Sustav za pohranu i prijenos podataka u oblaku koristeći MERN stog

Ištvan, Mihael

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

Preddiplomski studij računarstva

SUSTAV ZA POHRANU I PRIJENOS PODATAKA U OBLAKU KORISTEĆI MERN STOG

Završni rad

Mihael Ištvan

Osijek, 2021.

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 1

2. TEHNOLOGIJE ZA RAZVOJ APLIKACIJE 2

2.1. MERN stog 2

2.2. Princip rada MERN stoga 2

2.3. React.js korisnička razina 3

2.4. Express.js i Node.js poslužiteljska razina 4

2.5. Razina baze podataka MongoDB 4

2.6. Izbor MERN stoga 4

3. PRINCIP RADA I ARHITEKTURA SUSTAVA 6

3.1. Arhitektura sustava u oblaku 6

3.2. Struktura programskog koda 7

3.2.1. Korisničko sučelje 8

3.2.2. Serverska strana 15

3.2.3. Baza podataka 17

4. ZAKLJUČAK 21

LITERATURA 22

POPIS I OPIS UPOTRIJEBLJENIH KRATICA 23

SAŽETAK 25

ABSTRACT 26

ŽIVOTOPIS 27

1. UVOD

Spremanje datoteka na sigurnosnom disku omogućuju vraćanje podataka s ranijeg vremena kako bi se lakše oporavili od neplanirane greške diska ili sustava. Pohranjivanje kopije podataka na zaseban medij ključno je za zaštitu od gubitka ili oštećenja primarnih podataka. Bez obzira premještate li datoteke s jednog uređaja na drugi, s telefona na tablet ili s telefona na računalo, prijenos podataka uvijek je dugotrajan zadatak, osobito ako želite dijeliti velike datoteke.

Problem ovog završnog rada je osigurati siguran prijenos manjih datoteka bez ikakvih ograničenja ili ograničenja. Srećom, svi se ovi problemi mogu izbjeći jednostavnom upotrebom pouzdane aplikacije za dijeljenje datoteka. Korištenje ovih aplikacija neće omogućiti samo trenutni prijenos, već i pregled i sigurnosnu kopiju podataka.

Završni rad navodi prednosti i potencijalne primjene modernih tehnologija na probleme prijenosa podataka. Izrađena je jednostavna aplikacija koja prikazuje njene glavne koncepte, potkrepljene primjerima. Koristeći aplikaciju kao primjer dano je rješenje čestih problema.

1.1. Zadatak završnog rada

Izraditi sustav za pohranu i prijenos podataka u oblaku koristeći MERN stog (engl. MERN stack – *MongoDB, Express.js, React.js and Node.js stack*). Aplikacija mora omogućiti osnovne funkcionalnosti navedene aplikacije i omogućiti moderan, pouzdan i siguran prijenos datoteka između računala međusobno povezanih internetskom vezom.

2. TEHNOLOGIJE ZA RAZVOJ APLIKACIJE

2.1. MERN stog

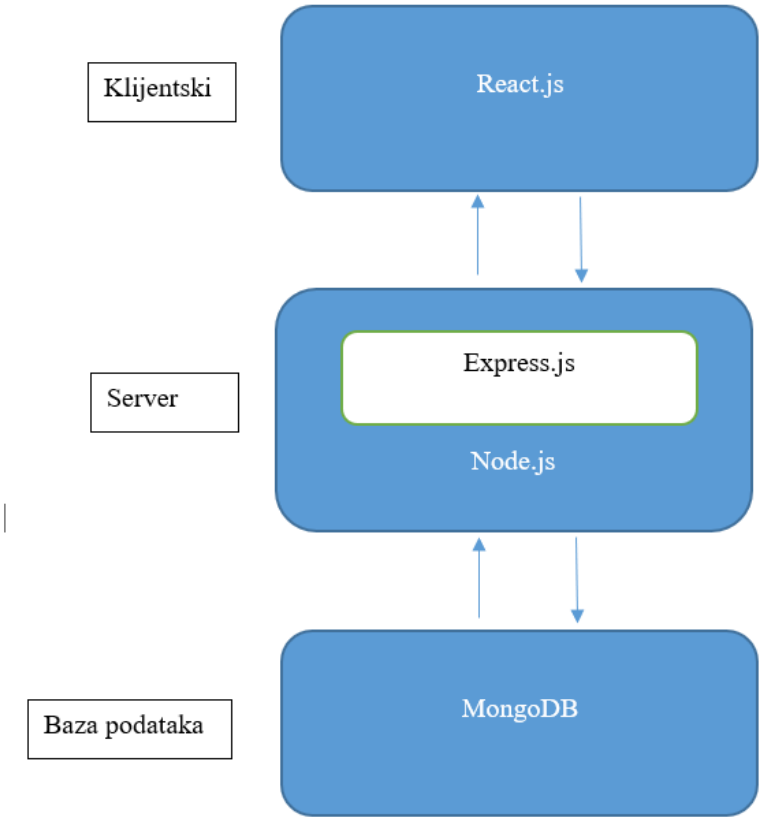
MERN je kratica za MongoDB, Express, React, Node, nakon četiri ključne tehnologije koje čine stog:

1. MongoDB - baza podataka
2. Express (.js) - Node.js internetski okvir (engl. *Node.js web framework*)
3. React (.js) - JavaScript internetski okvir na strani klijenta (engl. *Javascript frontend framework*)
4. Node (.js) - vodeći JavaScript internetski poslužitelj (engl. *Javascript web server*)

MERN je jedna od nekoliko varijacija MEAN stoga (engl. MEAN - *MongoDB, Express.js, Angular.js and Node.js stack*), gdje je tradicionalni Angular.js internetski okvir na strani klijenta zamijenjen sa React.jsom. Ostale inačice uključuju MEVN stog (engl. MEVN - *MongoDB, Express.js, Vue.js and Node.js stack)* gdje može funkcionirati bilo koji JavaScript klijentski okvir. Express.js i Node.js čine srednju (aplikacijsku) razinu. Express.js je mrežni okvir na strani poslužitelja, a Node.js popularna i moćna JavaScript poslužiteljska platforma. Bez obzira koju varijantu odabrali, ME (R ili V ili A) N je idealan pristup radu s JavaScriptom i JSON-om (engl. JSON – *Javascript object notation*), sve do kraja.

