Optimizacija pristupa podacima pomoću GraphQL i SQL pogleda u React aplikaciji

Gašpar, Filip

VELEUČILIŠTE U BJELOVARU
STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

OPTIMIZACIJA PRISTUPA PODACIMA POMOĆU GRAPHQL I SQL POGLEDA U REACT APLIKACIJI

Završni rad br. 09/RAČ/2023

Filip Gašpar

Bjelovar, listopad 2023.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Filip Gašpar** JMBAG: **0314022982**

Naslov rada (tema): **Optimizacija pristupa podacima pomoću GraphQL i SQL pogleda u React aplikaciji**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Primijenjeno računarstvo**

Mentor: **Tomislav Adamović, mag. ing. el.** zvanje: **viši predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Krunoslav Husak, dipl. ing. rač., predsjednik**
2. **Tomislav Adamović, mag. ing. el., mentor**
3. **Ivan Sekovanić, mag. ing. inf. et comm. techn., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 09/RAČ/2023

U sklopu završnog rada potrebno je:

1. Odabrati i implementirati programski paket za generiranje web formi u React aplikaciji
2. Odabrati i implementirati programski paket za generiranje tabličnih pregleda podataka u React aplikaciji
3. Dohvaćati podatke primjenom GraphQL tehnologije
4. Dohvaćati podatke primjenom pogleda u SQL tehnologiji
5. Usporediti performanse GraphQL tehnologije i pogleda u SQL tehnologiji

Datum: 18.07.2023. godine

Mentor: **Tomislav Adamović, mag. ing. el.**

Zahvala

Zahvaljujem svim profesorima i studentima na Veleučilištu u Bjelovaru koji su mi omogućili lijepo visoko školsko obrazovanje. Također zahvaljujem svojoj majci i sestri na upornosti da završim studij te najviše svom ocu koji makar nije više živ, uvijek je bio uz mene.

Sadržaj

1. UVOD

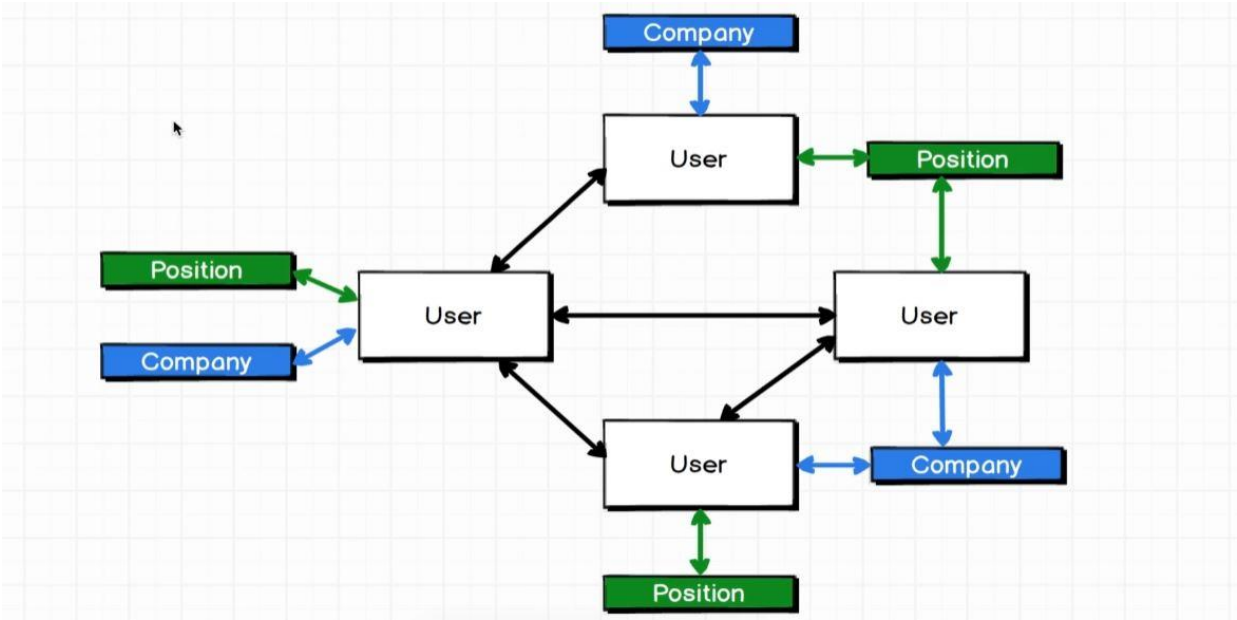
U ovom završnom radu istražuje se primjena tehnologija *GraphQL* i *SQL* Pogleda (engl. *Views*) u *React* aplikaciji. Fokus je stavljen na upotrebu *GraphQL* tehnologije, često primjenjivane u projektima koji uključuju velike baze podataka, mnogo tablica i složenih veze među njima. Također, razmatra se upotreba klasičnih *SQL* Pogleda, koji olakšavaju izradu i izvođenje složenih upita prema bazi podataka. U oba slučaja, *React* tehnologija koristi se kao sučelje za upotrebu ovih tehnologija, omogućujući usporedbu različitih načina pohrane, prikaza i ažuriranja podataka u *React* aplikaciji.

Prvi dio rada posvećen je detaljnom pregledu *GraphQL* tehnologije, uključujući njezinu primjenu, ponašanje, način rada te izvođenja upita. Nadalje, istražuje se upotreba *Apollo* tehnologije, koja omogućava integraciju *GraphQL* tehnologije u *React* aplikaciji. Detaljno se opisuje kako *Apollo* server funkcionira, kako prima i izvršava *GraphQL* upite prema bazi podataka. Slijedi poglavlje koje se bavi primjenom *GraphQL*-a na serverskom sustavu *VUBApp*, uključujući primanje ruta i aspekte koji zahtijevaju posebnu pažnju. U idućem poglavlju analizira se klijentski dio projekta, uključujući stvaranje *React* stranica, pohranu tokena i logiku koja se koristi za pozivanje ruta u projektu. Detaljno se opisuje sustav s i bez upotrebe *GraphQL* tehnologije. Na kraju, provodi se testiranje i analiza primjene *GraphQL* tehnologije i *SQL* Pogleda u radu.

2. GRAPHQL TEHNOLOGIJA

U ovom poglavlju će se opisivati GraphQL tehnologija, njezino značenje te proces stvaranja shema, upita, mutacija i slično.

GraphQL je jezik upita i sustav za kreiranje te izvođenje upita prema serveru. On predstavlja sustav tipova (engl. type system) za definiranje podataka koji se koristi za izvođenje upita, a nije ograničen na bazu podataka već se može koristiti uz postojeći kod i podatke. Prilikom kreiranja GraphQL servisa, koriste se tipovi podataka i polja unutar njih. Osnovni koncept GraphQL-a temelji se na strukturi podataka poznatoj kao grafovi. Svaki čvor u grafu predstavlja jedan entitet i može imati mnogo veza koje opisuju odnose s drugim čvorovima. Na primjer, postoji čvor "korisnik" s poljima kao što su ime, prezime, email adresa i pozicija na poslu. Također, postoji čvor "tvrtka" s poljima kao što su ime, pozicija posla i vlasnik. Primjer veza između tih čvorova može obuhvaćati situacije gdje korisnik ima prijatelja koji radi u drugoj tvrtci na istoj poziciji ili prijatelja koji radi u istoj tvrtci, ali na drugoj poziciji. Također, moguće je da korisnik ima veze s drugim korisnicima koji rade u različitim tvrtkama na različitim pozicijama.



Slika 2.1 - primjer veza između čvorova [2]

Stvarajući takve upite s ovakvim složenim vezama u relacijskim bazama podataka, proces može biti izrazito težak i kompliciran, dok GraphQL nastoji pojednostaviti rukovanje takvim

relacijama. Prilikom pozivanja upita u GraphQL-u, koriste se argumenti slično kao prilikom pozivanja funkcija, te se u njima specificiraju tražena polja. Dodatno, GraphQL omogućuje upotrebu varijabli upita, što olakšava i pojednostavljuje definiranje potrebnoh podataka prilikom izvođenja upita. Također, ključna komponenta je tip pod nazivom Root, preko kojeg se stvaraju upiti prema bazi podataka. U kontekstu GraphQL-a postoje i mutacije, koje se koriste za unos i ažuriranje podataka u bazi. Prilikom definiranja čvorova ili tipova, bitno je pažljivo razmotriti polja unutar tih čvorova te njihove tipove podataka.

U GraphQL servisu postoji mnogo različitih vrsta podataka, kao što su string, int, float i druge. Jedan čvor može imati više različitih polja, svako s drugačijim tipom podataka i potencijalno različitih duljina. Nadalje, tip podataka može biti i postojeći tip koji je korisnik prethodno definirao. Na primjer, čvor "tvrtka" može sadržavati polje "user_id" koje može biti tipa "User", koji je također čvor sa svojim vlastitim poljima. Osim toga, u GraphQL-u postoji tzv. scalar type koji se koristi za prikaz i unos JSON tipova podataka ili nizova (engl. array).

2.1 Jednostavni primjer GraphQL baze podataka

[2] Kako bi se ilustrirala jednostavna GraphQL baza podataka, pružen je primjer koji koristi JSON server za stvaranje osnovnih upita prema serveru, koristeći ga kao lokalnu bazu podataka.

U navedenom programskom kodu, mogu se uočiti dva tipa ili čvora: UserType i CompanyType.

Programski kod 2.1 – čvorovi UserType i CompanyType [2]

```
const CompanyType = new GraphQLObjectType({
    name: 'Company',
    fields: ()=> ({
        id: { type: GraphQLInt },
        name: { type: GraphQLString },
        description: { type: GraphQLString },
        users: {type: new GraphQLList(UserType),
            resolve(parentValue, args) {
                return
axios.get(`http://localhost:3000/companies/${parentValue.id}/users`)
 .then(res => res.data);
  }}
 })
});
const UserType = new GraphQLObjectType({
```

```
    name: 'User',
    fields: {
        id: { type:GraphQLInt },
        firstName: { type: GraphQLString },
        age: { type: GraphQLInt },
        company: {
            type: CompanyType,
            resolve(parentValue, args) {
                return
axios.get(`http://localhost:3000/companies/${parentValue.companyId}`)
     .then(res => res.data);
     }
   }}
});
```

CompanyType sastoji se od četiri polja: *id*, *name*, *description* i *users* koji se koristi kao poziv prema svim userima koji rade u toj tvrtci. Čvor *UserType* sastoji se također od četiri polja koja su*: id, firstName, age* i *companyId* koji predstavlja tvrtku u kojoj je taj korisnik. Prije toga je bitno odabrati koje će se vrste podataka koristiti poput Int, String itd.

Programski kod 2.2 – vrste podataka u GraphQL tehnologiji [2]

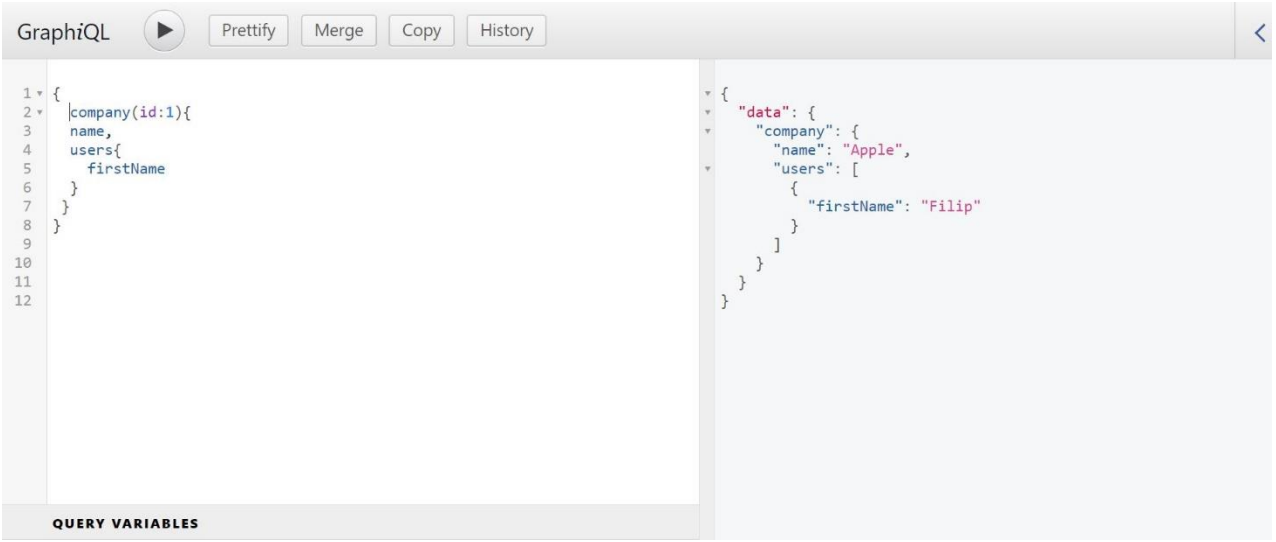
```
const graphql = require('graphql');
const axios = require('axios');


const {
GraphQLObjectType,
GraphQLString,
GraphQLInt,
GraphQLSchema,
GraphQLList,
GraphQLNonNull
}= graphql;
```

Nakon toga kreira se tzv. *RootQuery* čvor koji predstavlja tkz. upite koji se zovu query.

Programski kod 2.3 – RootQuery čvor [2]

```
const RootQuery = new GraphQLObjectType({
    name: 'RootQueryType',
    fields: {
        user: {
            type: UserType,
            args: { id: { type: GraphQLInt } },
            resolve(parentValue, args) {
                return axios.get(`http://localhost:3000/users/${args.id}`)
                    .then(res => res.data);
            }
        },
        company:{
            type: CompanyType,
            args: { id: { type: GraphQLInt } },
            resolve(parentValue, args) {
                return axios.get(`http://localhost:3000/companies/${args.id}`)
                    .then(res => res.data);
            }
        }
    }
});
```

Preko njega se mogu stvarati upiti koristeći prethodno napravljene čvorove. Jedan od tih upita dohvaća sve korisnike koje rade u nekoj određenoj tvrtci.



