Razvoj web aplikacije upotrebom MERN Stack razvojnog okvira

Hajdin, Adrian

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije upotrebom MERN Stack razvojnog okvira

Rijeka, rujan 2022.

Adrian Hajdin
0069088288

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije upotrebom MERN Stack razvojnog okvira

Mentor: Doc. dr. sc. Marko Gulić

Rijeka, rujan 2022.

Adrian Hajdin
0069088288

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 7. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Adrian Hajdin (0069088288)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **Razvoj web aplikacije upotrebom MERN Stack razvojnog okvira / Web application development using the MERN Stack framework**

Opis zadatka:

Razviti web aplikaciju i detaljno objasniti postupak izrade te aplikacije upotrebom MERN Stack razvojnog okvira koji se sastoji od 4 tehnologije (MongoDB, Express, React, Node.js) koje pružaju kompletnu podršku za cjeloviti razvoj neke web aplikacije. Objasniti postavljanje (instalaciju) MERN Stack razvojnog okruženja kroz detaljan opis instalacije svake od navedenih tehnologija. Razviti web aplikaciju za dijeljenje zanimljivih isječaka programskog koda i detaljno objasniti svaki korak izrade iste pomoću MERN Stack okvira. Također, implementirati cjelovitu autentifikaciju (JWT token) unutar spomenute web aplikacije. Za vizualni dizajn web aplikacije koristiti neki od React UI knjižica.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2022.

_____
Adrian Hajdin

Sadržaj

Popis slika

Poglavlje 1

Uvod

1.1 Opis aplikacije

Programerska zadaća je neprestano pisanje programskog koda, no nerijetko se određeni dijelovi već napisanog koda mogu ponovno iskoristiti na novim projektima. Ti isječci koda mogu biti različiti algoritmi, pretraživanja, dijelovi korisničkog sučelja, ili bilo koji drugi dijelovi koda koji bi programeru mogli ponovno zatrebati.

Koristeći aplikaciju napravljenu za potrebe ovog završnog rada, programeri mogu dijeliti isječke koda s drugim programerima, ili ih spremiti za kasniju upotrebu. Korisnik može zalijepiti kod u obrazac i podijeliti ga putem web adrese. Druga osoba može posjetiti tu vezu i jednim pritiskom miša kopirati cijeli isječak koda. Isječci koda su također sortirani po oznakama i po programskim jezicima, kako bi programeri mogli lakše pronaći one koji su im potrebni u određenom trenutku.

U svrhu izgradnje projekta, korišten je MERN razvojni okvir koji se koristi za lakšu i bržu implementaciju full-stack web aplikacija.

Poglavlje 2

Korištene tehnologije

Kako bi se izradila jedna web aplikacija, potrebno je znanje *frontend* i *backend* strana programskog razvoja. Zajedno *frontend* i *backend* omogućavaju programerima izradu *full-stack* aplikacija poput one koja je izrađena za potrebe pisanja ovog završnog rada. *Full-stack* programeri imaju znanja, vještine i upoznati su s metodologijama koje im omogućuju izradu bilo koje web aplikacije ispočetka pa sve do stavljanja u produkciju.

Frontend razvoj usmjeren je na izgled i strukturu web stranice ili aplikacije, dok je backend razvoj odgovoran za sve funkcije na poslužiteljskoj strani.

Zajedno, ove dvije komponente čine kompletan skup vještina programera.

2.1 *Frontend* - klijentska strana

Razvoj klijentske strane usmjeren je na izgled i vizualni dojam web stranice ili aplikacije. To uključuje izgled i dizajn te izradu korisničkog sučelja i planiranje korisničkog iskustva. Ključno je dobro razumjeti frontend pri izradi vizualno privlačne web stranice ili aplikacije. Neki od često korištenih frontend razvojnih alata uključuju HTML, CSS i JavaScript. HTML se koristi pri stvaranju strukture stranice, CSS se koristi za njezin stil, a JavaScript se koristi za dodavanje interaktivnosti.

2.2 *Backend* - poslužiteljska strana

Razvoj poslužiteljske strane odgovoran je za sve funkcije u pozadini web stranice ili aplikacije. To uključuje stvari poput upravljanja bazom podataka, administracije poslužitelja i razvoja API-ja. Važno je dobro razumjeti backend pri stvaranju funkcionalne web stranice ili aplikacije. Neki od najčešćih backend razvojnih alata uključuju PHP, Ruby on Rails i Node.js. PHP je popularan jezik za razvoj web aplikacija, Ruby on Rails je popularan framework, a Node.js je popularno JavaScript radno okruženje koje je korišteno prilikom izrade ovog završnog rada. Također postoji niz pozadinskih okvira i knjižnica koje mogu olakšati razvoj; Laravel i Express.

2.3 Prednosti full-stack razvoja

Full-stack razvoj koristan je tvrtkama, programerima i korisnicima. Full-stack razvoj posljednjih je godina sve popularniji zbog raznih prednosti i učinkovitosti u razvoju aplikacija ili web stranica. Koristan je i programerima i tvrtkama pri razumijevanju koraka implementacije i buduće optimizacije. Neke od prednosti full-stack razvoja su:

- **Profitabilnost** - Full-stack razvoj je profitabilan upravo zato što isti programer ili tim razvojnih programera radi na prednjem i stražnjem dijelu.
- **Brži razvoj i rješavanje problema** - Full-stack razvoj često zahtjeva grupni rad u kontekstu većeg tima koji pomaže efikasnoj isporuci cijelog projekta u zadanom vremenskom okviru. *Bugovi*, pogreške i nekompatibilnosti koje se javljaju tijekom razvoja aplikacija i web stranica, mogu se brzo riješiti.
- **Povećana učinkovitost** - Kao jedna od značajnih prednosti, full-stack razvoj omogućuje tvrtkama bržu i lakšu izradu i implementaciju aplikacija. Ova povećana učinkovitost štedi tvrtkama vrijeme i novac te im pomaže da budu konkurentnije na tržištu.
- **Poboljšana skalabilnost** - Još jedna prednost korištenja full-stack razvoja je poboljšanje skalabilnosti aplikacija. Skalabilnost je važna jer omogućuje poduzećima da se nose s povećanim opterećenjem ili prometom bez potrebe

za značajnim izmjenama svojih aplikacija. Poboljšanjem skalabilnosti svojih aplikacija, tvrtke mogu zadovoljiti potrebe svojih kupaca čak i dok njihovo poslovanje raste.

- **Fleksibilnost** - Programer ili firma koji razvijaju full-stack aplikacije nude visoku razinu fleksibilnosti jer uključuju klijentsku i poslužiteljsku stranu. Besprijekorna komunikacija i tijek rada između obje strane mogu smanjiti troškove i vrijeme razvoja.
- **Pojednostavljena implementacija, nadogradnja i održavanje** - Full-stack programeri mogu jednostavno nadograditi ili optimizirati aplikaciju i web stranicu budući da uključuju i klijentsku i poslužiteljsku stranu razvoja.
- **Više učenja i prilika** - Programeri u full-stack domeni mogu napredovati u svim sferama web razvoja. To im omogućuje neprestano učenje, te nadograđivanje i usavršavanje svojih vještina tijekom profesionalne karijere.