2.2. Princip rada MERN stoga

Arhitektura MERN stoga omogućuje jednostavnu izgradnju troslojne arhitekture (sučelja, pozadine, baze podataka) u potpunosti koristeći JavaScript i JSON. Tipična internetska aplikacija traži znanja o više nezavisnih tehnologija za razvoj što može dodatno zakomplicirati i povećati cijenu razvoja aplikacije. Razvoj aplikacije sa tehnologijama koje imaju zajedničku platformu znači dublje razumijevanje programera o osnovama aplikacije te o mogućim problemima i naprednim metodama za razvoj. Olakšava razvoj boljih performansa i smanjuje broj linija koda naprednom programeru. MERN stog je primjer stoga sa zajedničkom razvojnom platformom – Javascript.



Slika 2.1 Princip rada MERN stoga

2.3. React.js korisnička razina

Najviša razina MERN stoga je React.js, deklarativni JavaScript okvir za stvaranje dinamičkih aplikacija na strani klijenta u HTML-u (engl. HTML – *Hypertext Markup Language*). React.js omogućuje izgradnju složenih korisničkih sučelja kroz jednostavne komponente, povezivanje s podacima na pozadinskom poslužitelju i prikazivanje u obliku HTML-a. React.jsova jaka strana je rukovanje korisničkih sučeljima zasnovanim na podacima, s minimalnim kodom i minimalnom boli, a sadrži sva čudesa koja bi očekivali od modernog internetskog okvira:

- izvrsna podrška za obrasce
- rukovanje pogreškama
- događaje
- popise

2.4. Express.js i Node.js poslužiteljska razina

Sljedeća razina je Express.js okvir na strani poslužitelja, pokrenut unutar Node.js poslužitelja. Express.js se računa kao „brzi minimalistički internetski okvir za Node.js“, i to je doista upravo ono što jest. Express.js ima moćne modele za usmjeravanje URL-ova (podudaranje dolaznog URL-a s poslužiteljskom funkcijom) (engl. URL – *Uniform Resource Locator*) i rukovanje HTTP (engl. HTTP – *Hypertext Transfer Protocol*) zahtjevima i odgovorima. Izradom XML HTTP zahtjeva (engl. XML – *Extensible Markup Language*) ili GET-ova ili POST-ova (HTTP API poziva) (engl. API – *Application Programming Interface*) s React.js klijentskog dijela, može se povezati s Express.js funkcijama koje pokreću aplikaciju. Te funkcije zauzvrat koriste MongoDBove upravljačke programe Node.js, bilo putem povratnih poziva za korištenje *Promisa* (asinkrona funkcija koja vraća obećanje) za pristup i ažuriranje podataka u MongoDB bazi podataka.

2.5. Razina baze podataka MongoDB

Ako aplikacija pohranjuje bilo kakve podatke (korisničke profile, sadržaj, komentare, prijenose, događaje itd.), Tada ćete htjeti bazu podataka s kojom je jednako lako raditi kao React.js, Express.js i Node.js. Tu dolazi MongoDB: JSON dokumenti stvoreni na React.js klijentskom dijelu mogu se poslati na Express.js poslužitelj, gdje se mogu obraditi i (pod pretpostavkom da su valjani) pohraniti izravno u MongoDB za kasnije preuzimanje.

2.6. Izbor MERN stoga

MongoDB, baza podataka dokumenata u korijenu MERN stoga. MongoDB je dizajniran za izvorno pohranjivanje JSON podataka (tehnički koristi binarnu verziju JSON-a koja se naziva BSON) (engl. BSON - *computer data interchange format*), a sve od sučelja naredbenog retka do jezika upita (MQL ili MongoDB Query Language) (engl. MQL - *Marketing Qualified Lead*) izgrađeno je na JSON-u i JavaScriptu. MongoDB izuzetno dobro surađuje s Node.jsom te nevjerojatno olakšava pohranu, manipulaciju i predstavljanje JSON podataka na svakoj razini aplikacije. Za matične aplikacije u oblaku, MongoDB Atlas to čini još lakšim, pružajući automatsko skaliranje MongoDB klastera na davaču usluga u oblaku po izboru. Express.js (pokrenut na Node.js) i React.js čine JavaScript / JSON aplikaciju MERN punim stogom (engl. *full stack*). Express.js je aplikacijski okvir na poslužitelju koji obavija HTTP zahtjeve i odgovore i olakšava modulaciju URL-ova na funkcije na

strani poslužitelja. React.js je sučelni JavaScript okvir za izgradnju interaktivnog korisničkog sučelja u HTML-u i komunikaciju s udaljenim poslužiteljem [1].

Kombinacija znači da JSON podaci prirodno teku od naprijed prema natrag, što ga čini brzim za nadogradnju i relativno jednostavnim za otklanjanje pogrešaka. Osim toga, potrebno je znati samo jedan programski jezik i strukturu dokumenata JSON kako bismo razumjeli cijeli sustav. MERN je odabir za današnje internetske programere koji se žele brzo kretati, posebno za one s React.js iskustvom [2].

3. PRINCIP RADA I ARHITEKTURA SUSTAVA

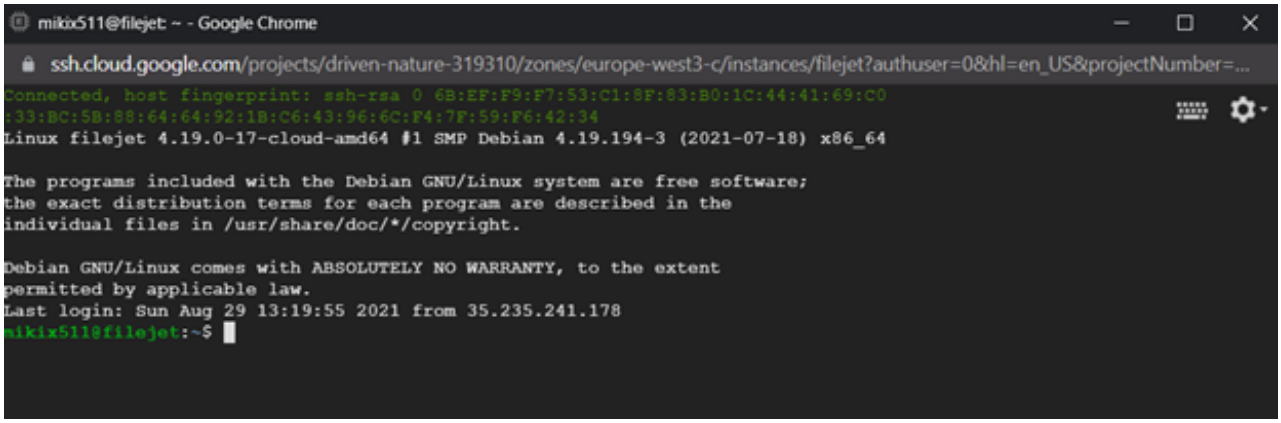
Kao primjer rada prijenosa podataka u oblaku izrađena je aplikacija pod imenom Filejet. Ime aplikacije objašnjava svrhu aplikacije i asocira na brzi prijenos datoteka.

