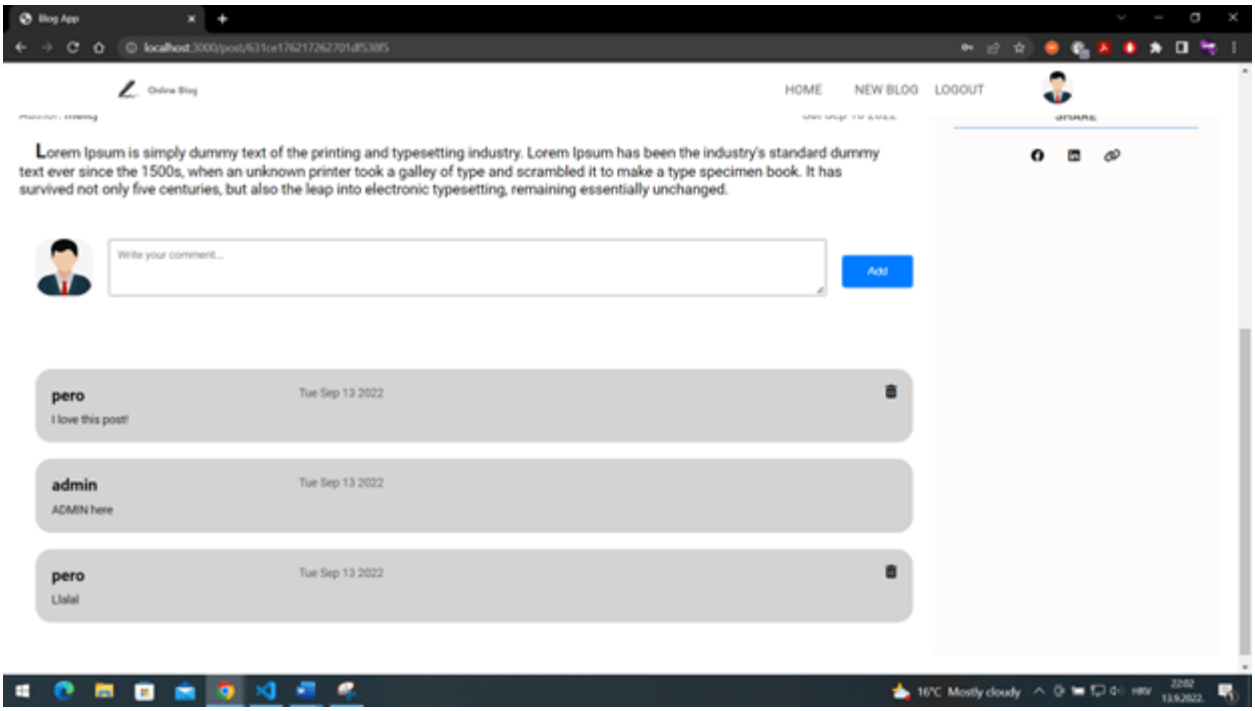
Slika 3.29. *Shema modela Comment*

Komentar se nakon dodavanja prikazuje ispod blog objave i forme za unos komentara. Primjer predloška komentara se nalazi na slici 3.30.

```
<div className="comment">
  <div className="comment__info">
    <h2 className="comment__username">{comment.username}</h2>
    <p className="comment__content">{comment.content}</p>
  </div>
  <p className="comment__date">
    {new Date(comment.createdAt).toDateString()}
  </p>
  {(user.username === comment.username || user.isAdmin === true) && (
    <i
      className="comment__icon fa-solid fa-trash-can"
      onClick={handleDelete}
    ></i>
  )}
</div>
```

Slika 3.30. *Predložak komentara*

Na navedenom predlošku se vide osnovne informacije komentara kao što su korisničko ime autora komentara, sadržaj komentara te datum nastanka samog komentara. Komentar se dohvaćalo na identičan način kao kategorije i blog objave - pomoću funkcije *getComments* koja je pomoću biblioteke *axios* koristila metodu GET. Za svaku blog objavu su se dohvaćali komentari koji pripadaju izrazito samo toj blog objavi. Prikaz komentara i forme za unos komentara se nalazi na slici 3.31.