2.4 Korištene tehnologije - MERN Stack

Za izradu ovog završnog rada korišten je tehnološki skup (*stack*) JavaScript *ekosustava*. Tehnološki skup je skup okvira i alata koji se koriste za razvoj softverskog proizvoda. Ovaj skup okvira i alata vrlo je specifično odabran kako bi se sve korištene tehnologije međusobno nadopunjavale. Evo nekoliko primjera popularnih nizova tehnologija web razvoja koji se danas koriste:

- **MERN** (MongoDB, ExpressJS, ReactJS, NodeJS)
- **LAMP** (Linux, Apache, MySQL, PHP)
- **MEAN** (MongoDB, ExpressJS, AngularJS, NodeJS)

Cijeli projekt izgrađen je u MERN tehnološkom skupu. MERN je besplatni JavaScript softverski skup otvorenog koda za izradu dinamičkih web stranica i web aplikacija. MERN Stack je JavaScript *stack* koji se koristi za lakšu i bržu implementaciju full-stack web aplikacija. MERN Stack sastoji se od 4 tehnologije: MongoDB[1], Express JS [2] , React JS [3] i Node JS [4]. MongoDB je NoSQL baza podataka u kojoj je svaki zapis dokument koji sadrži parove ključeva i vrijednosti objektima. Express

JS je brz i minimalistički web okvir za Node.js. React JS je JavaScript knjižnica koja se koristi za izgradnju korisničkih sučelja. Node JS je JavaScript okruženje koje korisniku omogućuje pokretanje svog koda na poslužitelju (izvan web preglednika).

Osmišljen je kako bi razvojni proces bio brži i lakši. Svaka od ove 4 tehnologije ima ključnu fukncju u razvoju *full-stack* web aplikacija.

Glavna prednost korištenja MERN skupa je razvoj aplikacija pomoću jednog programskog jezika - JavaScript-a. Gore navedene četiri tehnologije, koje čine MERN skup tehnologija, temelje se na JavaScript programskog jeziku. MERN tvrtkama omogućuje jednostavan razvoj troslojne strukture (frontend, backend, baza podataka) u potpunosti koristeći JavaScript i JSON.

2.4.1 React.js - klijentska razina

React je JavaScript knjižnica koja se koristi za izgradnju korisničkih sučelja.

Najviša razina MERN stoga je React.js. React.js omogućuje izgradnju složenih sučelja putem jednostavnih komponenti te njihovo povezivanje u finalni HTML, CSS i JavaScript kod. Neke od prednosti React.js-a su sljedeće:

- **Virtualni DOM (Document Object Model)** – Virtualni DOM je kopija originalnog DOM-a. Svaka izmjena u web aplikaciji uzrokuje ponovno prikazivanje virtualnog DOM-a na cijelom korisničkom sučelju. Zatim se razlika između izvornog DOM-a i virtualnog DOM-a uspoređuje i promjene se događaju u skladu s izvornim DOM-om. Ovo čini React izuzetno efikasnim s obzirom da se ponovno prikazuju samo elementi koji su se doista promijenili.
- **JSX sintaksa** – označava JavaScript XML. To je HTML/XML JavaScript sintaksa koja se koristi u React-u. Olakšava i pojednostavljuje pisanje React komponenti jer se struktura (HTML) i logika (JS) nalaze na istom mjestu.
- **Komponente** – ReactJS podržava komponente. Komponente su manji blokovi korisničkog sučelja pri čemu svaka komponenta ima logiku i doprinosi cjelokupnom korisničkom sučelju. Komponente također omogućuju ponovnu upotrebu koda i čine cjelokupnu web aplikaciju lakšom za razumijevanje.

- **Efikasnost** – značajke poput Virtualnog DOM-a, JSX-a i komponenti čine React.js puno bržim i efikasnijim od ostalih postojećih tehnologija.

2.4.2 Node.js - poslužiteljska razina

Node.js je JavaScript okruženje koje korisniku omogućuje pokretanje svog koda na poslužitelju (izvan web preglednika). Node ima vlastiti upravitelj paketa tj. NPM (Node Package Manager) koji omogućuje programeru odabir između tisuća besplatnih paketa (*node* modula) za preuzimanje. Neke od prednosti Node.js-a su sljedeće:

- JavaScript radno okruženje otvorenog koda - omogućava da se JavaScript kod pokreće na serveru.
- Skalabilnost - omogućava skaliranje aplikacije *horizontalno* preko nekoliko servera ili *vertikalno* tako da se optimiziraju performance jednog servera
- Brzina – Koristi *Chrome-ov V8 JavaScript engine* [5] napisan u C++ programskom jeziku.

2.4.3 Express.js - *framework* za poslužiteljsku razinu

Sljedeća razina je Express.js koji radi unutar Node.js poslužitelja. Express.js sebe predstavlja kao "brz, samostalan, minimalistički web okvir za Node.js", i to je doista točno ono što jest. [2]

Express je okvir Node.js-a. Umjesto pisanja koda pomoću Node.js, može se jednostavnije pisati poslužiteljski kod pomoću Expressa. Express pomaže u dizajniranju API-ja (Application Programming Interface), koji su ključan koncept svake backend tehnologije.

Iz React.js aplikacije izradom HTTP zahtjeva (GET, POST, PUT, DELETE...) povezujemo se s Express.js funkcijama koje pokreću poslužiteljsku stranu aplikacije. Te funkcije putem "*promise-a*" ili obećanja, ažuriraju i vraćaju podatke s MongoDB baze podataka. Neke od prednosti korištenja Expressa su sljedeće:

- Učinkovit, brzo i skalabilan

- Ima najveću podršku u Node.js zajednici programera
- Express podrazumijeva višekratnu upotrebu koda sa svojim ugrađenim routerom.
- Robustan API

2.4.4 MongoDB - NoSQL baza podataka

MongoDB je NoSQL baza podataka u kojoj je svaki zapis dokument koji sadrži parove ključeva i vrijednosti koji su slični JSON (JavaScript Object Notation) objektima. MongoDB je fleksibilan te omogućuje stvaranje shema, baza podataka i tablica.

Za aplikacije koje pohranjuju bilo kakvu vrstu podataka (korisničke profile, sadržaj, komentare, događaje i slično), potrebna je baza podataka koja se može lako povezati s React-om, Express-om i Node-om, a to je MongoDB.

JSON dokumenti stvoreni u React.js prednjem dijelu šalju se putem Express.js poslužitelja, gdje se mogu obraditi i (pod pretpostavkom da su valjani) pohraniti izravno u MongoDB za kasnije dohvaćanje. Neke od prednosti MongoDB baze podataka su sljedeće:

- **Brzina** - Budući da je baza podataka orijentirana na dokumente, lako ih je indeksirati.
- **Korištenje JavaScript-a** - MongoDB koristi JavaScript što je najveća prednost.
- **Jednostavno postavljanje okruženja** - Jednostavno je postaviti MongoDB, posebno uz pomoć njihove usluge u oblaku pod nazivom MongoDB Atlas.
- **Fleksibilni model dokumenta** - MongoDB podržava model dokumenta (tablice, sheme i slično).