3.1. Arhitektura sustava u oblaku

Za posluživanje klijentskog i API dijela aplikacije koristimo Google Cloud. GCP (engl. GCP – *Google Cloud Platform*), koji nudi Google, skup je računalnih usluga u oblaku koje rade na istoj infrastrukturi koju Google interno koristi za svoje proizvode krajnjih korisnika, kao što su Google pretraživanje, Gmail, pohrana datoteka i YouTube [3]. Postupak posluživanja API dijela aplikacije malo je kompliciraniji nego što je klijentskog. Google Cloud daje 300$ početnog kredita novom korisniku što nam omogućuje posluživanje API-a na njihovom servisu. Koraci posluživanja API-a na Google Cloudu su sljedeći:

- Prijavimo se na Google Cloud (kreditna kartica je potrebna).
- Nakon registracije kreiramo projekt
- Nakon izrade projekta dodajemo VM instancu (engl. *Virtual Machine*), koja je zapravo operacijski sustav preko virtualnog diska. Izradom VM instance izabiremo lokaciju servera, operacijski sustav i konfiguraciju našeg virtualnog sustava. Konfiguracija i lokacija sustava varira u cijeni usluge.
- Nakon izrade VM instance dobivamo vanjsku IP adresu (engl. IP – *Internet Protocol*) servera kojeg koristimo za spajanje unutar aplikacije a za konfiguraciju u sami server spajamo se pomoću SSH (engl. SSH - *Secure Shell Protocol*)

Nakon izrade VM instance imamo pristup našem virtualnom sustavu. Za izradu ove aplikacije izabrali smo Debian 10 (Linux) kao optimalno rješenje.



Slika 3.1 Prikaz SSH terminala „Filejet“ VM instance

Kako bi instalirali naš API na virtualni sustav potrebno je instalirati MongoDB bazu podataka na sustav. Nakon instalacije, klijentski dio i API dohvaćamo pomoću Gita (sustav za praćenje projekata) te ga instaliramo pomoću Yarna i pokrećemo NPM-om (engl. NPM – *Node Package Manager*) kao i u procesu razvoja aplikacije. Yarn je upravitelj paketa koji se udvostručuje i kao voditelj projekta. Sličan NPM-u ali prilikom instalacije paketa instalira sve pakete u isto vrijeme pa je proces instalacije paketa minimiziran. Klijentski dio pokrenut je na porti 80 koja je industrijski standard za posluživanje klijentskog dijela aplikacije na serveru. Za omogućavanje rada procesima u pozadini servera korišten je paket tmux koji pomoću kreiranja sesija omogućuje nastavak rada procesa i kada nismo spojeni na server.

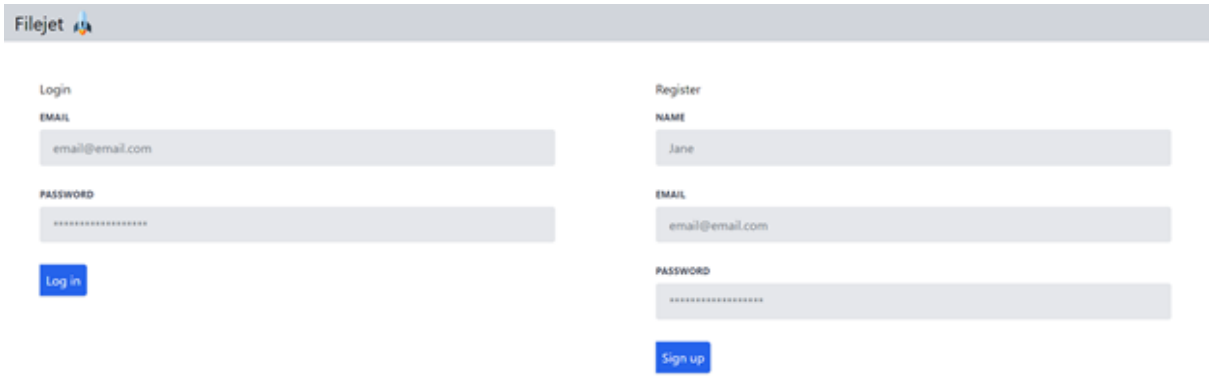
3.2. Struktura programskog koda

Za izradu projekta korišten je VSC (engl. VSC – *Visual Studio Code*), NPM, Yarn i GitHub. VSC je uređivač koda redefiniran i optimiziran za izgradnju i uklanjanje pogrešaka modernih internetskih i aplikacija u oblaku [4]. NPM je upravitelj paketa za programski jezik JavaScript koji održava NPM Inc. NPM je zadani upravitelj paketa za JavaScript runtime koji okruženje Node.js. Sastoji se od klijenta naredbenog retka, koji se naziva i NPM, i mrežne baze podataka javnih i plaćenih privatnih paketa, koja se naziva NPM registar [5]. GitHub je pružatelj internetskog posluživanja za razvoj softvera i kontrolu verzija pomoću Gita. Nudi distribuiranu kontrolu nad verzijama i funkcionalnost upravljanja izvornim kodom Gita, plus vlastite značajke [6]. Radi lakšeg diferenciranja korisničkog sučelja od servera, strukturu programskog koda ćemo podijeliti na dva poglavlja: korisničko sučelje (engl. Frontend) i pozadinski server (engl. Backend)