Slika 2.2 - primjer upita

Rezultat je u JSON formatu jer je JSON standard za slanje podataka putem Interneta. U prvom JSON objektu postoji polje "name", kao i polje "users", koje se dalje dekomponira na svoja polja "firstName" i "age". Također je moguće izvesti i obratni upit koji dohvaća informacije o korisniku i tvrtki u kojoj radi. Za stvaranje novih korisnika, ažuriranje postojećih i brisanje postojećih korisnika, nužno je izvršiti mutaciju (engl. mutation).

Programski kod 2.4 – Mutation čvor [2]

```
const mutation = new GraphQLObjectType({
    name: 'Mutation',
    fields: {
        addUser: {
            type: UserType,
            args: {
                firstName: { type: new GraphQLNonNull( GraphQLString) },
                age: { type: new GraphQLNonNull( GraphQLInt) },
                companyId: { type: GraphQLInt }
            },
            resolve(parentValue, { firstName, age }) {
                return axios.post('http://localhost:3000/users', { firstName, age
})
                    .then(res => res.data);
            }
        },
        deleteUser: {
            type: UserType,
            args: { id: { type: new GraphQLNonNull( GraphQLInt) } },
            resolve(parentValue, { id }) {
                return axios.delete(`http://localhost:3000/users/${id}`)
                    .then(res => res.data);
            }
        },
     editUser: {
            type: UserType,
            args: {
                id: { type: new GraphQLNonNull( GraphQLInt) },
                firstName: { type: GraphQLString },
                age: { type: GraphQLInt },
                companyId: { type: GraphQLInt }
            },
```

```
            resolve(parentValue, args) {
                return axios.patch(`http://localhost:3000/users/${args.id}`, args)
                    .then(res => res.data);
            }
        }
    }
});
```

U kodu je prikazana mutacija za kreiranje novog korisnika, ažuriranje i brisanje postojećeg korisnika. Primjer poziva mutacije za kreiranje korisnika može se vidjeti na slici 2.3:

Slika 2.3 - primjer unosa korisnika

Također bi se mogle koristiti i varijable upita kao argumenti, ali za prikaz jednostavnog primjera nije korišteno. Mutacija za ažuriranje postojećeg korisnika na slici 2.4:

Slika 2.4 - primjer ažuriranja korisnikovih podataka

U samoj mutaciji postoji mogućnost za stvaranje upita kao što je na slici prikazano. Brisanje korisnika na slici 2.5:

Slika 2.5 - brisanje korisnika

U njoj je potrebno poslati id korisnika kojeg se briše.

Na kraju, potrebno je izvesti (engl. *export*) cijelu tu shemu.

Programski kod 2.5 – izvezanje sheme [2]

```
module.exports = new GraphQLSchema({
    query: RootQuery,
    mutation: mutation
 })
```

Za kreiranje jednostavnog servera koristi se Node JS i Express JS skupa s GraphQL paketom. Potrebno je instalirati pakete graphql, nodemon, express-graphql, axios, express i json-server kao baza podataka. U sljedećem programskom kodu je prikazano kreiranje jednostavnog servera:

Programski kod 2.6 – kreiranje jednostavnog servera [2]

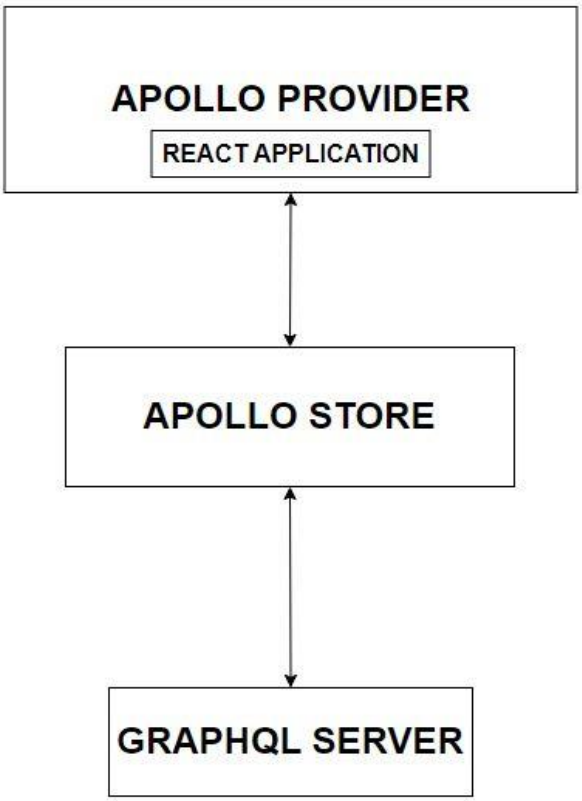
```
const express = require('express');
const expressGraphQL = require('express-graphql').graphqlHTTP;
const schema = require('./schema/schema');

const app = express();

app.use('/graphql',expressGraphQL({
    schema: schema,
    graphiql: true
}));
app.listen(4000, () => console.log('Server is running...'));
```

U njemu su dodani paketi express, express-graphql i shema koja je bila napravljena za ovaj primjer. Server se inicijalizira te se postavi korištenje *graphiql* sustava za izvršavanje svih upita kreiranim u tom primjeru. U ovome završnom radu koristit će se taj isti sustav za prikaz testiranja GraphQL upita. Zatim se server podešava za prihvaćanje komunikacije na *portu* 4000. U sljedećem poglavlju objasnit će se kako radi Apollo server koji se koristi kao *middleman*“ između React aplikacije i samog *GraphQL*-a.

3. APOLLO CLIENT

U ovom poglavlju bit će objašnjeno korištenje Apollo Clienta na klijentskom dijelu programa.

[3] Apollo Client je JavaScript biblioteka koja omogućuje upravljanje lokalnim i udaljenim podacima (engl. remote) pomoću GraphQL tehnologije. On djeluje kao posrednik između React aplikacije i GraphQL servera. Sastoji se od tri dijela: Apollo Provider, na kojem se smjestila React aplikacija, Apollo Store i GraphQL server.



Slika 3.1 - Apollo client dijelovi [2]

[2] Apollo Store predstavlja manji posrednik između Apollo Providera i GraphQL servera. On ostvaruje izravnu komunikaciju s GraphQL serverom i, kao što mu samo ime sugerira, skladišti podatke dobivene sa servera. Nakon toga, ti podaci se šalju prema Apollo Provideru i smještaju na klijentskoj strani aplikacije. Apollo Store se ističe svojom robusnošću i apstraktnošću, što ga čini primjenjivim u različitim vrstama aplikacija, neovisno o okruženju (frameworku) koje se koristi na klijentskoj strani, bilo da je to React, Vue.JS, React Native i slično.

Posrednik između Apollo Storea i React aplikacije u ovom radu predstavlja Apollo Provider. Apollo Provider djeluje kao opskrbljivač podacima za React aplikaciju. On preuzima podatke dobivene od Apollo Storea i integrira ih u React aplikaciju kako bi aplikacija znala kako prikazati podatke dobivene putem GraphQL servera.

Prikaz jednostavnog stvaranja Apollo Providera u React aplikaciji je u sljedećem programskom kodu. Za početak je potrebno stvoriti novi Apollo Client te se taj *client* prosljedi Apollo Provider elementu.

Programski kod 3.1 – stvaranje novog Apollo Clienta

```
const client = new ApolloClient({
  uri: 'http://localhost:3000/graphql',
  cache: new InMemoryCache()
});

const Root = () => {
  return (
    <div>
    <Provider store={store}>
      <PersistGate loading={null} persistor={persistor}>
        <ApolloProvider client={client}>
          …
        </ApolloProvider>
      </PersistGate>
    </Provider>
    </div>
  );
};
```

Izgled upita i izvršavanje upita prema GraphQL serveru je opisan u sljedećem programskom kodu.

Programski kod 3.2 – izgled i izvršavanje upita prema GraphQL serveru

```
const FETCH_COURSE_PROFESSORS_QUERY = gql`
  query FetchCourseProfessors($token: String!) {
    fetchCourseProfessors(token: $token)
  }
`;
```

```
const { loading, data } = useQuery(FETCH_COURSE_PROFESSORS_QUERY, {
    variables: { token },
  });
 const [register, { loading, error }] = useLazyQuery(REGISTER_QUERY, {
    onCompleted: (response) => {
      …
    },
  });

  const [addDocumentResponse, { loading }] = useMutation(addResponse, {
    onCompleted: (response) => {
              …
    },
  });
```

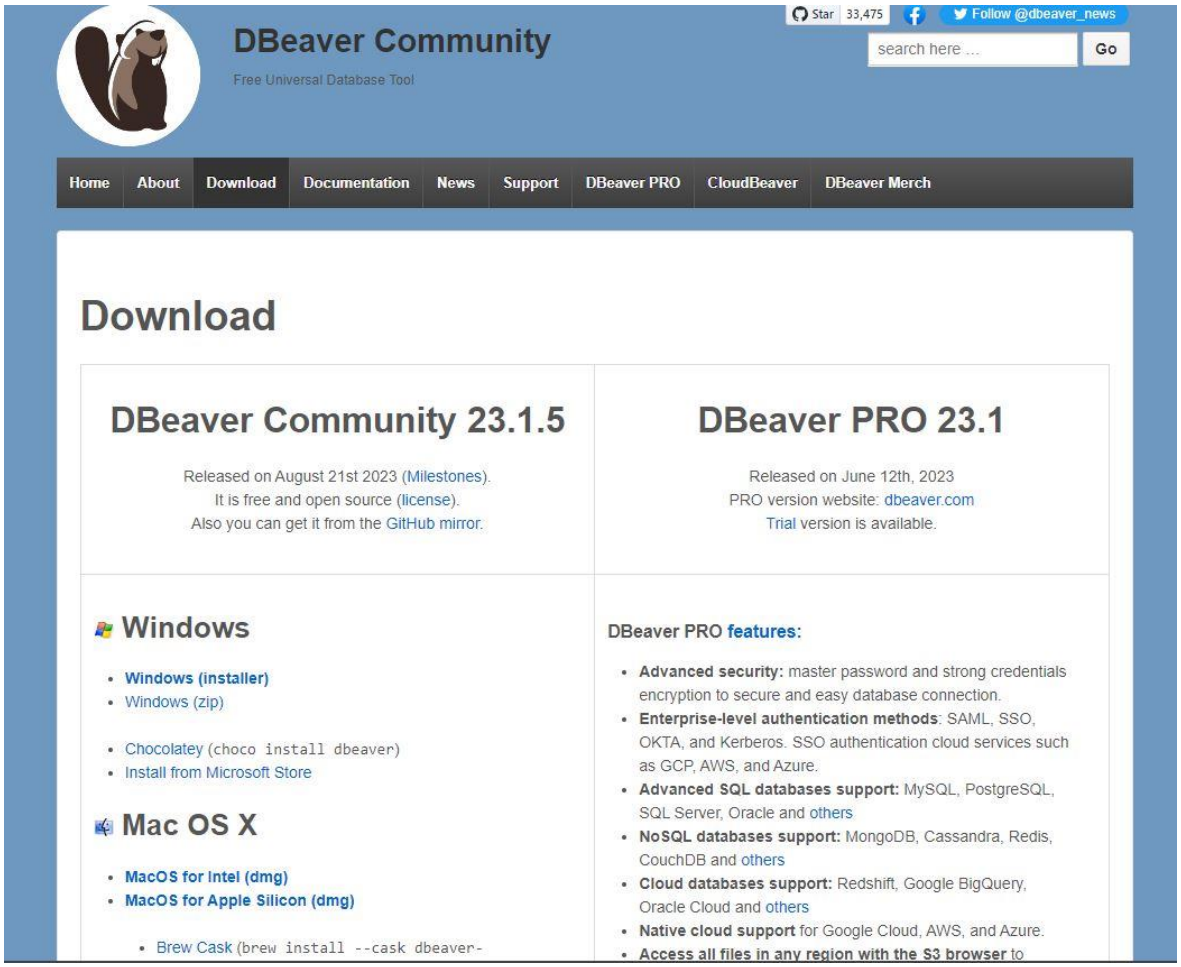
U programskim kodovima su prikazane funkcije useQuery, useLazyQuery i useMutation, koje potječu iz biblioteke Apollo Client. Funkcija useQuery koristi se kada je potrebno izvršiti određeni upit svaki put pri osvježavanju web stranice. S druge strane, useLazyQuery koristi se kada je upit potrebno izvršiti samo jednom, na primjer, prilikom pritiska na gumb. useMutation se primjenjuje za mutacije podataka, tj. za unos i ažuriranje podataka. Za potrebe testiranja Apollo Clienta, potrebno je instalirati sljedeće Node.js pakete: @apollo/client, @apollo/react-hooks i graphql. Paketi @apollo/client i @apollo/react-hooks koriste se za komponente klijenta, poput Apollo Providera, Apollo Storea i React hookova, dok paket graphql služi za kreiranje i izvršavanje GraphQL upita pomoću riječi "gql".

U narednom poglavlju bit će objašnjena baza podataka korištena u završnom radu, s posebnim naglaskom na middleware dio s GraphQL-om te upitima koji su se koristili u projektu.

4. BAZA PODATAKA I MIDDLEWARE

U ovom poglavlju bit će detaljno opisana baza podataka koja je upotrijebljena u završnom radu, zajedno s middleware-om.