2.4.5 Zašto MERN

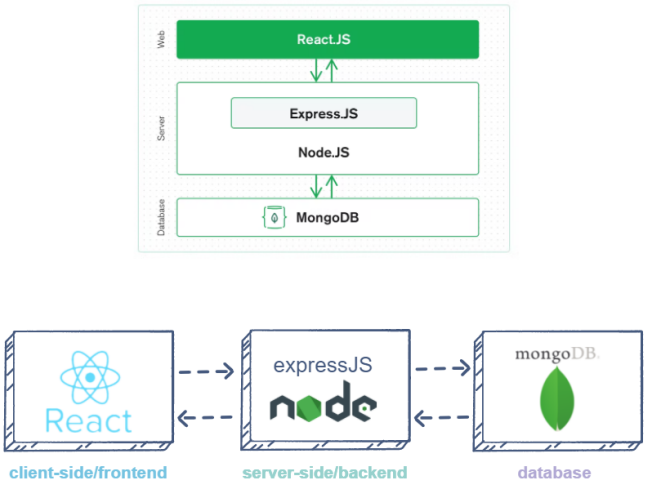
MongoDB je dizajniran za izvornu pohranu JSON podataka (tehnički koristi binarnu verziju JSON-a koja se zove BSON), a sve od sučelja naredbenog retka do jezika upita izgrađeno je na JSON-u i JavaScript-u. MongoDB iznimno dobro radi s Node.js i čini pohranu, manipuliranje i predstavljanje JSON podataka na svakoj razini aplikacije lakim. Express.js (koji radi na Node.js) i React.js čine JavaScript/JSON aplikaciju potpunim MERN stackom. Express.js je aplikacijski okvir na strani poslužitelja koji omotava HTTP zahtjeve i odgovore i olakšava mapiranje URL-ova na funkcije na strani poslužitelja. React.js je frontend JavaScript okvir za izgradnju interaktivnih korisničkih sučelja i komunikaciju s poslužiteljem. Prednosti MERN-a:

- **Bolje performanse** - Bolja izvedba odnosi se na brži odgovor između backend-a i frontend-a i baze podataka, što u konačnici poboljšava brzinu web stranice i pruža glatko korisničko iskustvo.
- **Jednostavan za prebacivanje između klijenta i poslužitelja** - MERN je vrlo jednostavan i brz jer je napisan u samo jednom jeziku. Također, vrlo je jednostavno prebacivati se između klijenta i poslužitelja prilikom pisanja koda.
- **Brza izrada aplikacija** - Sve web aplikacije i mobilne aplikacije stvorene pomoću MERN Stacka izrađuju se mnogo brže, što također pomaže u pružanju brže isporuke klijentima.

2.4.6 Troslojna struktura

Mnogo je dobrih razloga za korištenje MERN Stack-a. Na primjer, omogućuje stvaranje 3-slojne strukture koja uključuje frontend, backend i bazu podataka koristeći JavaScript i JSON.

Na slici 2.1 može se vidjeti kako korisnik komunicira s komponentama korisničkog sučelja **ReactJS** na prednjem dijelu aplikacije koji se nalazi u pregledniku. Ovo sučelje je povezano s poslužiteljem putem **Expressa** koji radi povrh NodeJS-a.



Slika 2.1 Troslojna arhitektura

Svaka interakcija koja uzrokuje zahtjev za izmjenom podataka šalje se Express poslužitelju **NodeJS-u**, koji preuzima podatke iz **MongoDB** baze podataka ako je potrebno i vraća podatke prednjem dijelu aplikacije, što se zatim prezentira korisniku.

2.5 Korištene strukture i metodologije

Prilikom programiranja ovog projekta korišteno je nekoliko modernih i općeprihvaćenih metodologija, struktura i skupova pravila često korištenih prilikom programiranja MERN aplikacija; neke od njih su CRUD metodologija i REST arhitektura.

2.5.1 CRUD metodologija

CRUD je akronim za CREATE, READ, UPDATE, i DELETE. Ove četiri naredbe baze podataka temelj su CRUD-a:

- Funkcija CREATE dodaje jedan ili više unosa i ekvivalentna je funkciji Insert u SQL-u.
- Funkcija READ dohvaća podatke na temelju različitih kriterija i ekvivalentna je funkciji Select u SQL-u.

- Funkcija UPDATE mijenja ili modificira zapise.
- Funkcija DELETE uklanja jedan ili više navedenih unosa.

CRUD metodologija čini najrasprostranjeniji tip aplikacija današnjice. Gotovo svaka aplikacija mora stvarati, čitati, ažurirati i brisati dokumente. Na Twitteru, Facebooku, Instagramu i ostalim društvenim mrežama korisnici stvaraju, čitaju, ažuriraju i brišu objave; kod web trgovine to su proizvodi, a kod chat aplikacije to su poruke. Jasno je zaključiti da su koncepti CRUD-a sveprisutni.

2.5.2 REST API arhitektura

Pozadina Platforme za dijeljenje isječaka koda, osim CRUD-a, također koristi i REST API arhitekturu, koji odlično pašu zajedno.

Akronim CRUD u REST okruženju često odgovara idućim HTTP metodama:

- POST -> CREATE — Stvara novi zapis u bazi podataka.
- GET -> READ — Čita informacije iz baze podataka.
- PUT/PATCH -> UPDATE — Ažurira zapise u bazi podataka.
- DELETE -> DELETE — Uklanja zapise iz baze podataka.

Ovaj projekt je potpuna REST CRUD aplikacija što znači da ima mogućnost stvaranja, čitanja, ažuriranja i brisanja isječka koda.

Na slici 2.2 mogu se vidjeti podaci zahtjeva za stvaranje isječaka koda

```
POST https://code-sharing-platform.vercel.app/snippet
```

```
{
  "title" : "New post",
  "description" : "This is a new post",
  "code": "console.log('Hello World')",
  "language": "js",
  "postedBy": "1234567898765433"
}
```

Slika 2.2 Podaci zahtjeva za stvaranje isječka koda

Da bi se taj zahtjev poslao na poslužiteljsku stranu koristi se mali paket pod nazivom Axios.

Na slici 2.3 mogu se primjetiti kratke funkcije koje su stvorene kako bi ih se jednostavno moglo pozvati iz bilo koje komponente, bez da je potrebno svaki put ponovno pisati cijeli http zahtjev.

```
const API = axios.create({baseURL: "https://backend-code-sharing-platform.vercel.app"});

export const fetchSnippets = () => API.get("/snippet");
export const fetchSnippet = (id) => API.get(`/snippet/${id}`);
export const createSnippet = (newSnippet) => API.post(`/snippet`, newSnippet);
export const updateSnippet = (id, updatedSnippet) => API.patch(`/snippet/${id}`, updatedSni
export const deleteSnippet = (id) => API.delete(`/snippet/${id}`);
```

Slika 2.3 Ponovno iskoristive funkcije

Poglavlje 3

Backend - poslužiteljska strana

Početak rada s MERN razvojnim okvirom kreće sljedećim instalacijama:

- Instaliranje Node.js
- Instaliranje programa za pisanje koda, npr. Visual Studio Code. [6]

3.1 Postavljanje projekta

MERN omogućuje stvaranje potpunih programskih rješenja. Za ovaj projekt izrađuje se stražnji i prednji dio. Prednji dio bit će implementiran s React-om, a stražnji će biti implementiran s MongoDB, Node i Express tehnologijama.

Potrebno je stvoriti praznu mapu naziva code-sharing-platform*. Ova će mapa sadržavati sve datoteke nakon što se izradi novi projekt. Zatim se u njoj stvara React aplikacija kako slijedi:

```
mkdir code-sharing-platform
cd code-sharing-platform
npx create-react-app client
```

Zatim je potrebno stvoriti mapu za poslužiteljsku stranu i nazvati je "server".

```
mkdir server
```

Potrebno je prebaciti se u mapu server koja je prethodno kreirana i kreirati poslužitelj. Zatim je potrebno inicijalizirati package.json koristeći npm init komandu:

```
cd server
npm init -y
```

Također je potrebno instalirati sljedeće pakete.

```
npm install express mongoose cors dotenv morgan helmet
```

Gornja naredba koristi nekoliko ključnih riječi:

- mongoose naredba instalira MongoDB alat za modeliranje objekata koji omogućuje Node.js aplikacijama povezivanje s bazom podataka i rad s podacima.
- express instalira web okvir za Node.js.
- cors instalira paket Node.js koji omogućuje dijeljenje resursa s više izvora.
- dotenv instalira modul koji učitava varijable okruženja iz .env datoteke u datoteku process.env. To omogućuje odvajanje konfiguracijskih datoteka od koda.
- morgan instalira modul koji bilježi HTTP zahtjeve upućene poslužitelju
- helmet instalira modul koji osigurava Express aplikacije postavljanjem različitih HTTP zaglavlja.