3.2.1. Korisničko sučelje

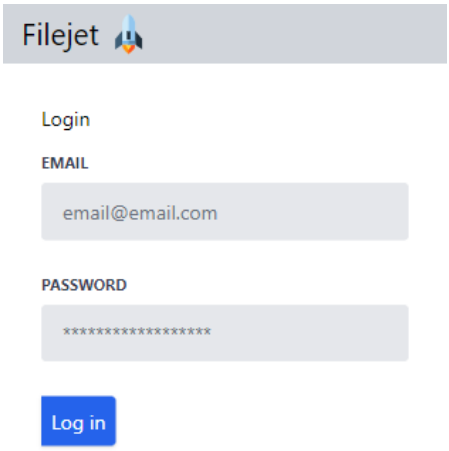
Kako je već prije navedeno, za izradu korisničkog sučelja korišten je React.js. Radi jednostavnosti projekta i brže instalacije korišten je Next.js. Next.js je otvoreni izvorni React.jsov internetski okvir za razvojni razvoj koji je stvorio Vercel koji omogućuje funkcionalnost kao što je generiranje na strani poslužitelja i generiranje statičkih internetskih stranica za internetske aplikacije utemeljene na React.jsu [7]. Radi brzine i jednostavnosti korisničkog sučelja, aplikacija je dizajnirana u samo dva prikaza:

- Login / Register komponenta
- DashBoard komponenta

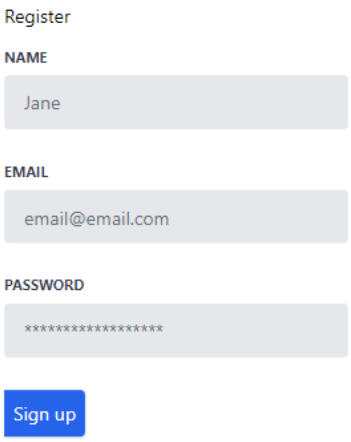


Slika 3.2 Prikaz Login / Register sučelja

Aplikacija je potpuno responzivna. Responzivan internetski dizajn pristup je internetskog razvoja koji stvara dinamičke promjene u izgledu web stranice, ovisno o veličini zaslona i orijentaciji uređaja koji se koristi za njegovo pregledavanje. Responzivan dizajn omogućuje pregledan pristup aplikaciji sa bilo kojeg uređaja neovisno o veličini i obliku njegovog zaslona.



Slika 3.3 Prikaz Login sučelja na mobilnom uređaju



Slika 3.4 Prikaz Register sučelja na mobilnom uređaju

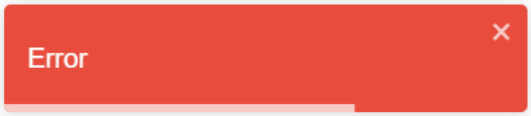
Statička Navbar komponenta služi za prikaz logo.svg vektora aplikacije te imena aplikacije u lijevom kutu. Nakon uspješne autentifikacije prikazuje se i „Log out“ dugme za odjavu korisnika a izgled aplikacije korišten je Tailwind.css. Tailwind je CSS (engl. CSS - *Cascading Style Sheets*) okvir koji je od uslužnih programa pun predefiniranih klasa koji se mogu koristiti za izradu bilo kojeg dizajna izravno u označavanju. On omogućuje brzo stilsko oblikovanje HTML elemenata te direktno korištenje vlastitih klasa u HTML elementima [8]. Za izradu formi, korišten je NPM paket *Tailwind Forms*. Tailwind Forms je baziran na Tailwind.cssu te nudi dodatne predefinirane klase za oblikovanje HTML form elemenata poput *input*, *textarea* i drugih.

Kako bismo pristupili svojim datotekama potrebno se prijaviti u vlastiti račun. Ako korisnik nema vlastiti račun onda se može registrirati. Nakon uspješne registracije pojavi se poruka uspješnosti, NPM paket *react-toastify*, kako bi aplikacija prikazala korisniku dodatne informacije o uspješnosti prijave i registracije.

```
<ToastContainer
        position="bottom-right"
        autoClose={5000}
        hideProgressBar={false}
        newestOnTop={false}
        closeOnClick
        rtl={false}
        pauseOnFocusLoss
        draggable
        pauseOnHover
      />
```

Programski kod 3.1 Toast poruka

Slika 3.5 Prikaz Toast.success poruke



Slika 3.6 Prikaz Toast.error poruke

Nakon uspješne registracije prikazuje se UserDashboard komponenta koja se sadrži od tablice sa 6 stupaca, te se prikaže „Log Out“ dugme na navigacijskoj traci.

Slika 3.7 Prikaz UserDashboard komponente

Struktura komponenata je sljedeća: prilikom prve inicijalizacije aplikacije prikazuje se Auth komponenta. Auth komponenta sadrži Login i Register komponentu koje sadržavaju Login i Register forme postavljene u tablicu sa dva stupca prikazan na slici 3.2.1.1. Za autentikaciju koriste se funkcije *createContext*, *useState* i *useReducer* koje služe za stvaranje globalnog stanja koje se mogu prenositi u *child* komponente pomoću uvoza, time ih nije potrebno slati pomoću propova (engl. *props*). useReducer i useState oboje upravljaju globalnim stanjem. Dok useState koristimo za promjenu globalnog stanja, useReducer se koristi kako bi se definiralo početno stanje globalnog stanja i promjena stanja, što znači da daje više kontrole upravljanja stanjem nego useState.

```
function MyApp() {
  const [state, dispatch] = React.useReducer(reducer, initialState);
  return (
    <AuthContext.Provider
      value={{
        state,
        dispatch,
      }}
    >
      <div id="app">
        <Nav isAuthenticated={state.isAuthenticated} />
        <div>
          {!state.isAuthenticated ? <Auth /> : <UserDashboard state={state} />}
        </div>
      </div>
    </AuthContext.Provider>
  );
}
```

Programski kod 3.2 Upravljanje autentikacijom

```
export const AuthContext = React.createContext();
const initialState = {
  isAuthenticated: false,
  user: null,
  token: null,
const reducer = (state, action) => {
  switch (action.type) {
    case "LOGIN":
      localStorage.setItem("user", JSON.stringify(action.payload.data.user));
      localStorage.setItem("token", JSON.stringify(action.payload.data.token));
      return {
        ...state,
        isAuthenticated: true,
        user: action.payload.data.user,
        token: action.payload.data.token,
      };
    case "LOGOUT":
      localStorage.clear();
      return {
        ...state,
        isAuthenticated: false,
        user: null,
      };
    default:
      return state;
  }
};
```