Slika 3.31. *Prikaz komentara i forme za unos komentara*

Svaki komentar koji je napisan od strane samog korisnika je označen ikonom koša za smeće te ga je moguće obrisati. Klikom na ikonu koša za smeće se briše komentar na način da se pozove funkcija *handleDelete* koja pomoću već spomenute biblioteke *axios* poziva metodu DELETE. Brisanje je odrađeno na identičan način kao i kod brisanja blog objave, gdje se prvo provjerava je li korisnik vlasnik samog komentara ili ako ima status administratora.

3.5. Administrator

Ukoliko korisnik sadrži status administratora, tada ima mogućnost upravljanja korisnicima i sadržajem. Točnije, administrator može brisati korisnike i komentare te uređivati i brisati blog objave. Osim toga, korisnik ima poseban element navigacijske liste s nazivom „Users“. Klikom na „Users“, korisniku se prikazuju svi registrirani korisnici. Svaki od korisnika je prikazan s informacijama kao što su korisnička slika, ime, prezime, korisničko ime te elektronička pošta. Predložak korisnika se nalazi na slici 3.32.

```
{!user.isAdmin ? (
  <div className="user">
    <img
      className="user__image"
      src={publicFolder + user.image}
      alt="User"
    />
    <div className="user__info">
      <h2 className="user__data">
        {user.firstname} {user.lastname}
      </h2>
      <p className="user__data">{user.username}</p>
      <p className="user__data">{user.email}</p>
      <button className="user__button" onClick={handleDelete}>
        Delete
      </button>
    </div>
  </div>
) : (
  ""
)}
```

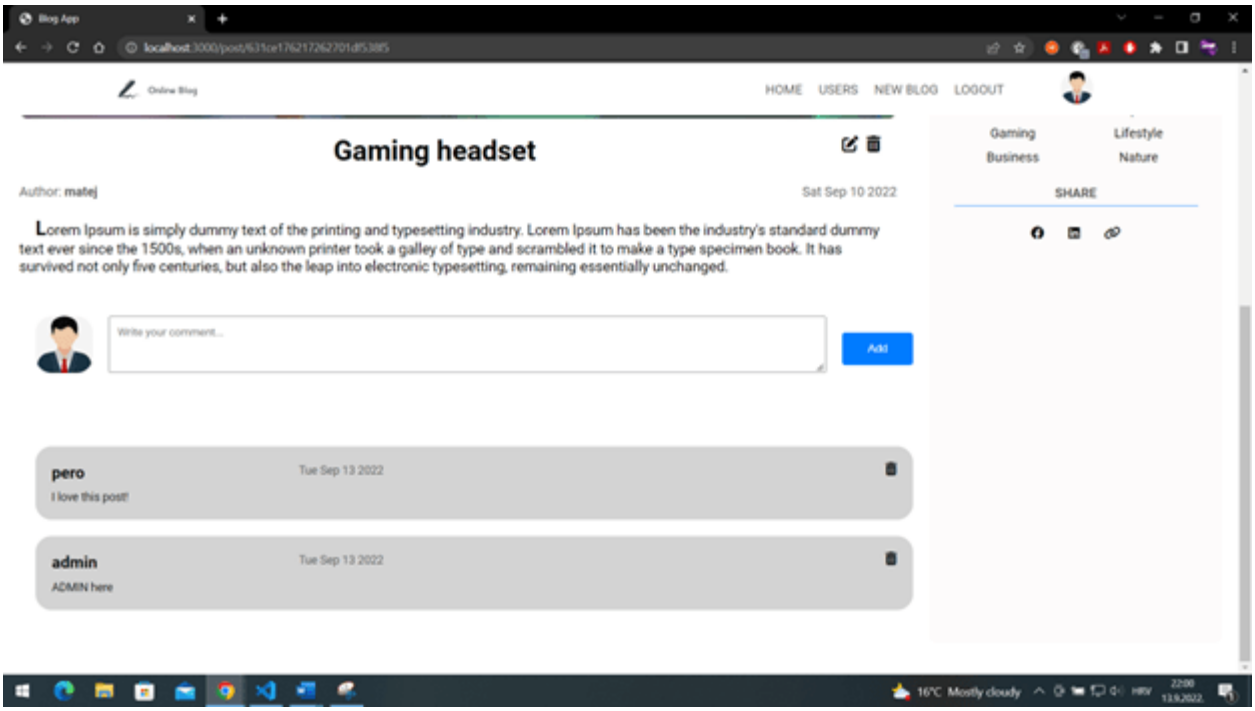
Slika 3.32. *Predložak korisnika*

Korisnici se dohvaćaju pomoću već spomenute GET metode, kojoj se od korisničkih podataka predaju korisnički ID-ovi. Primjer GET metode se nalazi na slici 3.33.