Potrebno je provjeriti instalirane pakete pomoću datoteke package.json.

Ukoliko su ovisnosti uspješno instalirane, potrebno je stvoriti datoteku pod nazivom index.js (Kodni isječak 3.1.).

Slika 3.1 Kodni isječak - postavljanje backenda

```
require("dotenv") config();
const express = require("express");
const morgan = require("morgan");
const helmet = require("helmet");
const cors = require("cors");

const app = express();

// Middlewares
app.use(helmet());
app.use(morgan("dev"));
app.use(cors());
app.use(express.json());

app.get("/", (req, res) => {
  res.json({
    message: "Hello World,
  });
});

const port = process.env.PORT || 1337;
app.listen(port, () => console.log(`Listening: http://localhost:${port}/`));
```

3.2 Povezivanje na MongoDB Atlas

Idući korak je povezivanje s bazom podataka. Kao bazu podataka koristi se MongoDB Atlas. MongoDB Atlas je usluga baze podataka temeljena na oblaku koja pruža robusnu sigurnost i pouzdanost podataka.

Nakon stvaranja računa na MongoDB Atlas platformi potrebno je stvoriti *config.env* datoteku u "server" mapi. Tamo je potrebno dodijeliti vrijednost varijabli ‘ATLAS URI‘. Nakon toga, potrebno je dodati podatke kao što su ‘<username>‘ i ‘<password>‘ koji odgovaraju korisničkom imenu i lozinki baze podataka. Implementacija je prikazana u kodnom isječku 3.2.

Slika 3.2 Kodni isječak - postavljanje konfiguracije

```
`mern/server/config.env`
MONGODB_URI=mongodb+srv://<username>:<password>
@sandbox.jadwj.mongodb.net/employees?retryWrites=true&w=majority
PORT=5000
```

Zatim, u index.js, na dnu datoteke potrebno je dodati sljedeći kod za povezivanje s bazom podataka (kodni isječak 3.3.)

Slika 3.3 Kodni isječak - povezivanje s bazom podataka

```
const mongoose = require("mongoose");

// Database Connectivity
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

mongoose.connection.on("connected", () => {
  console.log(" Connected to database");
});

mongoose.connection.on("error", (error) => {
  console.log(" Error while connecting to database ", error);
});
```

3.3 Krajnje točke API-ja poslužitelja

Nakon što su baza podataka i poslužitelj postavljeni potrebno je dodati krajnje točke poslužiteljskog API-ja. Treba otvoriti mapu ‘routes‘ i dodati snippet.js datoteku kako slijedi:

```
cd ../server
mkdir routes
touch routes/snippet.js
```

U kodnom isječku 3.4., koristi se funkcija ‘express.Router()‘ koja služi za stvaranje novog objekta usmjerivača. Također se definiraju različite rute koje koriste *GET, POST, PATCH i DELETE* metode.

Slika 3.4 Kodni isječak - definiranje ruta

```
const express = require("express");

const { getSnippets,
  getSnippet,
  createSnippet,
  updateSnippet,
  deleteSnippet,
  profileSnippets
} = require("../controllers/snippet");

const router = express.Router();

// CRUD OPERACIJE
router.get("/", getSnippets);
router.get("/:id", getSnippet);
router.post("/", createSnippet);
router.patch("/:id", updateSnippet);
router.delete("/:id", deleteSnippet);

router.get("/user/:id", profileSnippets);

module.exports = router;
```

Projekt slijedi MVC arhitekturu za strukturu mapa i datoteka.

Stoga je potrebno izraditi mapu pod nazivom controllers. Kontroleri su dio koji se brine za obradu zahtjeva klijenta te HTTP odgovorom vraća JSON ili HTML kod.

Potrebno je kreirati mapa controllers i dodati snippet.js kako slijedi:

```
cd /server
mkdir controllers
touch controllers/snippet.js
```

Datoteka controllers/snippet.js za sada sadrži sljedeće retke koda (kodni isječak 3.5.). Kasnije je potrebno dodati logiku.

Slika 3.5 Kodni isječak - definiranje funkcija

```
const mongoose = require("mongoose");

const getSnippets = async (req, res, next) => {};

const getSnippet = async (req, res, next) => {};

const createSnippet = async (req, res, next) => {};

const updateSnippet = async (req, res, next) => {};

const deleteSnippet = async (req, res, next) => {};

const profileSnippets = async (req, res, next) => {};

module.exports = {
  getSnippets,
  getSnippet,
  createSnippet,
  updateSnippet,
  deleteSnippet,
  profileSnippets
};
```

Pri pokretanju aplikacije, pokazati će se sljedeća poruku na terminalu.

```
> node index.js
```

```
Listening: http://localhost:5000/`
| Connected to database
```

3.4 *Middleware* za rukovanje pogreškama

Potrebno je stvoriti mapu middlewares i dodati errorHandler.js kako slijedi:

```
cd /server
mkdir middlewares
touch middlewares/errorHandler.js
```

Implementacija je vidljiva u kodnom isječku 3.6.

Slika 3.6 Kodni isječak - stvaranje *error handlera*

'code-sharing-platform/server/controllers/snippet.js''

```
const notFound = (req, res, next) => {
  const error = new Error(`Not Found  ${req.originalUrl}`);
  res.status(404);
  next(error);
};
const errorHandler = (err, req, res, next) => {
  const statusCode = res.statusCode ? res.statusCode : 500;
  res.status(statusCode);

  res.json({
    error: true,
    message: err.message,
    stack: process.env.NODE_ENV === "production" ? "" : err.stack,
  });
};

module.exports = { notFound, errorHandler };
```

Middleware su jednostavne funkcije za rukovanje pogreškama. Važno je osigurati

da Express prepozna sve pogreške koje se javljaju tijekom pokretanja rukovatelja rutama i međuprograma.

Definiranje funkcije međuprograma za rukovanje pogreškama izvodi se na isti način kao i druge funkcije međuprograma, osim što funkcije za rukovanje pogreškama imaju četiri argumenta umjesto tri: '(err, req, res, next)'

Middeware za rukovanje pogreškama definira se kao posljednji poziv, nakon drugih 'app.use()' poziva; Dakle, potrebno je dodati idući kod u index.js datoteku:

```
app.use(notFound);
app.use(errorHandler);
```

3.5 MongoDB shema baze podataka

Kreira se mapa models i dodaje se snippet.js u nju. Počninje se sa stvaranjem datoteke u kojoj se definira shema. Potrebno je stvoriti novu mapu pod nazivom modeli i datoteku unutar nje pod nazivom snippet.js:

```
cd /server
mkdir models
touch models/snippet.js
```

U kodnom isječku 3.7. može se vidjeti stvaranje modela isječka koda.