Programski kod 3.3 upravljanje globalnim stanjem

Prilikom uspješne autentifikacije komponenta dijete Login i Register koje kontroliraju autentikaciju aplikacije postavljaju globalno stanje na „LOGIN“, time se u globalno stanje isAuthenticated stavlja na *true* i sprema se korisnički objekt i JWT (engl. JWT – *Jason Web Token*) koji služi za upravljanje korisničkim datotekama prilikom trenutne sesije. Razlog spremanja podataka autentifikacije u globalno stanje je da sve komponente u aplikaciju mogu koristiti to sranje pomoću uvoza, time možemo u svakoj komponenti imati različite događaje ovisno o stanju autentifikacije i različite pozive na server ovisno o namjeni komponente. Nakon uspješne autentifikacije pokreće se UserDashboard komponenta koja prilikom svoje inicijalizacije radi poziv na server gdje pomoću globalnog stanja trenutno prijavljenog korisnika dohvaća sve datoteke trenutnog korisnika i prikazuje

ih u tablicu. Za upravljanje pozivima na server koristi se Axios. Axios je lagani HTTP klijent koji se temelji na obećanju (engl. *promise*), što znači da omogućuje asinkrono dohvaćanje datoteka [9]. Najpopularnije HTTP metode za slanje poziva na server su: GET, POST, DELETE i UPDATE.

```
function getData() {
    console.log("fetching data");
    axios({
      method: "get",
      url: " http://35.198.85.204:5000/api/files",
      params: {
        ownerId: state.user._id,
      },
    })
      .then((res) => {
        setData({
          ...data,
          files: res.data,
        });
      })
      .catch((error) => {
        console.log(error);
      });
```

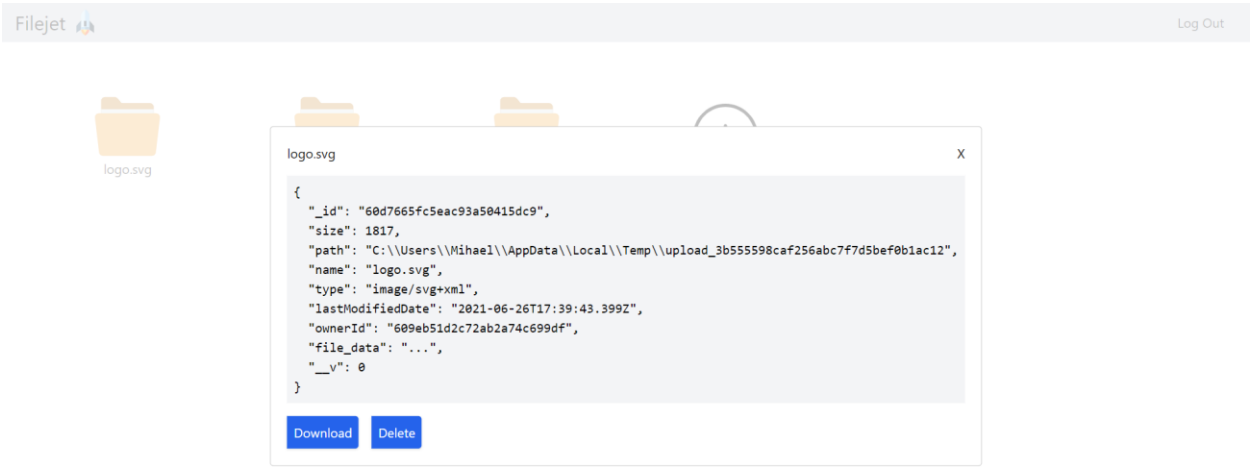
Programski kod 3.4 Dohvaćanje svih korisničkih datoteka

Tablica za prikaz datoteka sastoji se od File i InputFile komponente. Svaka dohvaćena datoteka prikazuje se kao zasebna File komponente a na kraju tablice nalazi se InputFile komponenta koja služi za slanje nove datoteke.

```
<div className="grid md:grid-cols-6 grid-cols-2 gap-8 p-12">
        {data.files.map((file) => (
          <File
            key={file._id}
            state={state}
            file={file}
            onFileDelete={refetchData}
          />
        ))}
        <InputFile state={state} onFileDelete={refetchData} />
      </div>
```

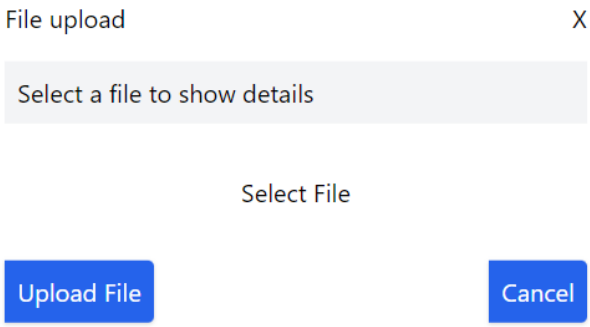
Programski kod 3.5 Prikaz datoteka

Klikom na File komponentu otvara se Modal File komponente. Modal je skočni prozor koji prikazuje dodatne funkcionalnosti neke komponente. U našem slučaju prikazuje dodatne funkcionalnosti svake zasebne datoteke: ime datoteke i objekt datoteke. Nudi dodatke funkcionalnosti poput dugmeta za pokretanje skidanja datoteke i dugmeta za brisanje datoteke iz baze podataka.

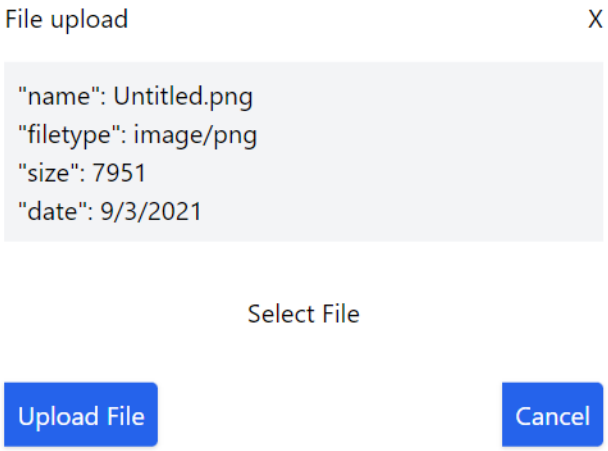


Slika 3.8 Modal File komponente

Klikom na InputFile komponentu otvara se Modal koji prikazuje dodatne funkcionalnosti za prijenos nove datoteke: izbor datoteke za slanje, objekt datoteke, dugme za slanje i dugme za zatvaranje Modala.