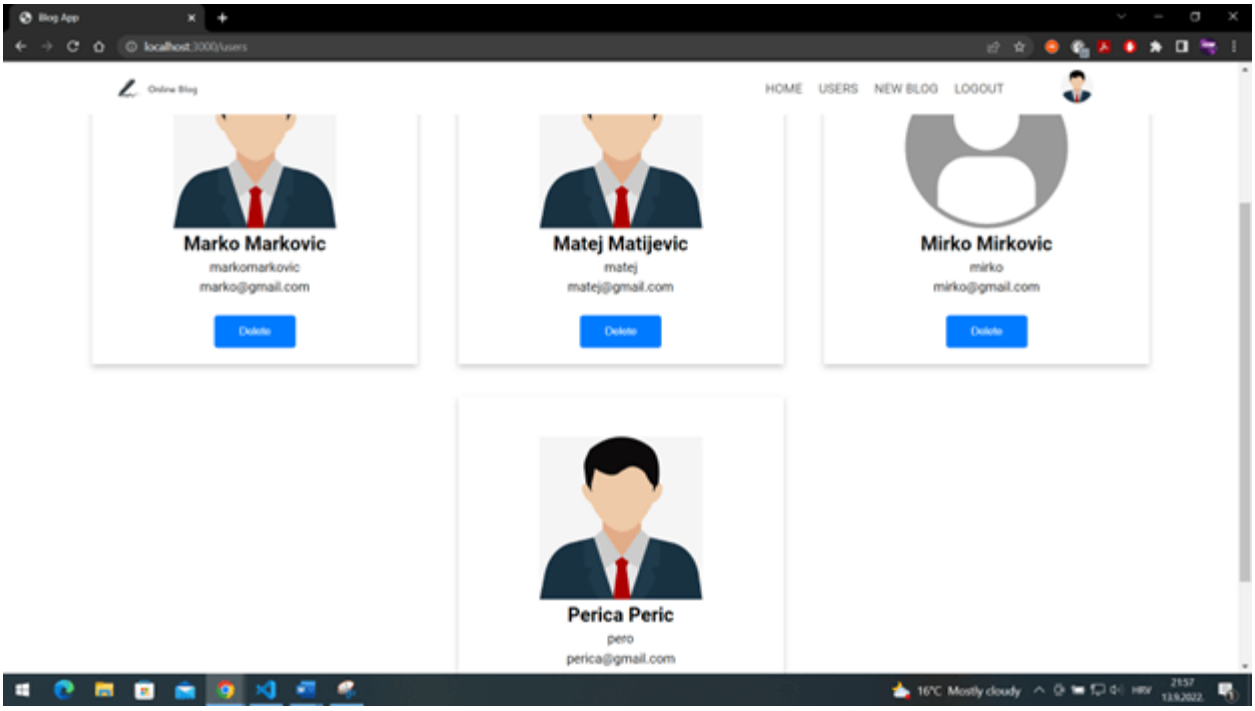
```
// GET USER
router.get("/:id", async (req, res) => {
  try {
    const user = await User.findById(req.params.id);

    // send all info except the password
    const { password, ...otherCredentials } = user._doc;
    res.status(200).json(otherCredentials);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Slika 3.33. *Dohvaćanje korisnika preko ID-a*

No osim toga, administrator može i brisati korisnike pomoću metode DELETE, kojoj se kao parametri predaju korisničko ime te korisnički ID. To je ista ona metoda koja se nalazi na slici 3.14. Primjer izgleda blog objave s komentarima s perspektive administratora se nalazi na slici 3.34., dok se kompletan izgled *Users* stranice nalazi na slici 3.35.