Postoje tri atributa – naslov, opis, šifra itd. Naziv i ostali atributi su tipa String i obavezni su. Većina ovih vrsta podataka bit će poznata programerima. Ako je vrsta podataka nepoznata, potrebno je definiranje vrste podataka koju se pohranjuje. I na kraju, stvara se model sheme. Prvi parametar 'mongoose.model' je naziv zbirke koja će sadržavati dokumente. Drugi parametar je shema koja je definirana ranije. Sada postoji model, samo ga se treba izvesti kako bi se mogao koristiti negdje drugdje. Sada je ovaj model spreman za uvoz i korištenje za izradu podataka prema shemi u zbirci 'snippets'. Ovo je uobičajeni razvojni obrazac kada se koriste NodeJS i MongoDB. Object Data Modeling (ODM) knjižnica kao što je mongoose čini upravljanje podacima intuitivnijim.

Slika 3.7 Kodni isječak - stvaranje modela isječka koda

‘code-sharing-platform/server/model/snippet.js‘

```
const mongoose = require("mongoose");
const { ObjectId } = mongoose.Schema.Types;

const snippetSchema = mongoose.Schema({
    title: { type: String, required: true },
    description: { type: String, required: true },
    code: { type: String, required: true },
    language: { type: String, required: true },
    postedBy: { type: ObjectId, ref: "User", required: true },
    tags: [String],
}, { timestamps: true });

const Snippet = mongoose.model("Snippet", snippetSchema);

module.exports = Snippet;
```

3.6 CRUD Funkcionalnost

CRUD je akronim koji označava CREATE (STVORI), READ (ČITAJ), UPDATE (AŽURIRAJ) i DELETE (IZBRIŠI) funkcionalnosti. Ove četiri naredbe baze podataka temelj su CRUD-a.

Rute su već definirane, a definirane su i funkcije kontrolera. Sada je samo potrebno napisati logiku u pojedinu funkciju.

U isječku koda 3.8. može se vidjeti funkcija za dohvaćanje svih isječaka koda. U isječku koda 3.9. može se vidjeti funkcija za dohvaćanje jednog isječka koda.

U isječku koda 3.10. može se vidjeti funkcija za stvaranje jednog isječka koda. U isječku koda 3.11. može se vidjeti funkcija za ažuriranje jednog isječka koda.

U isječku koda 3.12. može se vidjeti funkcija za brisanje jednog isječka koda. U isječku koda 3.13. može se vidjeti funkcija za dohvaćanje isječaka koda pojedinog korisnika.

Slika 3.8 Kodni isječak - dohvaćanje svih isječaka koda

```
const getSnippets = async (req, res, next) => {
  try {
    Snippet.find()
      .populate("postedBy")
      .exec(function (err, snippets) {
        if (err) console.error(err);

        res.json({ error: false, snippets });
      });
  } catch (error) {
    next(error);
  }
};
```

Slika 3.9 Kodni isječak - dohvaćanje jednog isječka koda

```
const getSnippet = async (req, res, next) => {
  const { id } = req.params;

  try {
    if (!mongoose.Types.ObjectId.isValid(id)) {
      res.status(404);
      throw new Error(` No snippet with id: ${id}`);
    }

    Snippet.findById(id)
      .populate("postedBy")
      .exec(function (err, snippet) {
        if (err) console.error(err);

        if (!snippet)
          return res.json({
            error: true,
            message: ` No snippet with id: ${id}`,
          });

        res.json({ error: false, snippet });
      });
  } catch (error) {
    next(error);
  }
};
```

Slika 3.10 Kodni isječak - stvaranje jednog isječka koda

```
const createSnippet = async (req, res, next) => {
  const { title, description, code, language, postedBy } = req.body;

  const tags = req.body.tags.split(",");

  try {
    const newSnippet = await Snippet.create({
      title,
      description,
      code,
      language,
      postedBy,
      tags,
    });

    res.status(200).json({ error: false, post: newSnippet });
  } catch (error) {
    next(error);
  }
};
```

Slika 3.11 Kodni isječak - ažuriranje jednog isječka koda

```
const updateSnippet = async (req, res, next) => {
  const { id } = req.params;
  const { title, description, code, language, postedBy, tags } = req.body;

  try {
    if (!mongoose.Types.ObjectId.isValid(id)) {
      res.status(404);
      throw new Error(` No snippet with id: ${id}`);
    }

    const snippet = await Snippet.findById(id);
    if (!snippet) {
      res.status(404);
      throw new Error(` No snippet with id: ${id}`);
    }

    const updatedSnippet = {
      title,
      description,
      code,
      language,
      postedBy,
      tags,
      _id: id,
    };

    const newSnippet = await Snippet.findByIdAndUpdate(id, updatedSnippet, {
      new: true,
    });

    res.json({ error: false, newSnippet });
  } catch (error) {
    next(error);
  }
};
```

Slika 3.12 Kodni isječak - brisanje jednog isječka koda

```
const deleteSnippet = async (req, res, next) => {
  const { id } = req.params;

  try {
    if (!mongoose.Types.ObjectId.isValid(id)) {
      res.status(404);
      throw new Error(` No snippet with id: ${id}`);
    }

    const snippet = await Snippet.findById(id);

    if (!snippet) {
      res.status(404);
      throw new Error(` No snippet with id: ${id}`);
    }

    await Snippet.findByIdAndDelete(id);

    res.json({ error: false, message: " Snippet deleted successfully!" });
  } catch (error) {
    next(error);
  }
};
```

Slika 3.13 Kodni isječak - dohvaćanje isječaka koda pojedinog korisnika

```
const profileSnippets = async (req, res, next) => {
  const { id } = req.params;

  try {
    if (!mongoose.Types.ObjectId.isValid(id)) {
      res.status(404);
      throw new Error(` No user with id: ${id}`);
    }

    const user = await User.findById(id);

    if (!user) {
      res.status(404);
      throw new Error(` No user with id: ${id}`);
    }

    Snippet.find({ postedBy: id })
      .populate("postedBy")
      .exec(function (err, snippets) {
        if (err) console.error(err);

        res.json({ error: false, snippets, user });
      });
  } catch (error) {
    next(error);
  }
};
```

3.7 JWT Autentifikacija

3.7.1 Rute za autentifikaciju

Započinje se stvaranjem datoteke u mapi ruta pod nazivom auth.js pomoću sljedećih naredbi:

```
cd ../server/routes
touch routes/auth.js
```

Datoteka routes/auth.js također će sadržavati sljedeće retke koda (Kodni isječak 3.14.):

Slika 3.14 Kodni isječak - postavljanje autentifikacije

‘code-sharing-platform/server/routes/auth.js‘

```
const express = require("express");

const { signup, login } = require("../controllers/auth");

const router = express.Router();

router.post("/signup", signup);
router.post("/login", login);

module.exports = router;
```

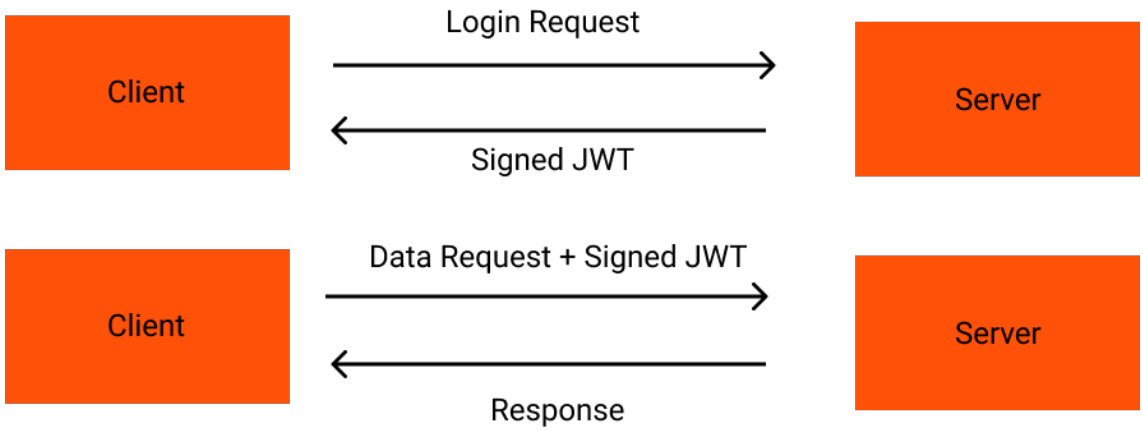
3.7.2 JWT - JSON web token

Za potrebe provjere autentičnosti, projekt koristi JSON web token provjeru autentičnosti.