Slika 3.9 Modal InputFile komponente



Slika 3.10 Modal InputFile komponente nakon izbora datoteke

Nakon uspješnog slanja datoteke UserDashboard ponovo šalje GET poziv na server te prikazuje novo stanje korisničkih datoteka.

3.2.2. Serverska strana

Za izradu serverske strane korišten je Node.js, Express.js te baza podataka MongoDB. U ovom poglavlju objasnit ćemo programski kod vezan uz Express.js. Osnovna struktura serverske aplikacije su rute, središnje funkcije (engl. *middleware*) i upravljanje bazom podataka. Rute su kao mape koje se mogu na različit način pristupiti od strane klijenta. Osnovan primjer toga bio bi da klijent pomoću GET poziva može pristupiti ruti /api/files te time dohvatiti sve datoteke trenutnog korisnika. To ne znači da GET poziv radi i na /api/users kako bi dohvatio sve korisnike ali i može raditi ako je takva obrada poziva definirana na toj ruti.

```
const app = express();
const helmet = require("helmet");
const files = require("./routes/files");

app.use(helmet());
app.use("/api/files", files);
```

Programski kod 3.6 Korištenje rute files i središnje funkcije helmet

Osnovne rute potrebne za rad aplikacije su:

- /api/files
- /api/users

- /api/auth

Iz razloga što je instanca objekta Express.jsa moguća samo jedna biti aktivna na projektu, za konfiguriranje različitih ruta potrebno je koristiti *express.Router()* koji može biti instanciran beskonačno puta te koristi kako bi rute mogle biti aktivne. Helmet NPM paket koji je korišten kao središnja funkcija koja podacima slanih između klijenta i servera stavlja dodatni HTTP zaglavlje (engl. *header*) koji osigurava podatke [10]. Cors je također NPM paket korišten na projektu kao središnja funkcija koja eliminira greške za zaštitu pristupa od lokalnog servera što omogućuje održavatelju projekta da testira promjene slanjem probnih podataka na server. [11] Sljedeća središnja funkcija koja je korištena je npm paket Morgan, koji se koristi kao dnevnik na konzolu [12]. Također koristimo *express.json()* i *express.urlencoded({extended: true})* kao središnju funkciju za oblikovanje podataka koji dostižu na server. Svaki HTTP poziv koji definiramo sastoji se od zahtjeva i odgovora. Radi zaštite oblika podataka koji smiju stizati na server ovisno o ruti koristimo NPM paket Joi. Joi je zbirka za validaciju podataka Javascripta. Princip rada Joia je tako na svakoj ruti i na svakom pozivu servera definiramo shemu podataka koji se smiju nalaziti u JSON objektu. Način na koji definiramo shemu takav objekt mora stići od klijent na server. Ako se objekt ima najmanje varijacije, vraćamo odgovor na klijent sa odgovarajućem statusnim kodom i greškom. Ako Joi validacija ne vrati grešku, onda funkcija nastavlja sa svojom zadaćom te vraća klijentu odgovor o svojem rezultatu [13].

```
router.get("/", (req, res) => {
  const schama = Joi.object({
    ownerId: Joi.string().required(),
  });
  const { error } = schama.validate(req.query);
  if (error) return res.status(400).send(error.details[0].message);

  getFiles(req.query.ownerId)
    .then((files) => res.send(files))
    .catch((err) => res.send(err));
});
```

Programski kod 3.7 Primjer konfiguracije GET poziva na /api/files ruti

Za autentifikaciju korisnika koristimo JWT i NPM paket Bcrypt. JWT jedna je od metoda kojom se omogućuje provjera autentičnosti, a da se zapravo ne pohranjuju nikakve informacije o korisniku u samom sustavu. JWT je najnoviji standard za održavanje trenutne sesije sa korisnikom prilikom njegove uspješne autentifikacije. Način rada JWT-a je da prilikom svake autentifikacije

generiramo tajni ključ pomoću korisničkog identifikacijskog broja spremljenog u bazi podataka. [9] Taj ključ šaljemo natrag na klijent te korisnik prilikom svake svoje promjene podataka (CRUD operacije nad svojim podacima) (engl. CRUD - *Create, read, update and delete*) natrag šalje svoj JWT na server. Server zatim provjerava JWT klijenta sa svojim JWT-om te ako su isti nastavlja dalje sa radom [14].

```
userSchema.methods.generateAuthToken = function () {
  const token = jwt.sign({ _id: this._id }, config.get("jwtPrivateKey"));
  return token;
};
```

Programski kod 3.8 Funckija za generiranje trenutnog ključa sesije

Bcrypt je NPM paket kao zbirka koji služi za spremanje korisničke lozinke u bazu podataka. Koristi se radi zaštite lozinke korisnika od vanjskih nametnika ili administratora baze podataka. Način rada Bcrypta je da prilikom registracije novog korisnika generira sol (engl. *salt*) određene veličine. Salt se koristi kao tajni kod za kriptiranje korisničke lozinke koju zna samo Bcrypt. Prilikom prijave korisnika u sustav poslana šifra od klijenta uspoređuje se sa kriptiranom šifrom u bazi podataka te vraća odgovarajući odgovor korisniku ovisno o točnosti lozinke [15].

3.2.3. Baza podataka

Baza podataka korištena na projektu je MongoDB. Kako bi koristili navedenu bazu podataka sa Expressom potrebno je instalirati NPM paket Mongoose. Mongoose je moguće instancirati više puta te se koristi na svakoj ruti [16].

```
const mongoose = require("mongoose");
mongoose
  .connect("mongodb://localhost/filejet")
  .then(() => console.log("Connected to MongoDB.."))
  .catch((err) => console.error("Cound not connect to MongoDB..", err));
```

Programski kod 3.9 Spajanje na bazu podataka

Za spremanje različitih tipova podataka u MongoDB potrebno je definirati model podataka, slično kao i sa relacijama u SQL-u (engl. SQL - *Structured Query Language*). Definiramo sva polja koje objekt podataka sadržava i tipove tih polja. Za odnose među podacima koristimo referencu na model podataka.

```
const File = mongoose.model(
  "file",
  new mongoose.Schema({
    file_data: { type: Buffer, required: true },
    name: { type: String, required: true },
    ownerId: { type: String, ref: "user" },
    path: { type: String, required: true },
    size: { type: Number, required: true },
    type: { type: String, required: true },
    lastModifiedDate: { type: Date, required: true },
  })
);
```