Slika 3.34. *Perspektiva administratora unutar blog objave i komentara*



Slika 3.35. *Prikaz svih korisnika*

4. ZAKLJUČAK

Ciljevi postavljeni u zadatku diplomskog rada su kompletno ostvareni. Izrađena je web aplikacija za pisanje *online* dnevnika, odnosno sustav koji omogućuje korisnicima objavu i komentiranje blog objave. Također, razvijen je i administracijski dio unutar kojeg administrator upravlja korisnicima i sadržajem. Za izradu aplikacije je bilo potrebno koristiti okvir React.

Za bazu podataka je korištena MongoDB baza podataka zajedno s njezinim sustavom. Za početak je bilo potrebno se prijaviti, odnosno registrirati unutar sustava MongoDB. Zatim je bilo potrebno kreirati klaster te povezati vlastiti projekt s bazom podataka. Povezati projekt s bazom podataka se može na više načina. U ovom slučaju je odabrana metoda u kojoj se baza podataka povezuje s projektom unutar samog programskog koda. Metoda se točnije primjenjuje na način da korisnik uzme privatni URL, ispuni ga sa svojim korisničkim imenom i zaporkom, te ga zalijepi unutar svog programskog koda. Ukoliko korisnik nije siguran u svoje podatke, uvijek ih može provjeriti unutar baze podataka pod izbornikom *Database Access*.

Što se tiče programskog koda, on je se odrađivao unutar Microsoft Visual Studio Code uređivača koda. Programski kod je podijeljen na *backend* i *frontend*. *Backend* se dijeli na modele i rute. Unutar modela su kreirani modeli s nazivima *User, Post, Comment* i *Category*. Za svaki od modela su kreirane sheme kojima su pridružena pojedina svojstva koja čine same modele. Unutar ruta su kreirane rute s nazivima *auth, categories, comments, posts* i *users*. Unutar rute *auth* se kreiraju metode za registraciju i prijavu korisnika. Za ostale rute se također kreiraju metode za kreiranje, dohvaćanje, uređivanje i brisanje – odnosno CRUD operacije. Osim toga, također je kreiran API za učitavanje slike unutar sustava aplikacije. Za odrađivanje svega navedenog, korištene su brojne biblioteke od kojih se ističu *express*, *mongoose*, *multer* i *bcrypt*. *Frontend* dio aplikacije je zapravo cijela React aplikacija koja je podijeljena na mnoštvo komponenti – od kojih neke čine cjelokupne stranice, dok su druge komponente koje grade te stranice. Unutar React aplikacije su bile definirane rute za svaku stranicu. React aplikacija je pretežito pisana u JSX jeziku, koji je po sintaksi vrlo sličan JavaScript programskom jeziku. Stil cijele aplikacije je odrađen pomoću SCSS-a, koji se pomoću proširenja *Live Sass Compiler* prevodio i minimizirao u obični CSS.