JSON Web Token je otvoreni standard za siguran prijenos podataka unutar strana pomoću JSON objekta. JWT se koristi za mehanizme provjere autentičnosti bez stanja za korisnike i pružatelje usluga, što znači održavanje sesije na strani klijenta

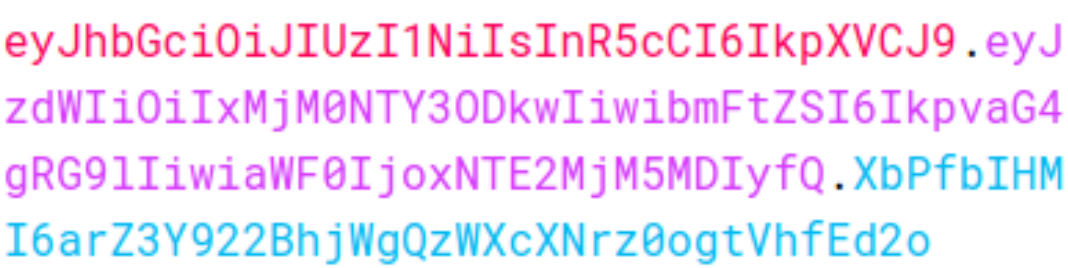
umjesto pohranjivanja sesija na poslužitelju. Ovdje se implementira JWT sustav autentifikacije u Node.js i Express.js.

Na slici 3.15. može se vidjeti dijagram koji objašnjava rad JWT-a.



Slika 3.15 Rad JWT-a

Struktura JWT-a može se vidjeti na slici 3.16.



Slika 3.16 Struktura JWT-a

Kao što slika prikazuje, postoje tri odjeljka ovog JWT-a, svaki odvojen točkom.

Prvi odjeljak JWT-a je zaglavlje, koje je Base64 kodirani niz. Ako je završeno dekodiranje, zaglavlje izgleda ovako:

```
{
```

```
  "alg": "HS256",
  "tip": "JWT"
}
```

Odjeljak zaglavlja sadrži algoritam raspršivanja koji je korišten za generiranje znaka i vrste tokena. Drugi odjeljak su korisni podaci koje sadrži JSON objekt koji je poslan natrag korisniku. Budući da je kodiran samo Base64, svatko ga može lako dekodirati. Preporuča se da se u JWT-ove ne uključuju osjetljivi podaci, poput lozinki ili osobnih podataka. Ukratko, JWT-ovi se koriste kao siguran način za autentifikaciju korisnika i dijeljenje informacija. Tipično, privatni ključ, ili tajni, koristi izdavatelj za potpisivanje JWT-a. Primatelj JWT-a će provjeriti potpis kako bi osigurao da token nije promijenjen nakon što ga je potpisao izdavatelj. Stvaranje *tokena* može se vidjeti u sljedećem redu:

```
const token = jwt.sign({
        id: existingUser._id },
    process.env.ACCESS_TOKEN_SECRET, { expiresIn: "1h" }
 );
```

Prilikom slanja obrasca postoje neki osjetljivi podaci (poput lozinki) koji ne smiju biti vidljivi nikome, čak ni administratoru baze podataka. Kako osjetljivi podaci ne bi postali vidljivi svima, koristi se "bcryptjs". Ovaj modul omogućuje pohranjivanje lozinki kao raspršenih lozinki umjesto otvorenog teksta. Bcrypt.js je funkcija raspršivanja zaporke.

Raspršivanje lozinke je vrlo jednostavno, prvi argument u metodi ‘bcrypt.hash()‘ je lozinka koja se dobiva od ‘req.body‘. Drugi argument je broj krugova rasprišivanja zaporke. Raspršivanje lozinke može se vidjeti u sljedećem redu:

```
const hashedPassword = await bcrypt.hash(password, 10);
```

Kada se korisnik prijavi u aplikaciju, API će provjeriti postoji li e-pošta u bazi podataka uz pomoć metode userSchema.findOne() . Zatim se potvrđuje pohranjena zaporka uz pomoć metode bcrypt.compare().

```
const isPasswordCorrect =
await bcrypt.compare(password, existingUser.password);
```

3.8 Provjera valjanosti podataka

Izrada pozadinskih API-ja dolazi s mnogim poteškoćama, a jedna od njih je provjera valjanosti korisničkog unosa. Važno je dodati dodatni sloj provjere dolaznih podataka jer se ne smije osloniti samo na unos korisnika.

Postoji mnogo načina provođenja provjere valjanosti unosa u Node.js, ali za ovaj projekt koristi se Joi [7].

Joi je najpoznatiji, učinkovit i široko korišten paket za opise i provjeru shema objekata. Joi omogućuje programerima da izgrade JavaScript nacrte i osiguraju da aplikacija prihvaća točno oblikovane podatke.

Joi paket se preuzima pokretanjem naredbe:

```
npm install joi
```

3.9 Stvaranje mape za pomoćne funkcije

Kreira se mapa utils i u nju dodaje validation.js pomoću sljedećih naredbi:

```
cd /server
mkdir utils
touch utils/validation.js
```

U datoteci ‘validation.js‘ potreban je joi paket. Za izradu provjere valjanosti objekta sheme koristi se funkcija object() od joi. Potrebno je provjeriti različita svojstva. Svako svojstvo treba funkciju ulančavanja joi za provjeru.

Slijede neke od joi funkcija koje se koriste za provjeru valjanosti svojstva. ‘Joi.string‘ provjerava jesu li sva svojstva *stringovi*; ‘Joi.trim‘ zahtijeva da vrijednost *stringa* ne sadrži razmake prije ili iza; ‘Joi.required‘ zahtjeva da vrijednost bude prisutna;

'Joi.email' potvrđuje da polje sadrži valjanu adresu e-pošte. Implementacija navedenog u kodu može se vidjeti u kodnom isječku 3.17.