Programski kod 3.10 Model datoteke

Kako bismo mogli razlikovati datoteke različitih korisnika, svaka datoteka sadrži referencu na identifikacijski kod korisnika. Za dohvaćanje datoteka na server koristimo NPM paket *formidable* koji omogućuje slanje podataka iz inputa datoteke na server u obliku JSON-a. Za spremanje datoteka koristimo binarni zapis. Binarni zapis omogućuje spremanje datoteka do 16MB te nije potrebno spremati datoteke u lokalnu mapu servera, već spremamo sami binarni zapis datoteke u model datoteke [17].

```
async function addFile(newFile) {
  const file = new File(newFile);
  try {
    const result = await file.save();
    return result;
  } catch (err) {
    return err;
  }
}
```

Programski kod 3.11 Funkcija za spremanje datoteka

Funkcija *addFile* je asinkrona funkcija što znači da unutar funkcije čekamo obećanje neke druge funkcije. Obećanja omogućuju asinkroni rad koda. Funkcija koristi „try-catch“ blok koji se koristi kao rukovatelj greškama u modernom kodu. Funkcija *.save* vraća objekt sa rezultatom spremanja datoteke. Rezultat uspješnog spremanja datoteke je sami objekt datoteke koju smo spremili.

```
var Binary = require("mongodb").Binary;
const fs = require("fs");

router.post("/", (req, res) => {
  const form = formidable({ multiples: true });

  form.parse(req, (err, fields, files) => {
    if (err) {
      next(err);
      return;
    }
    const verifyFile = {
      name: files.File.name,
      type: files.File.type,
      path: files.File.path,
      size: files.File.size,
      lastModifiedDate: files.File.lastModifiedDate,
    };
    const { error } = validate(verifyFile);
    if (error) {
      return res.status(400).send(error.details[0].message);
    }

    var data = fs.readFileSync(files.File.path);
    var insert_data = { ...files.File };
    insert_data.ownerId = fields.ownerId;
    insert_data.file_data = Binary(data);
    const result = addFile(insert_data);
    res.send(result);
  });
});
```

Programski kod 3.12 Funkcija Express rukovatelja za dodavanja datoteka

Proces *post* rukovatelja za spremanje binarnih datoteka je sljedeći:

- Dohvati datoteku
- Spremi datoteku u binarni zapis
- Vrati klijentu zapis datoteke kao potvrda za uspješnim spremanjem

Za preuzimanje raznolikih binarnih datoteka spremljenih u Mongo bazi podataka koristimo sljedeći kod:

```
router.get("/download", async function (req, res) {
const schama = Joi.object({
_id: Joi.string().required(),
ownerId: Joi.string().required(),
});

const { error } = schama.validate(req.query);
if (error) return res.status(400).send(error.details[0].message);

const file = await getFileById(req.query._id);

res.writeHead(200, {
"Content-Type": file.type,
"Content-disposition": "attachment;filename=" + file.name,
"Content-Length": file.size,
});

res.end(Buffer.from(file.file_data, "binary"));
});
```

Programski kod 3.13 Funkcija za preuzimanje datoteka

Prethodno prikazani kod je jedna od mnogih mogućih rješenja preuzimanja binarnih datoteka, *res.writeHead* javlja klijentskom dijelu aplikacije da šalje datoteku a *Buffer.from* šalje niz bitova datoteke *res.end* metodom.

4. ZAKLJUČAK

MERN stog jedan je od najmoćnijih i najtraženijih internetskih stogova u 2021. godini što Filejet aplikaciji pruža najmodernija i najbrža programska rješenja. MERN stog je bolji izbor za razvojne okvire namijenjene značajnijim internetskim projektima. Jednostavno, pregledano i intuitivno korisničko sučelje izgrađeno u React klijentskom okviru nudi klijentu savršenu preglednost svojim datotekama na bilo kojem uređaju. Korištenjem Reacta kao klijentskog okvira dobivaju se prednosti optimizacije za tražilice (engl. SEO), što Filejet aplikaciju stavlja na bližu poziciju na internetskoj tražilici. Node pozadinski okvir omogućuje Filejet aplikaciji skalabilnost, brzinu, performanse i fleksibinost za nadogradnju. Sa jednostavnosti održavanja sesije pomogću JWT-a, Nodeove neblokirajuće, ulazno-izlazne operacije čine okruženje jednom od najbržih dostupnih opcija. MongoDB omogućuje sigurno spremište podataka. Spremanjem podataka datoteka u binarnom obliku MongoDB osigurava visoku skalabilnost u smislu količine podataka i propusnosti. Spremanjem korisničkih lozinka kao šifriran tekst omogućuje šifriran i siguran pristup pozadinskom dijelu pa i siguran dohvat datoteka. Koristeći navedene prednosti, Filejet nudi najmodernija rješenja pohrane manjih datoteka na sigurni server. Poveznica za pregled aplikacije dostupna je na: http://35.198.85.204/ a izvorni kod aplikacije dostupan je na:

- Klijentski dio: https://github.com/blekso/filejet_client
- Pozadinski dio: https://github.com/blekso/filejet_api

LITERATURA

[1] R. Barger, "MERN Stack - The Complete Guide", 2019.

[2] MongoDB, "MERN stack", 2021. [Online]. Dostupno na: https://www.mongodb.com/mern-stack.

[3] "Wikipedija," Google Cloud Platform, [Online]. Dostupno na: https://en.wikipedia.org/wiki/Google_Cloud_Platform. [Accessed 2021].

[4] "Visual Studio Code",2021. [Online]. Dostupno na: https://code.visualstudio.com/.

[5] "NPM", 2021. [Online]. Dostipno na: https://www.npmjs.com/.

[6] "Github", 2021. [Online]. Dostupno na: https://github.com/.

[7] "Next.js", 2021. [Online]. Dostupno na: https://nextjs.org/.

[8] "TailwindCSS", 2021. [Online]. Dostupno na: https://tailwindcss.com/.

[9] N. Uraltsev, "Axios", 2021. [Online]. Dostupno na: https://www.axios.com/.

[10] A. Baldwin, "Helmet", NPM, 2021. [Online]. Dostupno na: https://www.npmjs.com/package/helmet.

[11] D. Wilson, "Cors", NPM, 2021. [Online]. Dostupno na: https://www.npmjs.com/package/cors.

[12] D. Wilson, "Morgan", NPM, 2021. [Online]. Dostupno na: https://www.npmjs.com/package/morgan.