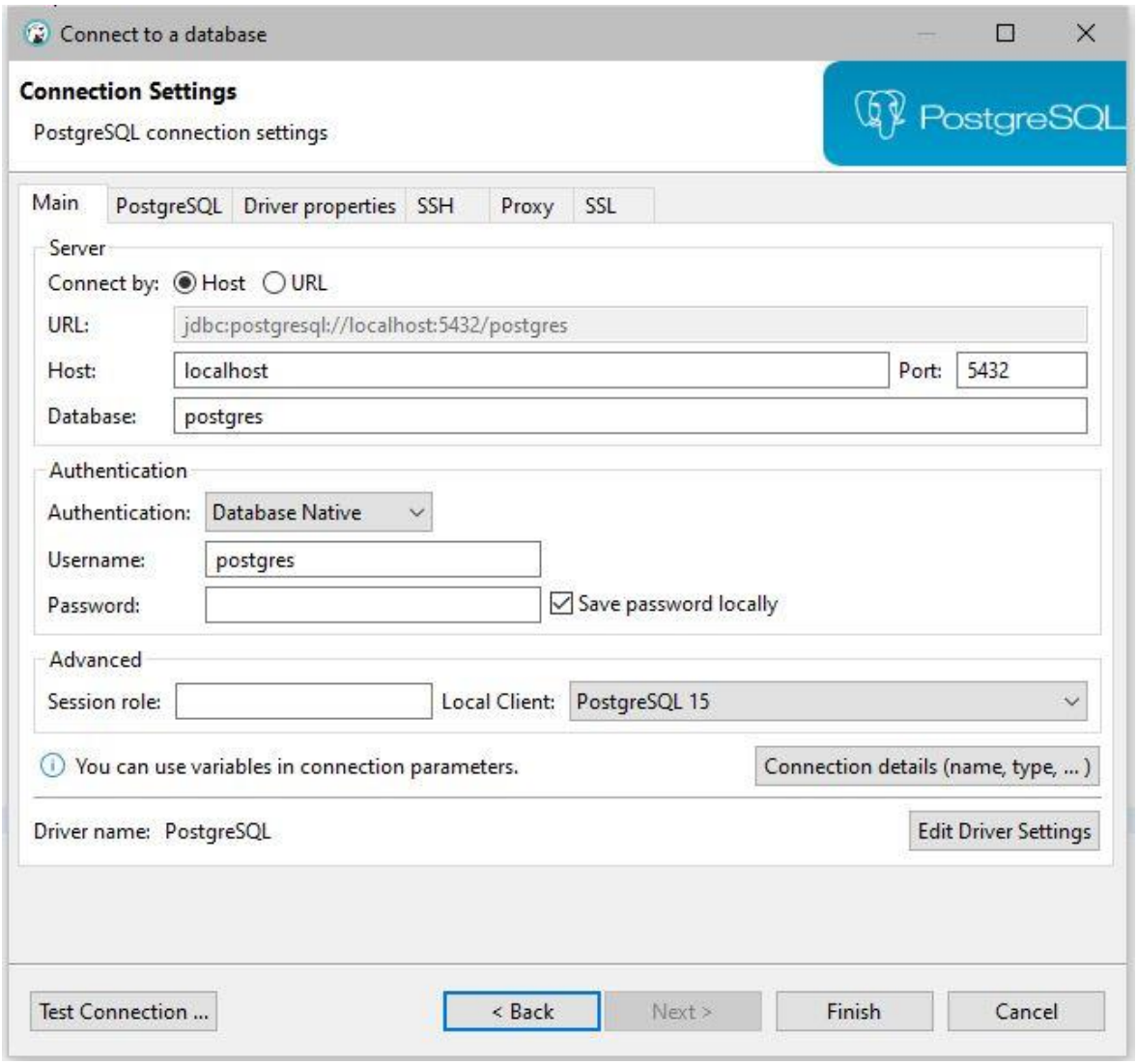
Prije nego što se krene s izradom baze podataka, preporučuje se instaliranje programa DBeaver kako bi se olakšalo upravljanje bazom podataka. Postupak instalacije programa DBeaver sastoji se od posjete službenoj web stranici te odabira opcije za instalaciju koja je kompatibilna s operativnim sustavom na kojem će se program koristiti.



Slika 4.1 - instalacija DBeaver [4]

Nakon instalacije, slijede se upute za postavljanje korisničkog profila. Postavke uključuju određivanje korisnika kao "postgres" i odabir lozinke prema vlastitim preferencijama.

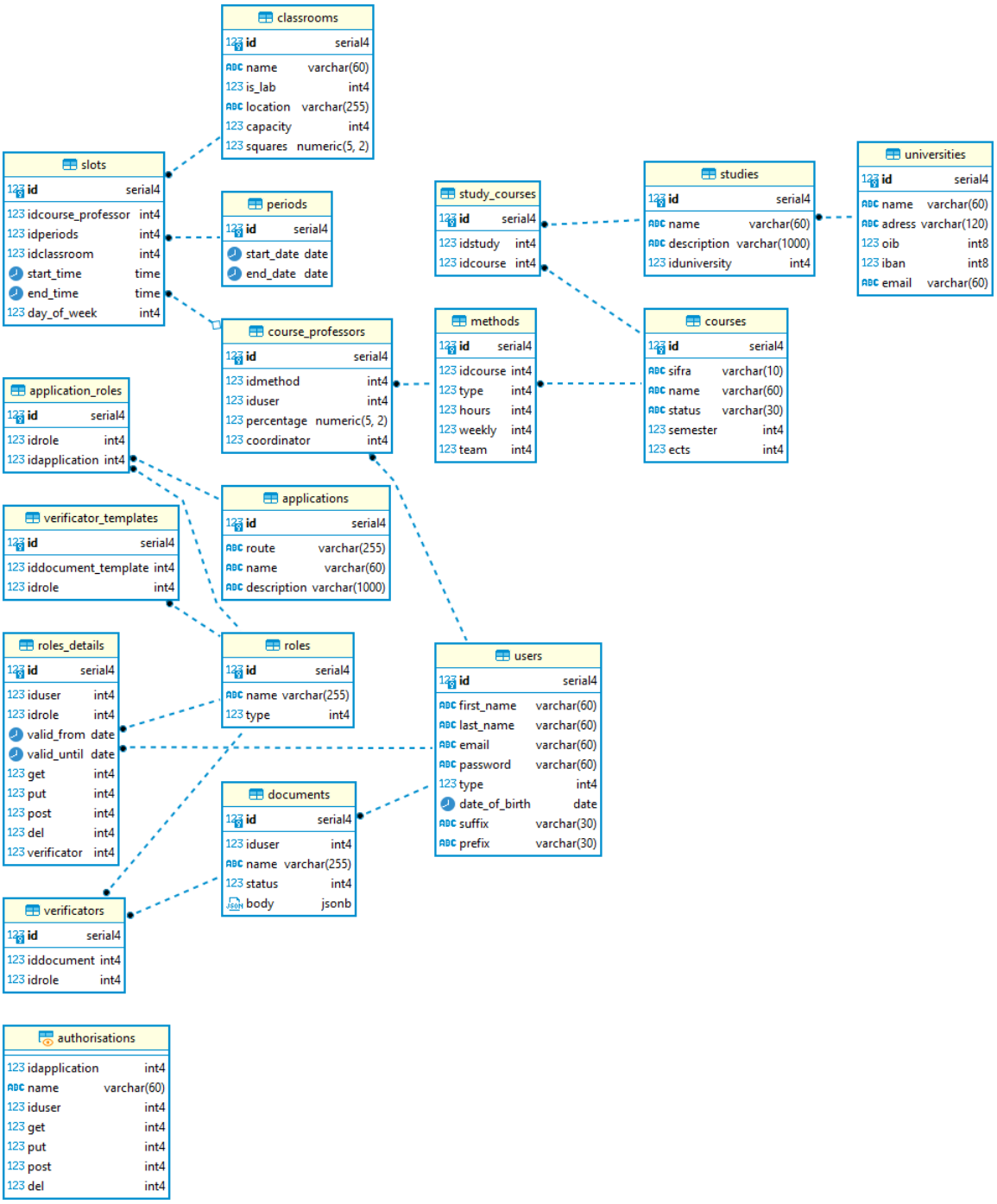
Nakon toga, korisnik kreira novu vezu s bazom podataka putem programa DBeaver. U ovom završnom radu, korišten je lokalni (localhost) domaćin, s obzirom da nije bilo potrebe za povezivanjem s vanjskim serverom. Postavljanje veze s bazom obavlja se prema navedenim uputama, a nakon toga se može započeti s izgradnjom baze podataka.



Slika 4.2 - konekcija na bazu

4.1 Baza podataka

Izgled baze podataka napravljene za ovaj završni je prikazana prema slici 4.3:



Slika 4.3 - VUBApp baza podataka

Baza podataka koja je korištena u ovom završnom radu preuzeta je iz već postojeće baze za VUBApp aplikaciju. Središnja fokusna točka ove baze podataka je tablica "users", u kojoj se čuvaju različiti podaci o korisnicima, uključujući njihova imena, prezimena, e-mail

adrese, lozinke, tip korisnika, datum rođenja, i drugo. Tip korisnika je numerički podatak koji određuje vrstu korisnika, primjerice, tip korisnika 2 označava profesora, dok tip korisnika 1 označava administratora, itd.

Sljedeća važna tablica je "roles", koja sadrži tri stupca: ID, name i type. Ova tablica služi za pohranu različitih uloga koje korisnici mogu imati u raznim aplikacijama. Ovisno o aplikaciji, neki korisnici mogu imati administratorske ovlasti, dok u drugim aplikacijama mogu imati obične korisničke ovlasti. Tablica "roles_details" služi kao poveznica između tablica "users" i "roles". U njoj se nalaze stupci "iduser" i "idrole", koji su strani ključevi, tj. poveznice prema svojim tablicama. Ova tablica bilježi koji korisnik ima koje uloge u različitim aplikacijama, omogućujući tako precizno definiranje korisničkih ovlasti za svaku aplikaciju. Primjerice, tablica "roles_details" sadrži podatke o tome ima li korisnik prava za dohvaćanje, unos, ažuriranje i druge operacije nad podacima u određenoj aplikaciji. Značajna tablica u bazi je i "applications", koja sadrži informacije o svim aplikacijama, uključujući njihova imena i rute do njih.

U ovom završnom radu postoje dvije aplikacije: jedna je namijenjena za prikaz i unos dokumenata u okviru obrazovne institucije, dok je druga aplikacija za prikaz, unos i ažuriranje informacija o kolegijima na fakultetu. Sustav evidentira koji profesor je nositelj kolegija, a koji izvođač, te druge relevantne informacije. Posljednja bitna tablica u bazi je "application_roles", u kojoj se čuvaju uloge za svaku aplikaciju. U svrhu rada s oba tipa aplikacija, ključan je SQL Pogled (engl. SQL View) pod nazivom "authorisations". Ovaj Pogled pohranjuje identifikacijske brojeve aplikacija, njihova imena, identifikacijske brojeve korisnika te podatke o tome imaju li korisnici prava za dohvaćanje, unos, ažuriranje i druge operacije nad podacima u aplikacijama koje su im dodijeljene.

Upit za izradu tog Pogleda prikazan u sljedećem programskom kodu:

Programski kod 4.1 – SQL Pogled authorisations

```
CREATE VIEW authorisations AS
    SELECT a.id idapplication, a.name, rd.iduser, rd.get, rd.put, rd.post, rd.del
    FROM applications AS a
    JOIN application_roles AS ar ON ar.idapplication = a.id
    JOIN roles_details AS rd ON rd.idrole = ar.idrole
    WHERE rd.valid_from <= current_date
AND rd.valid_until >= current_date;
```

Za rad s prvom aplikacijom koja služi za prikaz i unos dokumenata u obrazovnoj instituciji, koristi se tablica "documents". U ovoj tablici pohranjuju se svi rezultati nakon što korisnik popuni određeni dokument. Ovi rezultati uključuju ime dokumenta, status dokumenta, korisnika koji ga je ispunio i JSON podatke dobivene prilikom ispunjavanja dokumenta. Dokumenti se pohranjuju lokalno na serveru, u ovom radu to je na lokalnom stroju.

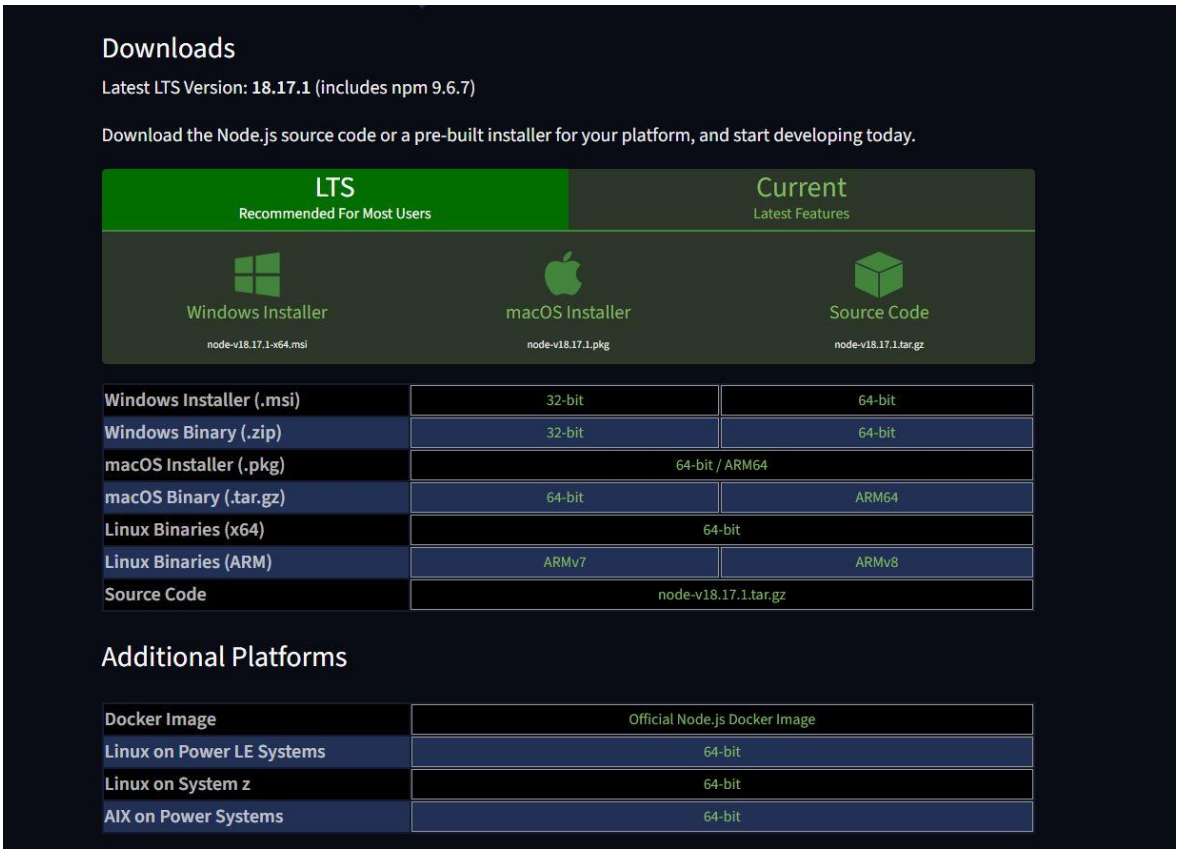
Za rad s drugom i posljednjom aplikacijom potrebne su sljedeće tablice: "courses", "methods", "slots", "periods", "classrooms", "study_courses", "studies", "universities" i "course_professors". U tablici "courses" pohranjuju se svi kolegiji u obrazovnoj ustanovi, zajedno s njihovim imenima, šiframa, semestrima u kojima se izvode, i drugim relevantnim podacima. Tablica "methods" koristi se za spremanje podataka o vrstama nastave koje kolegij može imati, kao što su predavanja, laboratorijske vježbe, auditorne vježbe i slično. Za svaku vrstu nastave zapisuje se trajanje i informacija o tome izvodi li se ta vrsta nastave u grupama.