Slika 3.17 Kodni isječak - postavljanje provjera

```
const Joi = require("joi");

const signupValidation = (data) => {
  const schema = Joi.object({
    name: Joi.string().required().min(3).max(50),
    email: Joi.string().required().email().max(320),
    password: Joi.string().required().min(6),
  });

  return schema.validate(data);
};

const loginValidation = (data) => {
  const schema = Joi.object({
    email: Joi.string().required().email().max(320),
    password: Joi.string().required().min(6),
  });

  return schema.validate(data);
};

module.exports = {
  signupValidation,
  loginValidation,
};
```

3.10 Auth Middleware

Samo autentificirani korisnici trebaju imati mogućnost stvaranja, ažuriranja i brisanja. Potrebno je stvoriti međuprogram za autentifikaciju, što čini nova datoteka 'auth.js' unutar mape 'middlewares' (kodni isječak 3.18.):

Slika 3.18 Kodni isječak - postavljanje *middlewarea* za autentifikaciju

code-sharing-platform/server/middlewares/auth.js

```
const jwt = require("jsonwebtoken");

const auth = async (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];

  if (!token)
    return res.status(401).json({ error: " Access Denied - Please Login" });

  try {
    const validToken = jwt.verify(token, process.env.ACCESS_TOKEN_SECRET);

    req.userId = validToken.id;

    next();
  } catch (error) {
    res.status(400).json({ error: " Invalid token" });
  }
};

module.exports = auth;
```

Middleware čita vrijednost zaglavlja autorizacije. Budući da zaglavlje ‘autorizacija‘ ima vrijednost u formatu ‘Nositelj [JWT TOKEN]‘, potrebno je podijeliti vrijednost razmakom i odvojiti token.

Zatim se potvrdi token pomoću JWT-a. Nakon provjere, objekt ‘korisnik‘ prilaže se zahtjevu. U protivnom se klijentu šalje poruka o pogrešci.

Konfiguracija je vidljiva u idućem kodnom isječku 3.19.

Implemetacija poslužiteljske strane sada je završena i moguće je prijeći na klijentsku stranu.

Slika 3.19 Kodni isječak - postavljanje ruta

```
const auth = require("../middlewares/auth");

router.get("/", getSnippets);
router.get("/:id", getSnippet);
router.post("/", auth, createSnippet);
router.patch("/:id", auth, updateSnippet);
router.delete("/:id", auth, deleteSnippet);
```

Poglavlje 4

Frontend - klijentska strana

Postavljanje React aplikacije vrši se pomoću sljedeće naredbe

```
cd ..
npx create-react-app client
```

Kada je aplikacija postavljena, potrebno je otvoriti mapu "client" i provjeriti kod React aplikacije.

Prije pokretanja aplikacije, potrebno je instalirati dva dodatna paketa. Potrebno je otvoriti novi terminal, navigirati do direktorija "client" i instalirati ‘react-router-dom‘.

```
npm install react-router-dom
```

Paket ‘react-router-dom‘ instalira komponente potrebne za preusmjeravanje unutar web aplikacije.

4.1 Postavljanje React usmjerivača

Idući korak je isprazniti src folder i dodati dvije nove datoteke u njega: index.js i App.js pomoću sljedećih naredbi:

```
rm src/**/*
touch src/index.js src/App.js
```

Unutar src/index.js potrebno je dodati isječak koda 4.1.:

Slika 4.1 Kodni isječak - postavljanje react-router paketa

```
import React from "react";
import ReactDOM from "react-dom/client";

import App from "./App";
import "./index.css";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Sada je potrebno postaviti react-router (Kodni isječak 4.2.)

Korišten je *BrowserRouter* kako bi korisničko sučelje bilo sinkronizirano s URL-om. *BrowserRouter* pomaže u prijelazima tijekom prebacivanja između komponenti. Ponovno će osvježiti samo komponentu koju je potrebno promijeniti umjesto ponovnog učitavanja cijele stranice.

Slika 4.2 Kodni isječak - postavljanje react-router paketa

```
import { BrowserRouter, Routes, Route } from "react-router-dom";

const App = () => (
  <BrowserRouter>
      <Routes>
        <Route path="/" element={<h1>Hello World</h1>} />
      </Routes>
  </BrowserRouter>
);

export default App;
```

4.2 Stvaranje komponenti

Nakon dodavanja koda u datoteke index.js i App.jsx, potrebno je izraditi mapu components unutar mape src. Za svaku izrađenu komponentu potrebno je dodati novu .jsx datoteku. U ovom slučaju potrebno je dodati Navbar.jsx i PostCard.js pomoću sljedećih naredbi:

```
mkdir src/components
cd src/components
touch Navbar.jsx PostCard.jsx
```

4.2.1 Navigacijska komponenta

Za stvaranje navigacijske komponente potrebno je koristiti NavLink komponentu iz react-router-dom paketa koja je korisna za prikaz trenutno aktivne stranice (kodni isječak 4.3.).

4.2.2 PostCard Komponenta

Kodni isječak 4.4. prikazuje stvaranje komponente isječka koda.

Slika 4.3 Kodni isječak - stvaranje navigacijske komponente

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

import { logo, logout as logoutIcon } from "../assets";

const generateActiveStyle = ({ isActive }) => isActive ? { color: "#FFFFFF" } : {

const Navbar = () => {

  return (
    <header>
      <Link to="/">
        <img src={logo} alt="Logo" />
      </Link>

      <nav>
        <NavLink to="/" style={generateActiveStyle} >
          Home
        </NavLink>
        <NavLink to="/login" style={generateActiveStyle} >
          Login
        </NavLink>
        <NavLink to="/signup" style={generateActiveStyle} >
          Sign Up
        </NavLink>
      </nav>
    </header>
  );
};

export default Navbar;
```

4.2.3 Komponente za stvaranje novih isječaka koda

Potrebno je stvoriti novu mapu "pages" unutar src mape kako bi se zadržala bolja struktura datoteka. Za svaku stranicu potrebno je dodati novu .jsx datoteku.

U ovom slučaju potrebno je dodati Home.jsx, Create.jsx, Post.jsx i Profile.jsx kako slijedi:

```
mkdir src/pages
cd src/pages && touch Home.jsx Create.jsx Post.jsx Profile.jsx
```

U 4.5. kodnom isječku, stvara se početna stranica aplikacije.

4.2.4 Spajanje klijentske i poslužiteljske strane

Za dohvaćanje isječaka koda s poslužitelja, potrebno je povezati klijentsku s poslužiteljskom stranom. S poslužiteljem je moguće komunicirati koristeći paket koji se zove ‘Axios‘ [8]. Axios omogućava lakši način rukovanja HTTP zahtjevima. Axios je potrebno instalirati pomoću NPM-a kako slijedi:

```
cd client/
npm i axios
```

Kako bi struktura datoteka ostala razumljivom, potrebno je stvoriti zasebnu mapu ‘api‘ te novu datoteku ‘index.js‘ kako bi sve bilo dosljedno. Nova mapa api i datoteka index.js stvara se sljedećim naredbama.

```
mkdir src/api
cd api
touch index.js
```

Kod u kodnom isječku 4.6. potrebno je dodati u novostvorenu index.js datoteku. Nakon što je kod iznad dodan, na početnoj stranici je omogućeno dohvaćanje svih isječaka koda pomoću React "kuke" ‘useEffect‘ i funkcije ‘fetchSnippets‘ . Dohvaćeni isječci koda se zatim pohranjuju u React ‘state‘ i prikazuju se korisniku na početnoj stranici.

```
import { fetchSnippets } from "../api";

useEffect(() => {
    fetchSnippets()
      .then(({ data }) => {
        setSnippets(data.snippets.reverse());
        setLoading(false);
      });
  }, []);
```

4.2.5 Stranica za stvaranje isječaka koda

Potrebno je napraviti stranicu na kojoj će korisnici moći stvarati isječke koda. Sljedeći isječak koda (isječak koda 4.7.) prikazuje implementaciju stranice za stvaranje isječaka koda.

Kada se POST zahtjev pošalje na kreirani URL, fetch će dodati novi zapis u bazu podataka.