Aplikacija je uspješno realizirana te je potpuno funkcionalna. Korisnik može pristupiti blog objavama bez registracije i prijave, da bi ih mogao detaljnije pogledati, ipak bi se morao registrirati, odnosno prijaviti ako već sadrži korisnički račun. Korisnik nakon što se prijavi u

sustav, može kreirati vlastitu blog objavu te komentirati druge objave u sustavu. Korisnik također može uređivati i brisati samo vlastite objave, osim ako je to administrator koji ima pristup svemu. Što se tiče komentara, administrator može brisati sve komentare, dok obični korisnik može samo svoje. Osim toga, korisnik može mijenjati i poneke vlastite podatke zajedno sa profilnom slikom. Administrator ima pristup svim korisnicima koje može proizvoljno brisati. Sve u svemu, aplikacija je funkcionalna i radi bez ikakvih problema. Ono u čemu bi se možda mogla unaprijediti je to da se poboljša i poveća sigurnost korisnika, kako bi njihovi podaci ostali sigurni.

LITERATURA

[1] M. Warcholinski, 10 Famous React Apps: Examples (2022): Why do Internet giants choose React apps? [online], Brainhub, dostupno na: https://brainhub.eu/library/famous-apps-using-reactjs [11. rujna 2022.]

[2] N. Samuel, Top 7 MongoDB Alternative sin 2022: A Comprehensive List [online], Hevo Data, dostupno na: https://hevodata.com/learn/mongodb-alternatives/ [11. rujna 2022.]

[3] J. Clark, Top 10 ExpressJS Alternatives [online], Back4App, dostupno na: https://blog.back4app.com/expressjs-alternatives/ [11. rujna 2022]

[4] Model-view-viewmodel [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel [11. rujna 2022.]

[5] D. Karczewski, Best React.js Alternatives For Front-End Development In 2022 [online], Ideamotive, dostupno na: https://www.ideamotive.co/blog/best-react-js-alternatives-for-front-end [11. rujna 2022.]

[6] SPEC INDIA, Top 10 Node.js Alternatives Gaining Traction In 2022 [online], SPEC INDIA, dostupno na: https://www.spec-india.com/blog/node-js-alternatives [11. rujna 2022.]

[7] Matt, 10 Postman alternatives you should know about [online], Testfully, dostupno na: https://testfully.io/blog/top-5-postman-alternatives/ [11. rujna 2022.]

[8] Visual Studio Code Alternatives [online], AlternativeTo, dostupno na: https://alternativeto.net/software/visual-studio-code/ [11. rujna 2022.]

[9] MERN Stack Explained [online], MongoDB, dostupno na: https://www.mongodb.com/mern-stack [11. rujna 2022.]

[10] B. Botelho, J. Vaughan, MongoDB [online], TechTarget, dostupno na: https://www.techtarget.com/searchdatamanagement/definition/MongoDB [11. rujna 2022.]

[11] MongoDB [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/MongoDB [11. rujna 2022.]

[12] Express.js [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Express.js [11. rujna 2022.]

[13] What is a REST API? [online], Red Hat, dostupno na: https://www.redhat.com/en/topics/api/what-is-a-rest-api [11. rujna 2022.]

[14] What is Express JS In Node JS [online], Simplilearn, dostupno na: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js [11. rujna 2022.]

[15] React (JavaScript library) [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/React_(JavaScript_library) [11. rujna 2022.]

[16] React [online], React, dostupno na: https://reactjs.org/ [11. rujna 2022.]

[17] Hooks API Reference [online], React, dostupno na: https://reactjs.org/docs/hooks-reference.html [11. rujna 2022.]

[18] Node.js [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Node.js [11. rujna 2022.]

[19] Sass (stylesheet language), Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Sass_(stylesheet_language) [11. rujna 2022.]

[20] BEM, Get BEM, dostupno na: https://getbem.com/ [11. rujna 2022.]

[21] What is Postman?, Postman, dostupno na: https://www.postman.com/ [11. rujna 2022.]

[22] Visual Studio Code, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Visual_Studio_Code [11. rujan 2022.]