Tablica "universities" služi za prikaz informacija o fakultetima, uključujući njihova imena, adrese, OIB-e i slično. "Studies" tablica sadrži informacije o studijima na fakultetu, uključujući njihova imena, opise i pripadnost određenom fakultetu. "Study_courses" tablica se koristi za povezivanje predmeta s određenim studijem. Tablica "periods" pohranjuje informacije o trajanju pojedinih nastavnih perioda, dok se u tablici "classrooms" nalaze podaci o imenima dvorana, njihovoj lokaciji i tome jesu li prikladne za laboratorijske vježbe i slično.

Tablica "course_professors" sadrži informacije o tome koji profesor drži koji kolegij i koju vrstu nastave. Osim toga, definira se postotak predmeta koji profesor izvodi te njegova uloga kao nositelja ili izvođača predmeta. Posljednja tablica, "slots", koristi se za spremanje rasporeda predavanja, uključujući informacije o trajanju, dvorani, tjednu nastave i slično. Ključne tablice korištene u drugoj aplikaciji su "course_professors" i "methods".

4.2 Izrada middleware dijela

Za izradu middleware-a prvo je potrebno instalirati NodeJS okruženje na računalo. Instalacijski postupak provodi se posjetom službenoj web stranici NodeJS okruženja i instaliranjem odgovarajuće verzije za operativni sustav koji se koristi. U ovom završnom radu korišten je Windows operativni sustav.



Slika 4.4 - NodeJS instalacija [5]

Nakon instalacije NodeJS-a, stvara se direktorij za projekt te se izvršava naredba "npm init" koja generira datoteku "package.json". Za izradu Node JS middleware servera, koriste se NPM paketi kao što su express, bcryptjs, jsonwebtoken, dotenv, pg, ajv, ajv-formats, express-graphql, axios, cors i nodemon. Za ovaj završni rad, ključni paketi su express-graphql, axios i cors. Prvi korak u izradi middleware servera je uspostavljanje veze s bazom podataka. Podaci potrebni za uspostavljanje veze čuvaju se u posebnoj konfiguracijskoj datoteci. U toj datoteci nalaze se informacije o korisniku, domaćinu, imenu baze podataka kojoj se pristupa i lozinki za pristup.

Programski kod 4.2 – pristup bazi podataka

```
const { Client } = require('pg');
const credentials = require('./credentials');
const client = new Client(credentials);

client.connect((err) => {
    if (err) {
        console.error('Error connecting to PostgreSQL database', err.stack);
    }
    else {
        console.log('Connected to PostgreSQL database');
    }
});

async function query(text, params) {
    const res = await client.query(text, params);
    return res;
}

module.exports = {
    query
};
```

Middleware do GraphQL dijela sastoji se od nekoliko dijelova. *JSONWebToken* za izradu tokena kako bi samo autentificirani korisnici imali pristup svim rutama.

Programski kod 4.3 – izrada JSONWebToken tokena

```
const jwt = require('jsonwebtoken');
require('dotenv').config();

const jwtSecret = process.env.JWT_SECRET;
const jwtExpiration = '24h';

function generateToken(user) {
    const payload = {
        id: user.id,
        email: user.email
    };
    return jwt.sign(payload, jwtSecret, { expiresIn: jwtExpiration });
}
```

```
function verifyToken(token) {
    return jwt.verify(token, jwtSecret);
}

module.exports = {
    generateToken,
    verifyToken
};
```

Ruter u kojem su spremljene rute za korištenje. Primjer jedne rute.

Programski kod 4.4 – primjer jedne rute na serveru

```
const express = require('express');
const router = express.Router();
const auth = require('../authorization');
const courseProfessorsController =
require('../controllers/courseProfessorsController');

// GET /api/courseProfessors
router.get('/', auth.authorize, courseProfessorsController.getCourseProfessors);

module.exports = router;
```

Kontroler datoteke u kojem se dohvaća servis za pristup bazi podataka. Primjer jedne funkcije u kontroleru.

Programski kod 4.5 – primjer jednog kontrolera

```
class UsersController {
    static async register(req, res) {
        if (!valid(req.body)) {
            return res.status(400).json({ message: valid.errors });
        }

        const { email, password } = req.body;
        try {
            const user = await usersService.getUserByEmail(email);
            if (user) {
                return res.status(409).json({ message: 'User already exists' });
 }
```

```
            const newUser = await usersService.createUser(email, password);
            const token = config.generateToken(newUser);
            const role = await usersService.createUserRole(newUser.id);
            res.json({
                data: {
                    user: newUser,
                    token: token,
                    role: role
                }
            });
        }
        catch (err) {
            console.log(err);
            res.status(500).json({ message: 'Internal server error' });
        }
    }

}

module.exports = UsersController;
```

Servis u kojem se direktno izvršava kod na bazi podataka i ostale datoteke. Primjer jedne funkcije servisa.

Programski kod 4.6 – primjer jedne funkcije servisa koja izvršuje SQL upit

```
class SchemasService {
    static async saveResponse(iduser, name, status, body) {
        const query = 'INSERT INTO documents (iduser, name, status, body) VALUES
($1, $2, $3, $4)';
        const values = [iduser, name, status, body];
        await db.query(query, values);
    }

} module.exports = SchemasService;
```

Dio tog middleware-a je izrađen od strane drugog studenta koji također sudjeluje u razvoju VUBApp aplikacije, na kojoj se temelji ovaj završni rad.

Datoteka potrebna za izradu GraphQL dijela middleware-a naziva se "schema.js". U njoj se nalaze varijable koje koriste funkcije axios i express-graphql za rad. GraphQL baza

podataka organizirana je tako da sadrži pet tipova čvorova: UserType, DocumentTemplateType, DocumentsType, GraphQLJSON, RootQuery i Mutation.

Programski kod 4.7 – GraphQL čvorovi

```
const UsersType = new GraphQLObjectType({
  name: 'Users',
  fields: () => ({
    id: { type: GraphQLInt },
    first_name: { type: GraphQLString },
    last_name: { type: GraphQLString },
    email: { type: GraphQLString },
    password: { type: GraphQLString },
    type: { type: GraphQLInt },
    date_of_birth: { type: GraphQLString },
    suffix: { type: GraphQLString },
    prefix: { type: GraphQLString },
    token: { type: GraphQLString }
  }),
});
const GraphQLJSON = new GraphQLScalarType({
  name: 'JSON',
  description: 'The `JSON` scalar type represents JSON objects',
  serialize(value) {
    return value;
  },
});
const DocumentTemplateType = new GraphQLObjectType({
  name: 'DocumentTemplate',
  fields: () => ({
    id: { type: GraphQLInt },
    name: { type: GraphQLString },
    type: { type: GraphQLInt },
    body: { type: GraphQLJSON },
  }),
});



const DocumentType = new GraphQLObjectType({
  name: 'Document',
```

```
  fields: () => ({
    id: { type: GraphQLInt },
    iduser: { type: GraphQLInt },
    idtemplate: { type: GraphQLInt },
    name: { type: GraphQLString },
    status: { type: GraphQLInt },
    body: { type: GraphQLJSON },
  }),});
```

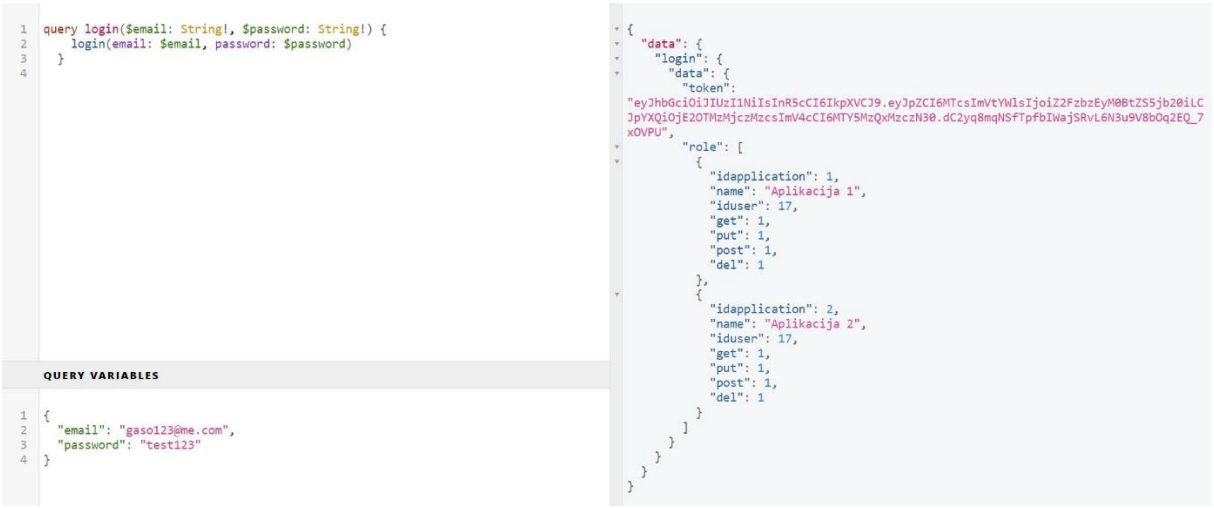
U završnom radu se najčešće koriste čvorovi *GraphQLJSON*, *RootQuery* i *Mutation*.

GraphQLJSON je skalarnog tipa, savršen za slanje i primanje JSON objekata koje šalju rute. Prvo polje odnosno upit u *RootQuery* čvoru je *login* upit.

Programski kod 4.8 – polje login

```
login: {
      type: GraphQLJSON,
      args: {
        email: { type: GraphQLString },
        password: { type: GraphQLString }
      },
      resolve(parent, args) {
        const loginData = {
          email: args.email,
          password: args.password
        };
        return axios.post('http://localhost:3000/api/users/login', loginData, {
          auth: {
            username: args.email,
            password: args.password
          }
        }).then((res) => {
          return res.data;
        }).catch((err) => {
          console.log(err);
        });
      }
    },
```

Čvor je tipa *GraphQLJON* i prima dva parametra, e-mail i lozinku te na temelju tih podataka zove rutu */api/users/login* koja *mu* šalje podatke prikazane na slici 4.5.



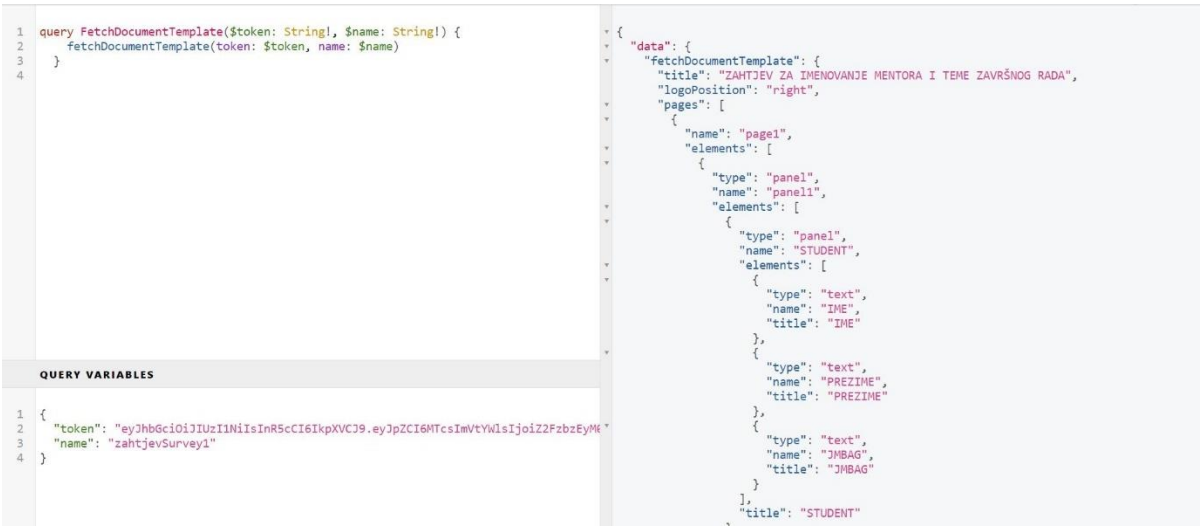
Slika 4.5 - primjer upita za prijavu

To je upit za Pogled *authorisations*, koji dohvaća token za tog korisnika i aplikacije na kojima ima ovlasti. Sljedeće polje je *fetchDocumentTemplate*, koje se koristi za dohvaćanje svih dokumenata spremljenih na serveru.