Slika 4.4 Kodni isječak - stvaranje komponente isječka koda

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

import { Avatar, Code } from "./";
import { copy, edit, remove } from '../assets'
const URL = "https://code-sharing-platform.vercel.app";

const PostCard = ({ snippet }) => {
  const {
          _id: snippetId,
    title,
    description,
    code,
    language,
    postedBy: { name, email, _id: userId }
 } = snippet;

  return (
    <article className="mb-16 border-b pb-10 border-black200">
      <header className="flex justify-between items-center">
        <Link to={`/user/${userId}`} title="Profile Page">
          <Avatar name={name} email={email} />
        </Link>
        <div className="flex gap-5 sm:gap-8 text-white700">
          <button
                  className="hidden sm:flex items-center gap-1"
                  title="Share Snippet"
            onClick={() => copyToClipboard(`${URL}/snippet/${snippetId}`)} >
                    <img src={copy} alt="copy" />
          </button>
        </div>
      </header>
      <Link to={`/snippet/${snippetId}`}>
        <h3 className="text-xl mt-10 mb-3 font-medium">{title}</h3>
        <p className="mb-8 text-white700">{description}</p>
        <Code language={language} code={code} />
      </Link>
    </article>
  );
};
export default PostCard;
```

Slika 4.5 Kodni isječak - stvaranje početne stranice aplikacije

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

import { PostCard, Loader } from "../components";

const HomePage = () => {
  const [snippets, setSnippets] = useState([]);
  const [loading, setLoading] = useState(true);

  return (
    <main>
      {loading && <Loader />}

      {!loading && !snippets.length && (
        <h1 className="text-3xl sm:text-4xl font-semibold text-center
        mb-8 sm:mb-14">
          No Snippets to show!
        </h1>
      )}

      {snippets && snippets.map((snippet) => ( <PostCard snippet={snippet}
      key={snippet._id} /> ))}
    </main>
  );
};

export default HomePage;
```

Slika 4.6 Kodni isječak - spajanje klijentske i poslužiteljske strane

```
import axios from "axios";

const API = axios.create({
  //baseURL: "https://backend-code-sharing-platform.vercel.app",
   baseURL: "http://localhost:1337",
});

API.interceptors.request.use((req) => {
  if (localStorage.getItem("auth")) {
    req.headers.Authorization = `Bearer ${JSON.parse(
            localStorage.getItem("auth")
    ).token}`;
  }

  return req;
});

// Snippet
export const fetchSnippets = () => API.get("/snippet");
export const fetchSnippet = (id) => API.get(`/snippet/${id}`);
export const createSnippet = (newSnippet) => API.post(`/snippet`, newSnippet);
export const updateSnippet = (id, updatedSnippet) =>
API.patch(`/snippet/${id}`, updatedSnippet);
export const deleteSnippet = (id) => API.delete(`/snippet/${id}`);

// Profile
export const profileSnippets = (id) => API.get(`/snippet/user/${id}`);

// Auth
export const signup = (data) => API.post("/auth/signup", data);
export const login = (data) => API.post("/auth/login", data);
```

Slika 4.7 Kodni isječak - kreiranje stranice za stvaranje isječaka koda

```
import { useState, useEffect } from "react";
import { useNavigate, useLocation } from "react-router-dom";

import { Input, Button, Textarea, Error } from "../components";
import { createSnippet, updateSnippet } from "../api";

const CreatePage = () => {
  const [snippet, setSnippet] = useState({ title: "", description: "", language:
          tags: "", code: "", postedBy: JSON.parse(localStorage.getItem("auth"))
  );
  const [isUpdating, setIsUpdating] = useState(false);
  const [error, setError] = useState("");
  const navigate = useNavigate();
  const location = useLocation();
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
        const { data } = await createSnippet(snippet);

        if (data.error) return setError(data.message);
      }
      navigate("/");
    } catch (error) {
      setError(error.response.data.message);
    }
  };

  const handleChange = ({ target: { name, value }}) => setSnippet({
  ...snippet, [name]: value
  });
  return (
    <main>
      <h1>
        Share code snippets in seconds.
      </h1>
      {error && <Error message={error} isAlert />}
      <form onSubmit={handleSubmit}>
        // Inputs...
      </form>
    </main>
  );
};
```

Poglavlje 5

Završetak i poboljšanja

Dijeljenje koda među kolegama programerima ima mnoge prednosti, uključujući ušted u vremena i novca te povećanu inovativnost. Koristeći aplikaciju napravljenu za potrebe pisanja ovog završnog rada, programeri mogu dijeliti isječke koda s drugim programerima, ili ih spremiti za kasniju upotrebu. Korisnik može zalijepiti kod u obrazac i podijeliti ga putem web adrese. Druga osoba može posjetiti tu vezu i jednim pritiskom miša kopirati cijeli isječak koda. Isječci koda su također sortirani po oznakama i po programskim jezicima, kako bi programeri mogli lakše pronaći one koji su im potrebni u određenom trenutku. U svrhu izgradnje projekta, korišten je MERN razvojni okvir koji se koristi za lakšu i bržu implementaciju full-stack web aplikacija. Projekt je završen u roku od mjesec dana. Bio je to doista ogroman projekt. Iako je bilo mnogo izazova, zahvaljujući marljivosti projekt je završen. Softverski projekt nikada nije uistinu dovršen, uvijek će postojati dio/značajka/funkcionalnost koja se može poboljšati i može učiniti aplikaciju još boljom. Projekt se u budućnosti može poboljšati dodavanjem novih funkcionalnosti kao što su *lajkanje* i komentiranje objave isječka koda, paginacija, mogućnost privatnih i javnih isječaka koda te se aplikacija može učiniti progresivnom web aplikacijom.

Literatura

[1] „Mongo DB". (2022.), adresa: https://www.mongodb.com (pogledano 2. 9. 2022.).

[2] „Express JS". (2022.), adresa: https://expressjs.com (pogledano 2. 9. 2022.).

[3] „React JS". (2022.), adresa: https://reactjs.org (pogledano 2. 9. 2022.).

[4] „Node JS". (2022.), adresa: https://nodejs.dev/en/ (pogledano 2. 9. 2022.).

[5] „The V8 JavaScript Engine". (2022.), adresa: https://nodejs.dev/en/learn/the-v8-javascript-engine (pogledano 2. 9. 2022.).

[6] „Visual Studio Code". (2022.), adresa: https://code.visualstudio.com/ (pogledano 5. 9. 2022.).

[7] „JOI". (2022.), adresa: https://www.npmjs.com/package/joi (pogledano 5. 9. 2022.).

[8] „Axios". (2022.), adresa: https://axios-http.com/docs/intro (pogledano 5. 9. 2022.).

Sažetak

Programerska zadaća je neprestano pisanje programskog koda, no nerijetko se određeni dijelovi već napisanog koda mogu ponovno iskoristiti na novim projektima. Koristeći MERN razvojni okvir, napravljena je aplikacija koja omogućava dijeljenje algoritama, dijelovi korisničkog sučelja i ostalih isječaka. Klijentska strana aplikacije (React.js), preko Expressa, na poslužiteljsku stranu aplikacije (Node.js) šalje isječke koda u JSON formatu koji se na kraju spremaju u MongoDB bazi podataka.

Ključne riječi **— MERN, React.js, Express.js, MongoDB, Node.js**

Abstract

Programmers constantly write code, but often certain parts of the already written code can be reused in new projects. Using the MERN development framework, the application allows sharing of algorithms, parts of the user interface and other snippets. The client side of the application (React.js), via Express.js, sends snippets of code in JSON format to the server side of the application (Node.js), which are then saved in the MongoDB database.

Keywords **— MERN, React.js, Express.js, MongoDB, Node.js**