SAŽETAK

Glavni zadatak ovog diplomskog rada je napraviti web aplikaciju za pisanje online dnevnika koja će korisniku omogućiti pisanje, objavu, uređivanje, brisanje i komentiranje raznih objava. Kao baza podataka, koristio se MongoDB. Baza podataka se povezuje s projektom pomoću metode gdje se privatni URL s korisničkim podacima povezuje unutar programskog koda. Programski kod je pisan unutar Microsoft Visual Studio Code uređivača koda. Express.js se koristio kao Node.js web aplikacijski *okvir* za pomoć u upravljanju poslužiteljem i rutama. *Backend* dio se podijelio na modele i rute. Za svaki od modela su kreirane sheme kojima su pridružena određena svojstva. Rute su kreirane za svaki model zajedno s API-jima za kreiranje, dohvaćanje, uređivanje i brisanje podataka. *Frontend* dio je kompletno realiziran unutar React-a. React aplikacija je podijeljena stranice, koje su izgrađene od različitih komponenti. Cijela aplikacija je stilski uređena pomoću SCSS-a, koji je unutar aplikacije preveden i minimiziran u CSS. Aplikacija je pokrenuta i testirana, stoga se može potvrditi da je potpuno funkcionalna i da radi bez ikakvih problema.

Ključne riječi: Express.js, MongoDB, React, SCSS, web aplikacija

ABSTRACT

DEVELOPMENT OF A WEB APPLICATION FOR WRITING AN ONLINE DIARY

The main task of this graduate thesis is to create a web application for writing an online diary that will allow the user to write, publish, edit, delete, and comment on various posts. MongoDB was used as a database. The database is connected to the project using a method where a private URL with user data is connected inside the program code. The program code is written within the Microsoft Visual Studio Code editor. Express.js was used as a Node.js web application framework to help manage the server and routes. The backend is divided into models and routes. For each of the models, schemes were created with certain properties associated with them. Routes are created for each model together with APIs for creating, reading, updating, and deleting data. The frontend part is completely implemented within the React. A React application is divided into pages, which are built from different components. The entire application is styled using SCSS, which is compiled and minimized to CSS within the application. The application has been launched and tested, so it can be confirmed that it is fully functional and works without any problems.

Keywords: Express.js, MongoDB, React, SCSS, web application

ŽIVOTOPIS

Max Lončar je rođen 25.2.1998. godine u Vinkovcima, Hrvatska. Prebiva u Ivankovu gdje je pohađao Osnovnu školu August Cesarec Ivankovo u razdoblju od 2005. do 2013. godine. Nakon završetka osnovne škole upisuje školu Tehnička škola Ruđera Boškovića Vinkovci, smjer Tehnička gimnazija. Tijekom srednjoškolskog obrazovanja, 2016. godine je sudjelovao na Međužupanijskom natjecanju učenika u obrazovnom sektoru Elektrotehnika i računalstvo u disciplini Osnove elektrotehnike i mjerenja u elektrotehnici. Nakon polaganja mature 2017. godine, upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, točnije sveučilišni preddiplomski studij s nazivom Preddiplomski sveučilišni studij Računarstvo. Godine 2020. završava preddiplomski studij i upisuje sveučilišni diplomski studij s nazivom Diplomski sveučilišni studij Računarstvo, modul DRC – Programsko inženjerstvo. Poprilično dobro se služi Microsoft alatima te poznaje razne programske i skriptne jezike kao što je JavaScript. Od okvira, odnosno *frameworka*, upoznat je s React-om i Angular-om. Od prezentacijskih i stilskih opisnih jezika, upoznat je s HTML-om, Slim HTML-om, SCSS-om, SASS-om te CSS-om. Također se na fakultetu pobliže upoznao s programskim jezicima C, C++ i C#. Što se tiče Android aplikacija, upoznat je s radom Android Studio-a te s radom baza podataka. Također posjeduje znanje engleskog jezika u smislu čitanja, pisanja i govora te vozačku dozvolu B kategorije.

Max Lončar

PRILOZI

Github poveznica cjelokupnog projekta: https://github.com/maxloncar/online-blog-app

Na CD-u priloženom uz diplomski rad nalaze se:

- Diplomski rad u .docx i .pdf formatu
- Kompletan projekt s *backend*-om i *frontend*-om