Programski kod 4.9 – polje fetchDocumentTemplate

```
fetchDocumentTemplate: {
      type: GraphQLJSON,
      args: {
        token: { type: GraphQLNonNull(GraphQLString) },
        name: { type: GraphQLString, defaultValue: null },
      },
      resolve(parent, args) {
        const config = {
          headers: { Authorization: `Bearer ${args.token}` },
          params: { name: args.name },
        };
        const name = args.name;
        return axios
        .get('http://localhost:3000/api/schemas/', { ...config, data: { name}})
        .then((response) => {
          return response.data;
        })
          .catch((error) => {
            console.log(error);
      throw new Error('Failed to fetch documents: ' + error.message);}); },},
```

Parametri koji se šalju su token, i ako je potrebno samo jedan dokument dohvatiti ime tog dokumenta. Bez slanja tog imena, ruta dohvaća sve dostupne dokumente na serveru. Primjer izvršavanje tih GraphQL upita su prikazani na slikama 4.6 i 4.7:



Slika 4.6 - izvršavanje upita s imenom

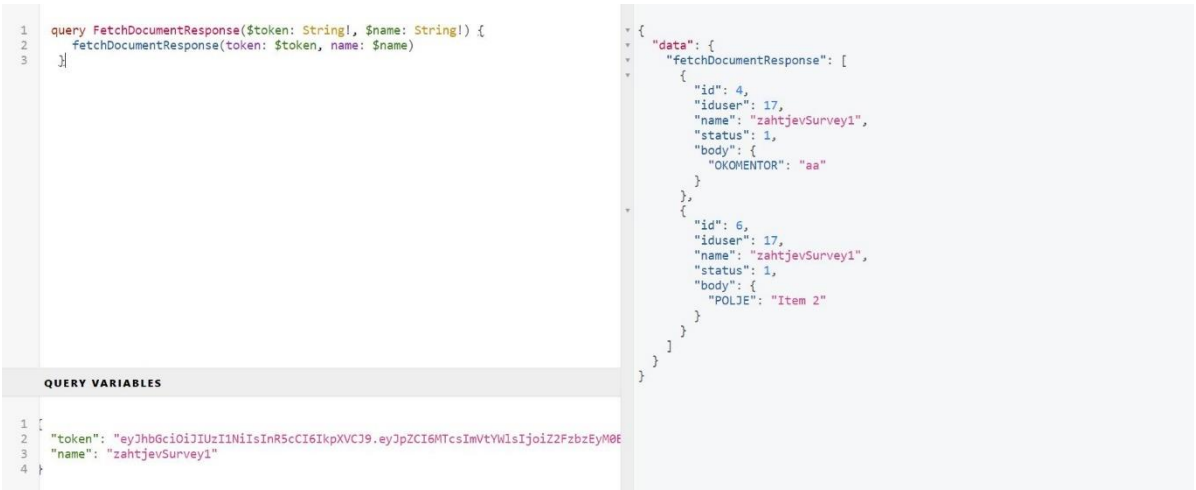
Slika 4.7 - izvršavanje upita bez imena

Sljedeće polje je *fetchDocumentResponse*, taj upit dohvaća sve rezultate spremljenje za određenu datoteku, parametri koji se šalju su token i ime tog dokumenta.

Programski kod 4.10 – polje fetchDocumentResponse

```
fetchDocumentResponse: {
      type: GraphQLJSON,
      args: {
        token: { type: GraphQLNonNull(GraphQLString) },
        name: { type: GraphQLNonNull(GraphQLString) },
      },
resolve(parent, args) {
        const config = {
          headers: { Authorization: `Bearer ${args.token}` },
          params: { name: args.name },};
     const name = args.name;
        return axios
         .get('http://localhost:3000/api/schemas/response', {...config,data:{name
} })
          .then((response) => {
            return response.data;})
          .catch((error) => {
            console.log(error);
            throw new Error('Failed to fetch documents: ' + error.message);
          });
      }
    },
```

Izgled izvršavanja tog upita je prema slici 4.8:



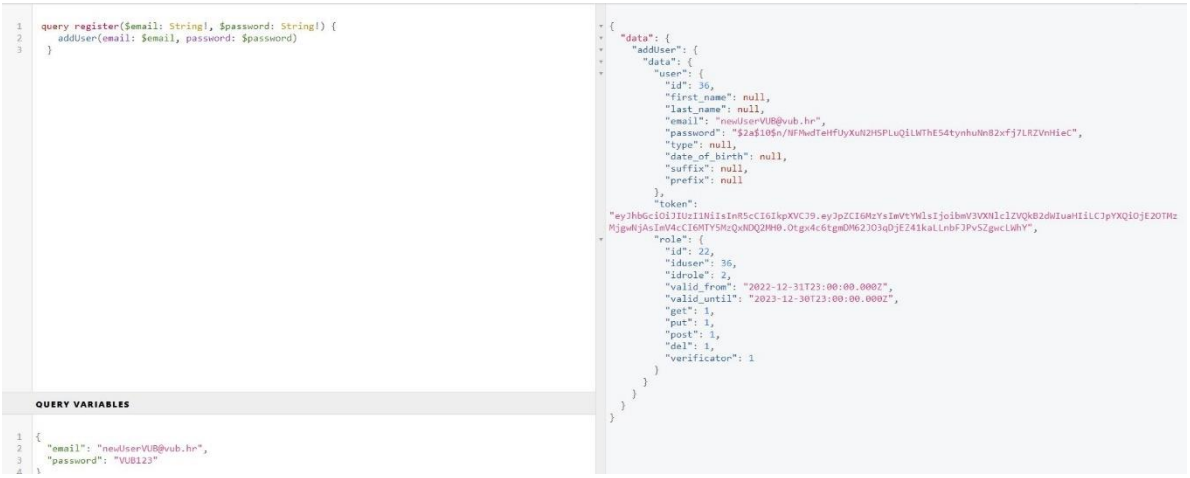
Slika 4.8 - izvršavanje upita fetchDocumentResponse

Sljedeće polje je *addUser*. Razlog zašto to polje nije npr. u mutaciji je zbog toga što vraća podatke nazad pri registraciji korisnika te radi olakšanja na klijentskom dijelu izrade projekta.

Programski kod 4.11 – polje addUser

```
addUser: {
  type: GraphQLJSON,
  args: {
    first_name: { type: GraphQLString },
    last_name: { type: GraphQLString },
    email: { type: GraphQLString },
    password: { type: GraphQLString },
    type: { type: GraphQLInt },
    date_of_birth: { type: GraphQLString },
    suffix: { type: GraphQLString },
    prefix: { type: GraphQLString }
  },
  resolve(parent, args) {
    const userData = {
      first_name: args.first_name,
      last_name: args.last_name,
      email: args.email,
      password: args.password,
      type: args.type,
      date_of_birth: args.date_of_birth,
      suffix: args.suffix,
      prefix: args.prefix
    }

    return axios.post('http://localhost:3000/api/users/register', userData)
      .then((res) => {
        console.log(res.data);
        return res.data;
      }).catch((err) => {
        console.log(err);
      });
  }
},
```

U njemu se nalaze svi podatci potrebni izradu korisnika po tablici *users* u bazi podataka. Izgled izvršavanja tog upita prema slici 4.9



Slika 4.9 - registracija korisnika

Kao što je vidljivo na slici, potrebno je unijeti samo email i lozinku budući da je ovo običan korisnik. To polje nije u čvoru *mutation* zbog toga što je bilo komplikacija na klijentskom dijelu projekta. Sljedeće polje je *addDocumentTemplate* koje ima isti problem kao i *addUser* u vezi s mutacijom.

Programski kod 4.12 – polje addDocumentTemplate

```
addDocumentTemplate: {
      type: GraphQLJSON,
      args: {
        token: { type: GraphQLNonNull(GraphQLString) },
        name: { type: GraphQLNonNull(GraphQLString) },
        body: { type: GraphQLNonNull(GraphQLJSON) },
      },
      resolve(parent, args) {
        const config = {
          headers: { Authorization: `Bearer ${args.token}` },
          params: { name: args.name ,
            response: args.body},
        };
        const name = args.name;
        const body = args.body;
        console.log(body);
        return axios
          .post('http://localhost:3000/api/schemas/', { name, body }, config)
          .then((response) => ({ data: response.data }))
          .catch((error) => {
            console.log(error);
            throw new Error('Failed to add document: ' + error.message);
```

```
});}},
```

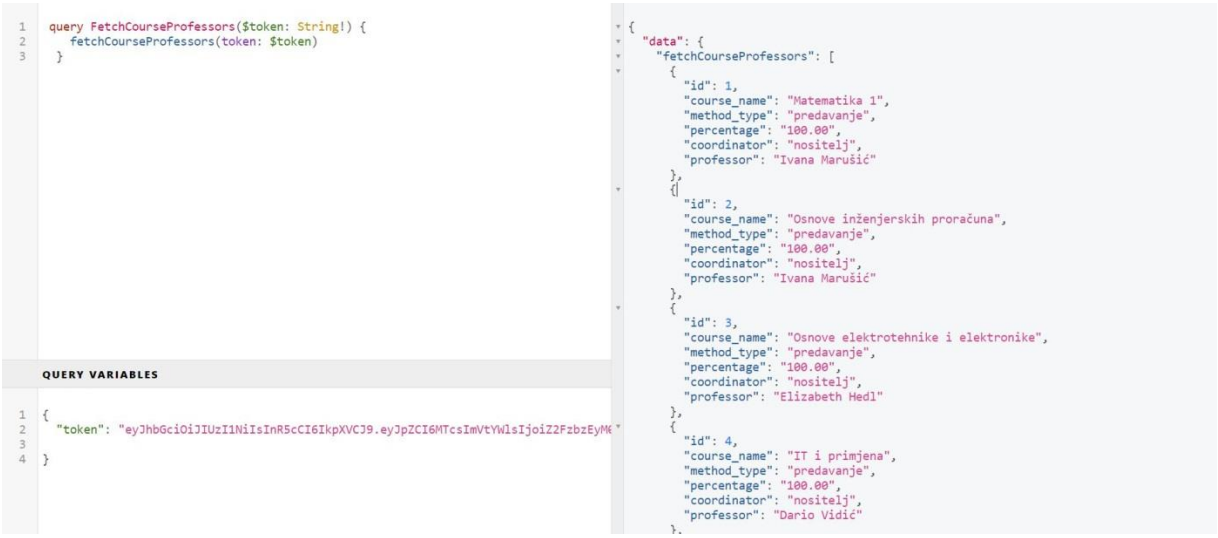
Taj upit koristi se za unos dokumenta koji se sprema na server. Potrebni parametri za njegov rad su token, ime dokumenta i sami SurveyJS JSON model dokumenta. Ovi upiti čine sve RootQuery upite potrebne za izradu aplikacije koja omogućava unos i prikaz rezultata dokumenata u okviru fakulteta. Nije prikazano izvršavanje upita "addDocumentTemplate" zbog problema s raščlanjivanjem JSON objekata u Grafičkom sustavu GraphiQL, ali taj upit se uspješno izvršava na klijentskoj strani.

Prvi čvor RootQuery-a koji je bitan za izradu druge aplikacije naziva se "fetchCourseProfessors". Prilikom pozivanja ovog čvora šalje se token i ID. ID se šalje samo kada se dohvaća informacija o pojedinom kolegiju nekog profesora, dok se bez tog ID-a dohvaćaju svi kolegiji.

Programski kod 4.13 – polje fetchCourseProfessors

```
fetchCourseProfessors: {
      type: GraphQLJSON,
      args: {
        token: { type: GraphQLNonNull(GraphQLString) },
        id: { type: GraphQLInt, defaultValue: null },
      },
      resolve(parent, args) {
        const config = {
          headers: { Authorization: `Bearer ${args.token}` },
          params: { id: args.id },
        };
        const id = args.id;
        return axios
          .get('http://localhost:3000/api/courseProfessors', { ...config, data: {
id } })
          .then((response) => {
            return response.data;
          })
          .catch((error) => {
            console.log(error);
            throw new Error('Failed to fetch courses: ' + error.message);
          });
      },
    },
```

Upiti za izvršavanje su prikazani na slikama 4.10 i 4.11.



Slika 4.10 - dohvaćanje svih profesora i njihovih predmeta

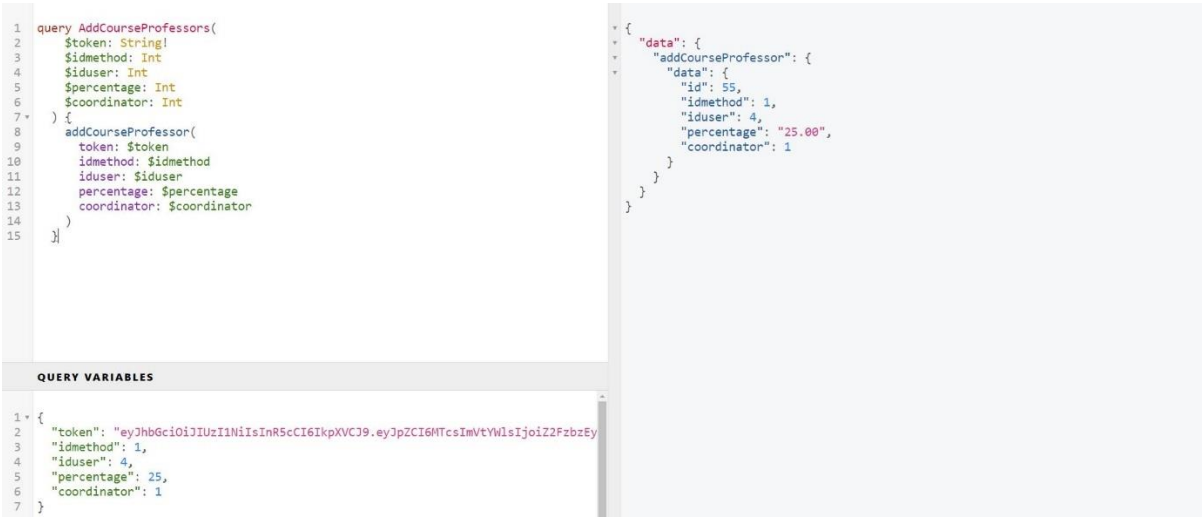
Slika 4.11 - dohvaćanje jednog profesora i njegovog predmeta

Sljedeći upit je za drugu aplikaciju *addCourseProfessors* koji dodijeljuje kolegije profesorima.