[13] E. Hammer, "Joi", NPM, 2021. [Online]. Dostupno na: https://www.npmjs.com/package/joi.

[14] "JWT", 2021. [Online]. Dostupno na: https://jwt.io/.

[15] "Bcrypt", NPM, [Online]. Dostupno na: https://www.npmjs.com/package/bcrypt.

[16] "Mongoose", NPM, 2021. [Online]. Dostupno na: https://www.npmjs.com/package/mongoose.

[17] "Formidable", NPM, [Online]. Dostupno na: https://www.npmjs.com/package/formidable.

[18] N. Joneja, "Google Cloud SQL", 2011.

[19] E. Garbarino, "Beginning Kubernetes on the Google Cloud Platform", 2020.

[20] T. H. King, AWS: The Ultimate Guide, 2020.

[21] "Google Cloud", 2021. [Online]. Dostupno na: https://cloud.google.com/.

POPIS I OPIS UPOTRIJEBLJENIH KRATICA

- MERN (engl. MERN stack – *MongoDB, Express.js, React.js and Node.js*) – Stog tehnologija korištene za izradu internetskih aplikacija
- MEAN (engl. MEAN - *MongoDB, Express.js, Angular.js and Node.js)* - Stog tehnologija korištene za izradu internetskih aplikacija
- MEVN stog (engl. MEVN - *MongoDB, Express.js, Vue.js and Node.js)* - Stog tehnologija korištene za izradu internetskih aplikacija
- BSON (engl. BSON - *computer data interchange format*) - Binarni oblik za predstavljanje jednostavnih ili složenih struktura podataka uključujući asocijativne nizove (poznate i kao parovi ime-vrijednost), cjelobrojno indeksirane nizove i skup osnovnih skalarnih tipova
- MQL (engl. MQL - *Marketing Qualified Lead*) - Potencijalni klijent koji je pokazao interes za ono što robna marka može ponuditi na temelju marketinških napora ili je na drugi način veća vjerojatnost da će postati kupac od ostalih potencijalnih klijenata.
- XML (engl. XML – *Extensible Markup Language*) - Jezik za označavanje je skup kodova ili oznaka koji opisuju tekst u digitalnom dokumentu
- HTML (engl. HTML – *Hypertext Markup Language*) - Standardni jezik za označavanje dokumenata dizajniranih za prikaz u internetskih pregledniku
- API (engl. *Application programming interface)* – Aplikacijsko programsko sučelje. Skup potprograma, gotovih funkcija i protokola koje programer može koristiti za kreiranje vlastitih programa.
- HTTP (engl. *Hypertext Transfer Protocol*) – Mrežni protokol za prijenos podataka kojeg koriste izvorišno i odredišno računalo za uspostavu komunikacije.
- CSS (engl. CSS - Cascading Style Sheets) - Jezik stilskog lista koji se koristi za opisivanje prezentacije dokumenta napisanog na označnom jeziku, kao što je HTML
- VSC (engl. VSC – *Visual Studio Code*) - Integrirano razvojno okruženje koje je izradio Microsoft za Windows
- NPM-om (engl. NPM – *Node Package Manager)* - Upravitelj paketa za programski jezik JavaScript

- (engl. IP – *Internet Protocol*) - Numerička oznaka kao što je 192.0.2.1 koja je spojena na računalnu mrežu koja za komunikaciju koristi internetski protokol
- SSH (engl. SSH - *Secure Shell Protocol*) - Kriptografski mrežni protokol za sigurno upravljanje mrežnim uslugama preko nezaštićene mreže
- JSON (engl. *JavaScript Object Notation*) – Jezik razmjene podataka, neovisan o platformi. Jednostavno ga je programski generirati i interpretirati. Jednako kako je jednostavno razumljiv računalima tako je razumljiv i ljudima.
- URI (engl. *Uniform resource identifier*) – Podatak koji jedinstveno određuje neki sadržaj.
- GCP (engl. *Google Cloud Platform*) – Paket usluga računalstva u oblaku
- SQL (engl. SQL - *Structured Query Language*) - Jezik specifičan za domenu koji se koristi u programiranju i dizajniran je za upravljanje podacima koji se drže u sustavu za upravljanje relacijskom bazom podataka
- CRUD (engl. CRUD - *Create, read, update and delete*) - Izrada, čitanje, ažuriranje i brisanje četiri su osnovne operacije trajne pohrane
- JWT (engl. JWT – *Jason Web Token*) - Predloženi internetski standard za stvaranje podataka s potpisom
- VM (engl.*Virtual Machine*) - Emulacija računalnog sustava
- SEO (engl. Search Engine Optimization) - proces poboljšanja kvalitete i količine prometa na internetsku stranicu ili internetsku stranicu s tražilica

SAŽETAK

Završni rad objašnjava proces izrade MERN stog aplikacije za prijenos i spremanje datoteka. Navodi tehnologije korištene za izradu aplikacije. Kratko objašnjava princip rada tehnologija i razloga korištenja navedenih tehnologija. Detaljno je objašnjen princip strukture programskog koda kao i načine korištenja aplikacije u oblaku. Posebna pozornost posvećena je na sigurnost i moderne metode korištene za izradu aplikacije Prikazani su najvažniji elementi klijentskog dijela aplikacije. Detaljno je objašnjena struktura programskog koda serverske strane te proces dodavanja aplikacije na oblak. Konačno, objašnjeni su principi rada baze podataka te zaključak rada.

Ključne riječi: baza podataka, klijentski dio, MERN stog, serverska strana

ABSTRACT

Application and implementation of MERN stack application

The final paper explains the process of creating MERN applications for file transfer and storage. Lists the technologies used to create applications. It briefly explains the principle of operation of the technology and the reasons for using this technology. The principle of the program code structure is explained in detail, as well as the ways of using cloud applications. Special attention is paid to security and modern methods used to create applications. The most important elements of the client part of applications are presented. The structure of the server program code on the side of the process of adding applications to the cloud is explained in detail. Finally, the principles of database operation and the conclusion of the work are explained.

Keywords: client side, database, MERN stack, server side

ŽIVOTOPIS

Mihael Ištvan, rođen 5. studenog 1999 u Koprivnici. Godine 2018. upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek pri Sveučilištu Josipa Jurja Strossmayera u Osijeku. Za vrijeme druge i treće godine studija obavlja poslove vanjskog suradnika kao internetski programer u poduzetništvima u inozemstvu.

Mihael Ištvan: _____