Programski kod 4.14 – polje addCourseProfessors

```
addCourseProfessor: {
      type: GraphQLJSON,
      args: {
        token: { type: GraphQLNonNull(GraphQLString) },
        idmethod: { type: GraphQLInt },
        iduser: { type: GraphQLInt },
        percentage: { type: GraphQLInt },
        coordinator: { type: GraphQLInt },
      },
      resolve(parent, args) {
        const config = {
          headers: { Authorization: `Bearer ${args.token}` },
        };
        const requestBody = {
          idmethod: args.idmethod,
          iduser: args.iduser,
          percentage: args.percentage,
          coordinator: args.coordinator,
        };
        return axios
          .post('http://localhost:3000/api/courseProfessors', requestBody, config)
          .then((response) => ({ data: response.data }))
          .catch((error) => {
            console.log(error);
            throw new Error('Failed to add course: ' + error.message);
          });
      }
    },
```

Izgled izvršenog upita je prikazan na slici 4.12.



Slika 4.12 - unos profesora i njegovog predmeta

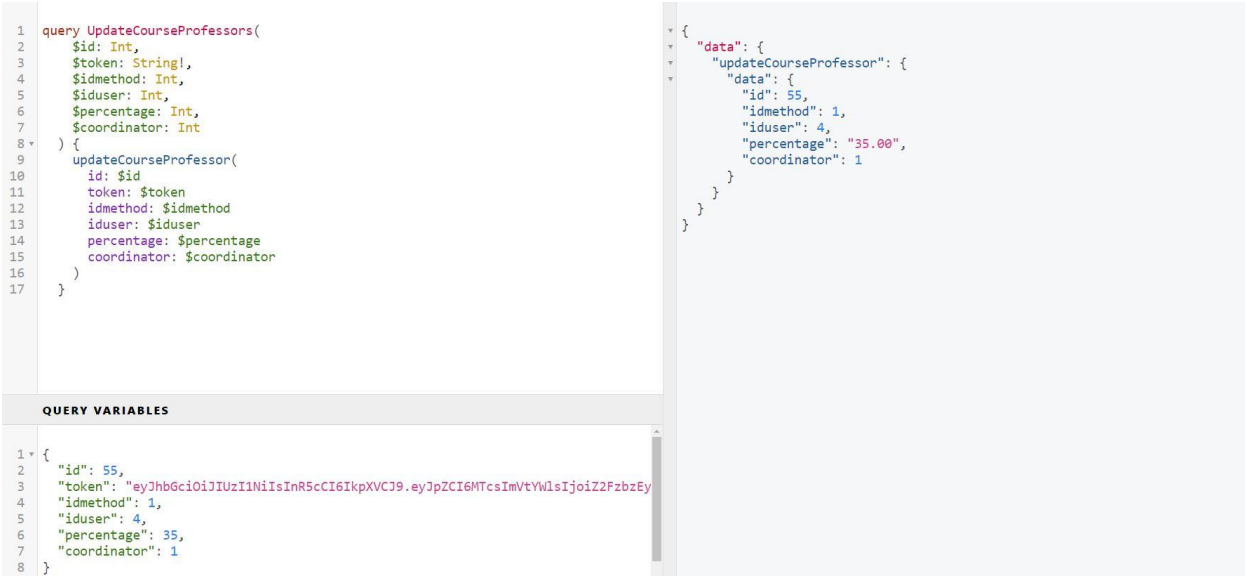
Sljedeći upit je updateCourseProfessors koji ažurira postojeći kolegij nekog profesora.

Programski kod 4.15 – polje updateCourseProfessors

```
updateCourseProfessor: {
      type: GraphQLJSON,
      args: {
        token: { type: GraphQLNonNull(GraphQLString) },
        id: { type: GraphQLInt },
        idmethod: { type: GraphQLInt },
        iduser: { type: GraphQLInt },
        percentage: { type: GraphQLInt },
        coordinator: { type: GraphQLInt },
      },
      resolve(parent, args) {
        const config = {
          headers: { Authorization: `Bearer ${args.token}` },
        };
        const requestBody = {
          id: args.id,
          idmethod: args.idmethod,
          iduser: args.iduser,
          percentage: args.percentage,
          coordinator: args.coordinator,
        };
        return axios
          .put('http://localhost:3000/api/courseProfessors/' , requestBody,
config)
          .then((response) => ({ data: response.data }))
          .catch((error) => {
```

```
            console.log(error);
            throw new Error('Failed to update course: ' + error.message);
          });
      }
    },
```

Izvršavanje upita ažuriranja je prikazan na slici 4.13.



Slika 4.13 - ažuriranje profesora i predmeta

Oba su u *RootQuery* čvoru zbog istih problema koje je *addUser* imao. Posljednja dva su *fetchMethods* i *fetchProfessors*.

Programski kod 4.16 – polja fetchMethods i fetchProfessors

```
fetchMethods: {
      type: GraphQLJSON,
      args: {
        token: { type: GraphQLNonNull(GraphQLString) },
      },
      resolve(parent, args) {
        const config = {
          headers: { Authorization: `Bearer ${args.token}` },
        };
        return axios
          .get('http://localhost:3000/api/courseProfessors/method', config)
          .then((response) => {
            return response.data;
          })
```

```
        .catch((error) => {
          console.log(error);
          throw new Error('Failed to fetch methods: ' + error.message);
        });},},

  fetchProfessors: {
    type: GraphQLJSON,
    args: {
      token: { type: GraphQLNonNull(GraphQLString) },},
    resolve(parent, args) {
      const config = {
        headers: { Authorization: `Bearer ${args.token}` },
      };
      return axios
        .get('http://localhost:3000/api/courseProfessors/user', config)
        .then((response) => {
          return response.data;
        })
        .catch((error) => {
          console.log(error);
          throw new Error('Failed to fetch professors: ' + error.message);
        });}},
```

Prvi upit dohvaća sve metode odnosno predmet i vrstu nastave (predavanje, laboratorijske vježbe,itd.), drugi dohvaća sve profesore. Ta su dva upita napravljena kako bi unos i ažuriranje podataka na klijentskom dijelu bio jednostavniji. Ti se upiti izvršuju prema slikama 4.14 i 4.15.

Slika 4.14 - dohvaćanje predmeta i njihovih metoda

Slika 4.15 - dohvaćanje profesora

Na kraju je ostala mutacija, u njoj se nalazi upit *addDocumentResponse*. Pomoću tog upita unose se rezultati dobiveni popunjavanjem dokumenata. Također nema prikaz izvršavanja zbog problema s *GraphiQL* grafičkim sustavom i raščlanjivanjem *JSON* objekata.

Programski kod 4.17 – polje addDocumentResponse

```
const Mutation = new GraphQLObjectType({
  name: 'Mutation',
  fields: {
    addDocumentResponse: {
      type: DocumentType,
      args: {
        token: { type: GraphQLNonNull(GraphQLString) },
        name: { type: GraphQLNonNull(GraphQLString) },
        response: { type: GraphQLNonNull(GraphQLJSON) },
      },
      resolve(parent, args) {
        const config = {
          headers: { Authorization: `Bearer ${args.token}` },
          params: { name: args.name ,
                    response: args.response},
        };
        const name = args.name;
        const response = args.response
        return axios
          .post('http://localhost:3000/api/schemas/response', { name, response },
config)
          .then((response) => ({ data: response.data }))

.catch((error) => {console.log(error);
```

```
throw new Error('Failed to add document: ' + error.message);});}},}});
```

Kako bi se ti upiti mogli koristiti mora se stvoriti shema. Ona se stvara na ovaj način:

Programski kod 4.18 – izvoz sheme

```
module.exports = new graphql.GraphQLSchema({
  query: RootQuery,
  mutation: Mutation
});
```

U njoj se nalaze dva dijela, *query* i *mutation*. Svaki reprezentira svoje upite. Za konačno pokretanje i stopostotno stvaranje GraphQL servera mora se stvoriti GraphQL krajna točka (engl. *endpoint*) koji završava na */graphql*. U njega je potrebno podstaviti shemu i parametar graphiql staviti pod true kako bi se mogli testirati upiti.

Programski kod 4.19 – GraphQL endpoint

```
// GraphQL endpoint
app.use('/graphql', expressGraphQL({
    schema: schema,
    graphiql: true
}));
```

U završnom radu korišten je također i *cors* biblioteka kako bi pozivanje ruta, izvršavanje upita s klijentskog dijela bila jednostavnija. U sljedećem poglavlju objasnit će se klijentski dio završnog rada, kako su napravljene stranice, spremaju podatci i ostalo.

5. IZRADA KLIJENTSKOG DIJELA

U ovome poglavlju objasnit će se klijentski dio završnog rada, izrada React stranica, usmjeravanje među njima, spremanje bitnih podataka i ostalo.

5.1 GraphQL klijentski dio

Za početak je potrebno instalirati Create React App biblioteku, koja se koristi kako bi se brzo i jednostavno stvorila React aplikacija. Stvaranje projekta izvršava se naredbama "npx create-react-app my-app" i "cd my-app". Nakon stvaranja projekta, potrebno je instalirati sve potrebne pakete za rad s GraphQL klijentskim dijelom, među kojima su: @apollo/client, @apollo/react-hooks, datatables, graphql, jquery, react, react-dom, react-redux, react-router, react-router-dom, redux, survey-react i survey-react-ui.

Programski kod 5.1 – NPM paketi za GraphQL klijentski dio

```
{
  "name": "vubapp-client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@apollo/client": "^3.7.16",
    "@apollo/react-hooks": "^4.0.0",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^11.2.6",
    "@testing-library/user-event": "^13.5.0",
    "datatables": "^1.10.18",
    "datatables.net": "^1.13.5",
    "graphql": "^15.8.0",
    "jquery": "^3.7.0",
    "react": "^16.8.0",
    "react-dom": "^16.8.0",
    "react-redux": "^8.1.1",
    "react-router": "^6.12.1",
    "react-router-dom": "^6.13.0",
    "react-scripts": "5.0.1",
    "redux": "^4.2.1",
    "redux-persist": "^6.0.0",
    "survey-react": "^1.9.97",
    "survey-react-ui": "^1.9.97","web-vitals": "^2.1.4"},
```

Za izradu UI/UX komponente korišten je administratorski predložak (engl. *template*) *nobleui2-main*. Iskorišten je *index.html* iz tog templatea te sve *CSS* datoteke. Izgled jednog dijela te *HTML* datoteke:

Programski kod 5.2 – HTML dio

```
    <!-- Layout styles -->
    <link rel="stylesheet" href="%PUBLIC_URL%/assets/css/styles/style.css" />
    <!-- End layout styles -->

    <link rel="shortcut icon" href="%PUBLIC_URL%/assets/images/favicon.png" />
  </head>
  <body class="sidebar-dark">
    <div class="main-wrapper">
      <!-- partial:partials/_sidebar.html -->
      <nav class="sidebar">
        …
      </nav>
      <!-- partial -->

      <div class="page-wrapper">
        <!-- partial -->

        <div id="root">
          <!-- The root div will be replaced by your React components -->
        </div>
        <!-- partial -->
    </div>
    </div>
```

U njemu se nalazi kod za bočnu traku i *div* element u koji se unose React komponente za izradu React stranica. Glavna datoteka u projektu je *index.js*. U njoj se nalaze najviše uvoza (engl. *import*) svih stranica, sustava za rad s Apollo Clientom, itd. Sastoji se od tri dijela svi potrebni uvozi, globalne varijable za rad te Root komponenta. Navedeni su u sljedećem programskom kodu:

Programski kod 5.3 – uvozi u glavnoj datoteci

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Login from './Login';
import ReactDOM from 'react-dom';
import { ApolloClient, InMemoryCache, ApolloProvider } from '@apollo/client';
import Register from './Register';
import Home from './Home';
import { Provider } from 'react-redux';
import { PersistGate } from 'redux-persist/integration/react';
import { store, persistor } from './store';
import ResponseCreate from './ResponseCreate';
import ResponseList from './ResponseList';
import DocumentTemplate from './DocumentTemplate';
import './css/styles/style.css';
…
```

U globalnim varijablama se spremaju podatci dobiveni u projektu kako bi se mogli koristiti s prelaska jedne na drugu stranicu.

Programski kod 5.4 - globalne varijable

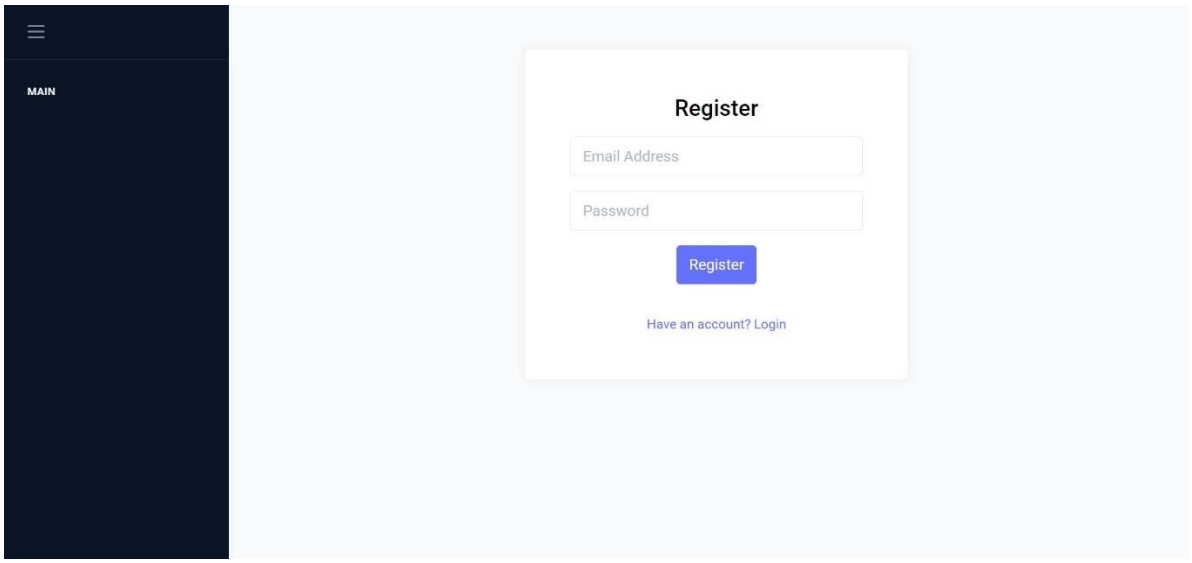
```
global.surveyData='';
let storedRole = JSON.parse(localStorage.getItem('role'));
global.role = storedRole && storedRole.get ? storedRole :
{get:0,post:0,put:0,delete:0};
global.name={};
let storedTableData = JSON.parse(localStorage.getItem('tableData'));



global.tableData = storedTableData && storedTableData.name ? storedTableData : {
  object:{
    object: [
      { name: 'Document 1', body: 'Answer 1' },
      { name: 'Document 2', body: 'Answer 2' },
    ]
  }
};
let storedData = JSON.parse(localStorage.getItem('applicationData'));
global.applicationData = storedData && storedData.name ? storedData : {
  id: 1,
  name: 'Example Application',
};
…
```

U projektu se koristi *selector* mogućnost za spremanje korisnikovog tokena. Ta mogućnost se ostvaruje korištenjem *react-redux* i *redux* biblioteka koje se koriste za upravljanje stanja u React aplikaciji. U svrhu završnog rada iskorištena je mogućnost *store* koja sprema to stanje odnosno token kako bi se nepromijeljen token koristio kroz sve React stranice jer token podatak je podatak koji se ne mijenja često u ovome završnom radu.

Programski kod 5.5 – Reducer i Store datoteke za spremanje i dohvaćanje stanja tokena

```
const initialState = null;

const tokenReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'SET_TOKEN':
      return action.payload;
    default:
      return state;
  }
};

export default tokenReducer;

import { createStore, combineReducers } from 'redux';
import { persistStore, persistReducer } from 'redux-persist';
import storage from 'redux-persist/lib/storage';
import tokenReducer from './reducer';

const persistConfig = {
  key: 'root',
  storage,
  whitelist: ['token']
};

const rootReducer = combineReducers({
  token: tokenReducer
});

const persistedReducer = persistReducer(persistConfig, rootReducer);

export const store = createStore(persistedReducer);

export const persistor = persistStore(store);
```

Za početak potrebno je pokrenuti Node JS server te React developing server. Prva stranica koja se prikaže je stranica za registraciju korisnika.



Slika 5.1 - izgled stranice za registraciju

U njoj se nalaze dva polja za unos e-mail adrese i lozinke i gumb za registraciju. Također se nalazi gumb u slučaju da postoji korisnik koji navigira na stranicu za prijavu. Za izradu registracijske stranice potrebni su ovi uvozi prikazani u programskom kodu.

Programski kod 5.6 – uvozi za registracijsku stranicu

```
import React, { useState } from 'react';
import { gql, useLazyQuery } from '@apollo/client';
import { Link, useNavigate } from 'react-router-dom';
import { useDispatch } from 'react-redux';
```

Koristi se *useLazyQuery* koji omogućuje pozivanje GraphQL upita nakon neke određene akcije. U cijeloj stranici nalaze se četiri anonimne funkcije, jedna za izvršavanje upita i dvije za postavljanje e-maila i lozinke te jedna koja rukuje s gumbom za registraciju u njoj se direktno poziva funkcija u kojoj se izvršuje upit.

Programski kod 5.7 – poziv upita za registraciju i anonimne funkcije

```
const [register, { loading, error }] = useLazyQuery(REGISTER_QUERY, {
    onCompleted: (response) => {
      if (response && response.addUser) {
        global.role={get:response.addUser.data.role.get,post:response.addUser.data
.role.post,put:response.addUser.data.role.put,delete:response.addUser.data.role.de
lete};
        dispatch(setToken(response.addUser.data.token));
```

```
        setData(response.addUser);
        navigate('/home');
      }
    },});
  const handleSubmit = (event) => {
    event.preventDefault();
    register({ variables: { email, password } });
  };

  const handleEmailChange = (event) => {
    setEmail(event.target.value);
  };

  const handlePasswordChange = (event) => {
    setPassword(event.target.value);
  };
```

Izgled JSX/HTML dijela stranice je prikazan u sljedećem programskom kodu.

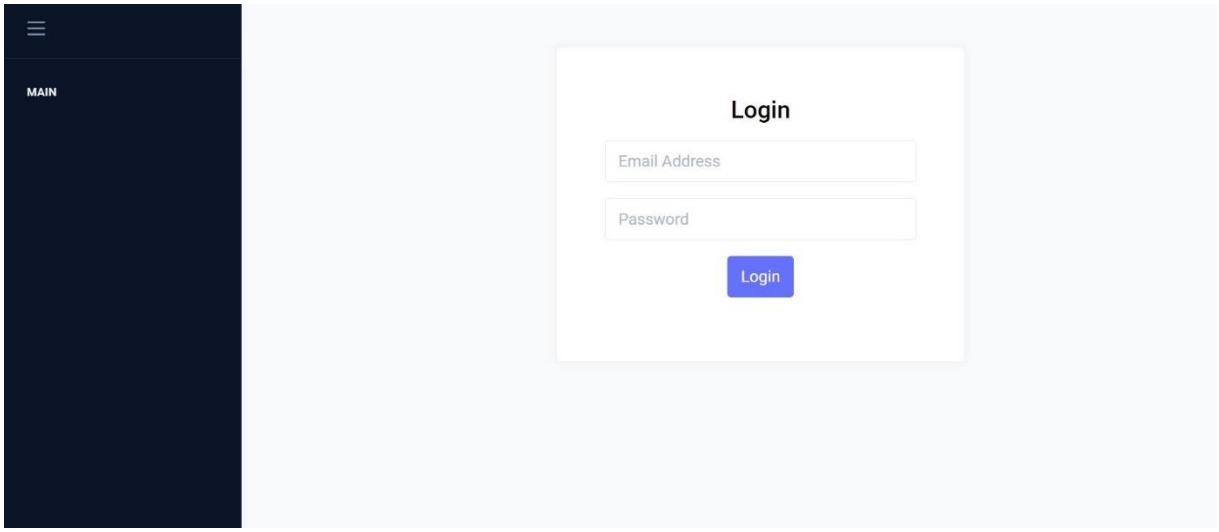
Programski kod 5.8 – JSX/HTML registracijske stranice

```
return (
    <div className="container">
      <div className="row justify-content-center">
        <div className="col-xl-5 col-lg-6 col-md-7">
          <div className="card my-5">
            <div className="card-body p-5 text-center">
              <div className="h3 font-weight-light mb-3">Register</div>
              <form onSubmit={handleSubmit}>
                <div className="form-group mb-3">
                  <input className="form-control form-control-lg" type="email"
name="email" value={email} onChange={handleEmailChange} placeholder="Email
Address" />
                </div>
                <div className="form-group mb-3">
                  <input className="form-control form-control-lg" type="password"
name="password" value={password} onChange={handlePasswordChange}
placeholder="Password" />
                </div>
                <button className="btn btn-lg btn-block btn-primary mb-3"
type="submit" disabled={loading}>Register</button>
                {loading && <p>Loading...</p>}
                {error && <p>Error: {error.message}</p>}
                {data.length > 0 && (
                  <ul>
                    {data.map((item) => (
```

```
                    <li key={item.id}>{item.body}</li>
                  ))}
                </ul>
              )}
            </form>
            <div className="text-center mt-3">
              <Link className="text-gray-500" to="/login">Have an account?
Login</Link>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
```

U anonimnoj funkciji za registraciju koristi se unos dodijeljenih uloga novonastalog korisnika u globalnu varijablu kako bi se stvorila autorizacija. Također se postavlja token kao stanje, a nakon uspješne registracije preusmjerava se na stranicu /home.

U slučaju postojanja korisnika, koristi se stranica /login.



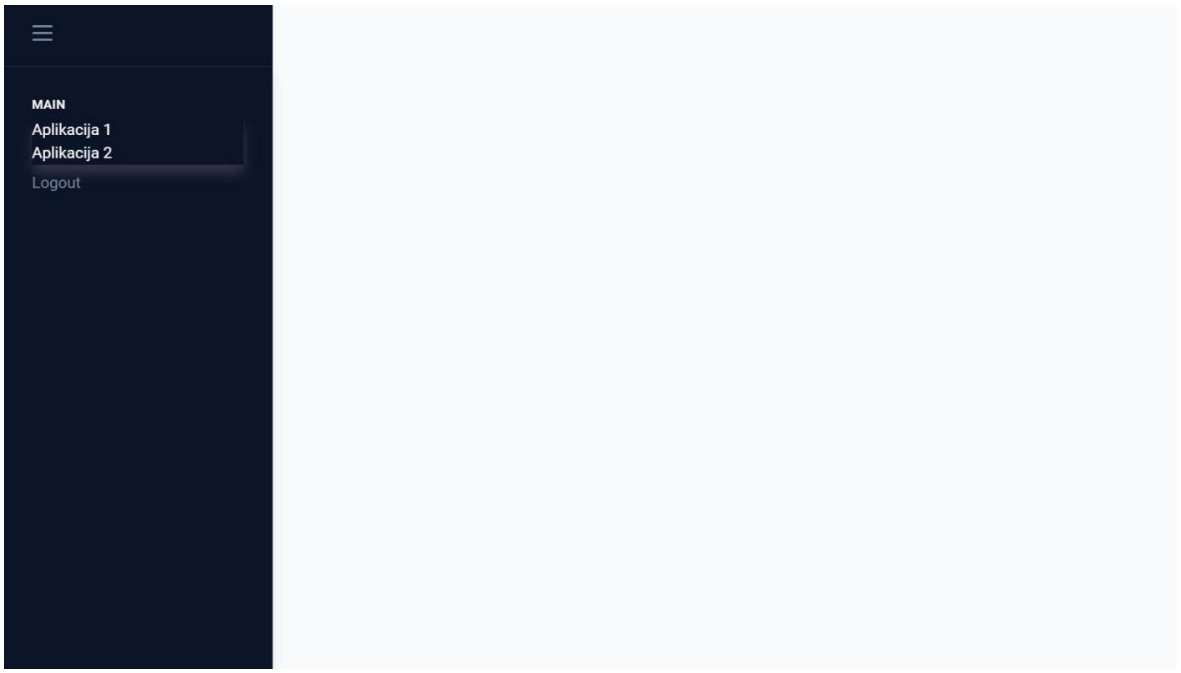
Slika 5.2 - prijava u aplikaciju

U njoj se nalaze identični uvozi kao i na stranici za registraciju. Obje stranice funkcioniraju na isti način, jedina razlika je u tome što postojeći korisnik ima već dobivene uloge i aplikacije za rad. U njoj se dodijeljuju podatci o aplikacijama i ulogama kako bi cijela aplikacija funkcionirala.

Programski kod 5.9 – dodijeljivanje podataka o ulogama i aplikacijama

```
const [login, { loading, error }] = useLazyQuery(LOGIN_QUERY, {
    onCompleted: (response) => {
      if (response && response.login) {
        console.log(response.login.data);
        setData(response.login.data);

        if (response.login.data.role && response.login.data.role.length > 0) {
          const roles = response.login.data.role;
          const applicationData = roles.map(roles => ({
            idapplication: roles.idapplication,
            name: roles.name,
          }));

           const roleData = roles.map(roles => ({
            get: roles.get,
            post: roles.post,
            put: roles.put,
            delete: roles.del,
          }));
            global.applicationData = applicationData.map(applicationData => ({
              id: applicationData.idapplication,
              name: applicationData.name,
            }));
          console.log(global.applicationData);
          global.role = roleData.map(roleData => ({
            get: roleData.get,
            post: roleData.post,
            put: roleData.put,
            delete: roleData.delete,
          }));

          console.log(global.role);
          localStorage.setItem('role', JSON.stringify(global.role));
          localStorage.setItem('applicationData',
JSON.stringify(global.applicationData));
        }

        dispatch(setToken(response.login.data.token));
        navigate('/home');
      }
    },
  });
```

Koristi se trostruko mapiranje jer se objekt *role* šalje u *JSON* formatu koji sadrži podatke o više aplikacija te korisničkim ulogama unutar tih aplikacija. Poslije toga se koristi lokalno spremanje pomoću *localStorage* funkcionalnosti, postavljanje tokena te navigiranje na stranicu */home*.



Slika 5.3 - početna stranica aplikacije

Sljedeća stranica je stranica s nazivom */home* . Na bočnoj traci se nalaze dva gumba budući da su samo dvije aplikacije, ali da ih ima više, bilo bi više gumbova. U globalnoj varijabli *applicationData* nalazi se podatak o aplikacijama.

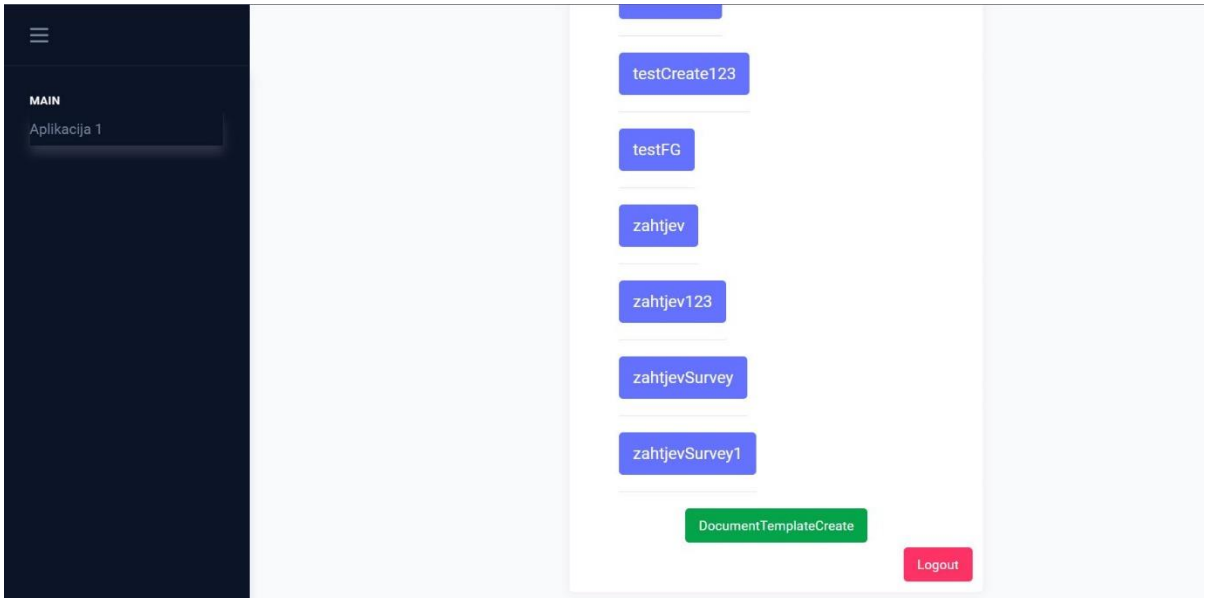
Programski kod 5.10 – punjenje div elementa aplikacijama

```
  state = {
    pageList: initialPageList,
    isCollapsed: false,
  };

  componentDidMount() {
    // Retrieve data from local storage
    const applicationData = JSON.parse(localStorage.getItem('applicationData')) ||
{};
    const pageList = Object.keys(applicationData).map((key) => ({
      name: applicationData[key].name,
      link: applicationData[key].name === 'Aplikacija 2' ? '/course-professors' :
'/document-template',
    }));this.setState({ pageList });}
```

```
  componentDidUpdate() {
    // Save the updated data in local storage when the component is updated
    localStorage.setItem('applicationData',
JSON.stringify(global.applicationData)); }
```

Iz varijable se dobije ime aplikacije te se podstavi veza koja vodi na odgovarajuće aplikacije. Prva aplikacija naziva „Aplikacija 1“ navigira na stranicu */document-template*, a druga s nazivom „Aplikacija 2“ na stranicu */course-professors*. Također, na bočnoj traci se nalazi gumb za odjavljivanje iz cijele aplikacije koji kada se pritisne vodi na stranicu */login*. To su bile tri glavne stranice odnosno tri zasebne stranice nevezanu za pojedinu aplikaciju.

Prva stranica koja nadolazi pritiskom gumba Aplikacija 1 je „/document-template“.



Slika 5.4 - dohvaćanje dokumenata

Na bočnoj traci nalazi se sivkasti tekst u kojem piše na kojoj aplikaciji se nalazi stranica. Na svim stranicama koje rade s dokumentima nalazit će se tekst „Aplikacija 1“.

Programski kod 5.11 – primjer upita za dohvaćanje dokumenata

```
const FETCH_DOCUMENT_TEMPLATE_QUERY = gql`
  query FetchDocumentTemplate($token: String!, $name: String!) {
    fetchDocumentTemplate(token: $token, name: $name)
  }
`;



const FETCH_DOCUMENT_TEMPLATES_QUERY = gql`
  query FetchDocumentTemplates($token: String!) {
    fetchDocumentTemplate(token: $token)
  }
```

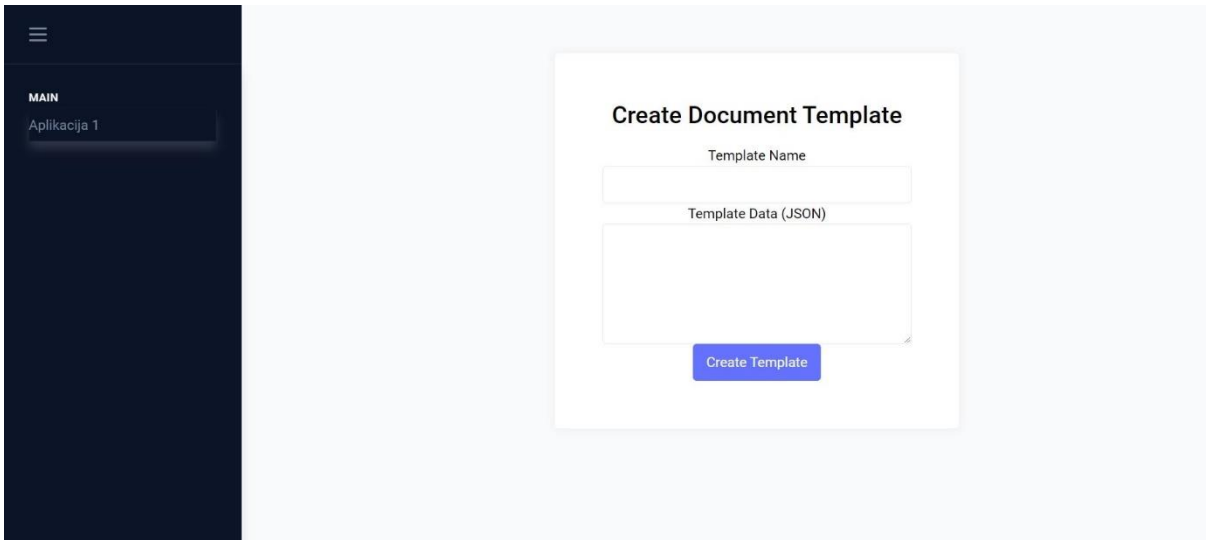
```
`;
```

Na stranici se poziva isti upit dva puta *fetchDocumentTemplate* , prvi put sa tokenom, a drugi put s tokenom i *name* varijablom.

Programski kod 5.12 – izvršavanje tih upita

```
const { loading: queryLoading, error, data: queryData } =
useQuery(FETCH_DOCUMENT_TEMPLATES_QUERY, {
    variables: { token },
  });

  useEffect(() => {
    if (queryData) {
      setDocumentData(queryData.fetchDocumentTemplate || []);
    }
  }, [queryData]);

  const [fetchDocumentTemplate, { data }] =
useLazyQuery(FETCH_DOCUMENT_TEMPLATE_QUERY, {
    onCompleted: (response) => {
      if (response && response.fetchDocumentTemplate) {
        global.surveyData = response.fetchDocumentTemplate;
        setLoading(false);
        navigate('/response-create');
      }
    },
  });
```

Prilikom prvog poziva, varijabla "queryData" se popunjava kako bi se omogućilo prikazivanje imena tih dokumenata na samoj stranici. Kada se pritisne jedan od tih gumbova, uzima se ime tog gumba, odnosno ime datoteke, te se izvršava drugi upit s tokenom i imenom, rezultirajući dohvaćanjem JSON objekata potrebnih za izgradnju SurveyJS forme kako bi se taj dokument mogao popuniti. Taj JSON podatak se sprema u globalnu varijablu "surveyData". Također postoji još jedan gumb koji vodi na stranicu za kreiranje novih dokumenata s nazivom "/document-template-create".Na toj stranici nalaze se dva polja za unos, jedan za ime dokumenta drugi za unos *SurveyJS* forme odnosno *JSON*.



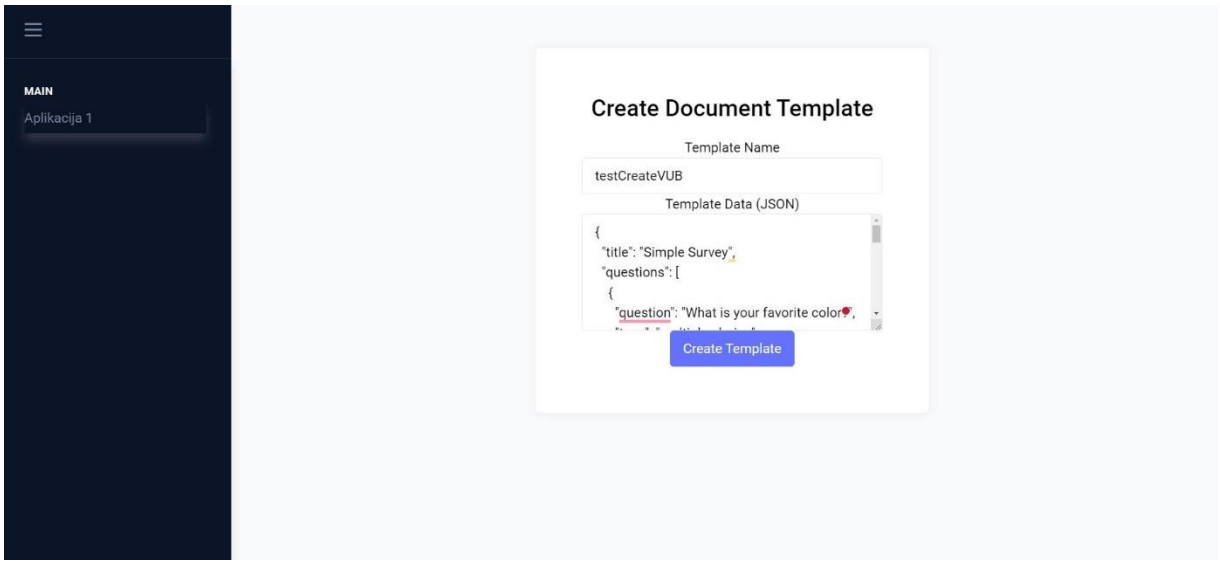
Slika 5.5 – izgled stranice za kreiranja dokumenta

Kada pritisne se gumb za unos vraća novog dokumenta vraća se na prethodnu stranicu na kojoj je prikazan novonastali dokument. Programski kod za unos novog dokumenta, kada je uspješan unos vraća se na prethodnu stranicu */document-template*.

Programski kod 5.13 – izvršavanje upita za unos dokumenta

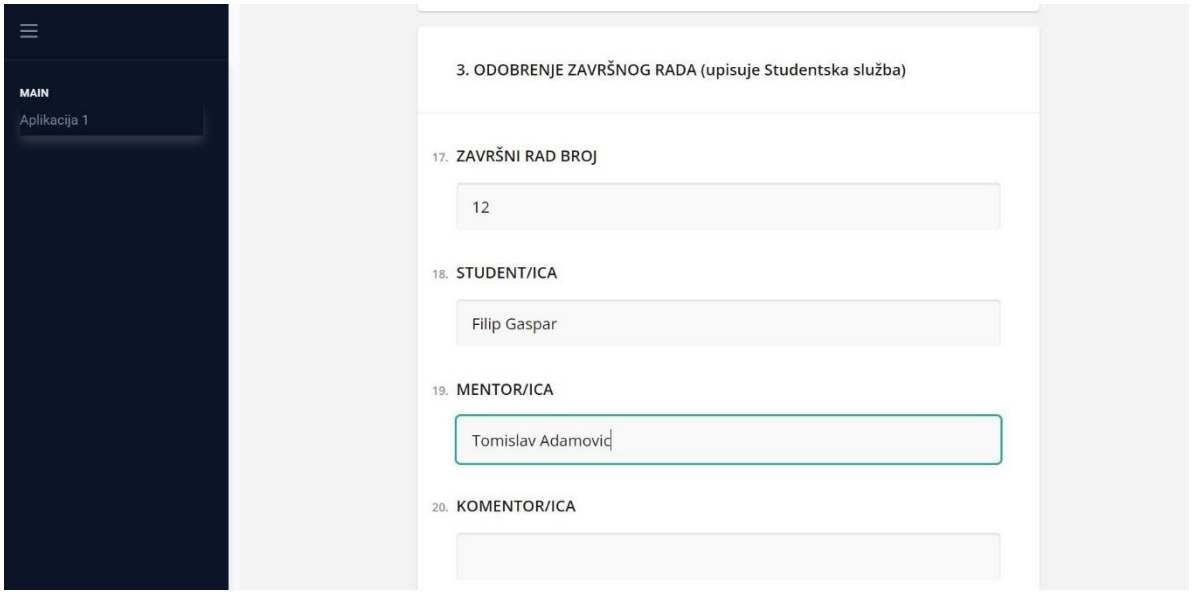
```
const handleCreateTemplate = async () => {
    try {
      const response = await addTemplate({
        variables: {
          token,
          name,
          body: data,
        },
      });

      if (response && response.data.addDocumentTemplate) {
        console.log('Document template added:',
response.data.addDocumentTemplate);
        navigate('/document-template');
      }
    } catch (error) {
      console.error('Error adding document template:', error);
    }
  };
```

Na slici 5.6 je prikazana izrada forme za jedan dokument. U prvom tekstnom polju se napiše ime dokumenta, a u drugom *JSON* podataka dokumenta.



Slika 5.6 - izgled unesene forme za kreiranje dokumenta

Kada se pritisne gumb s dokumentom, navigira se na stranicu */response-create*.



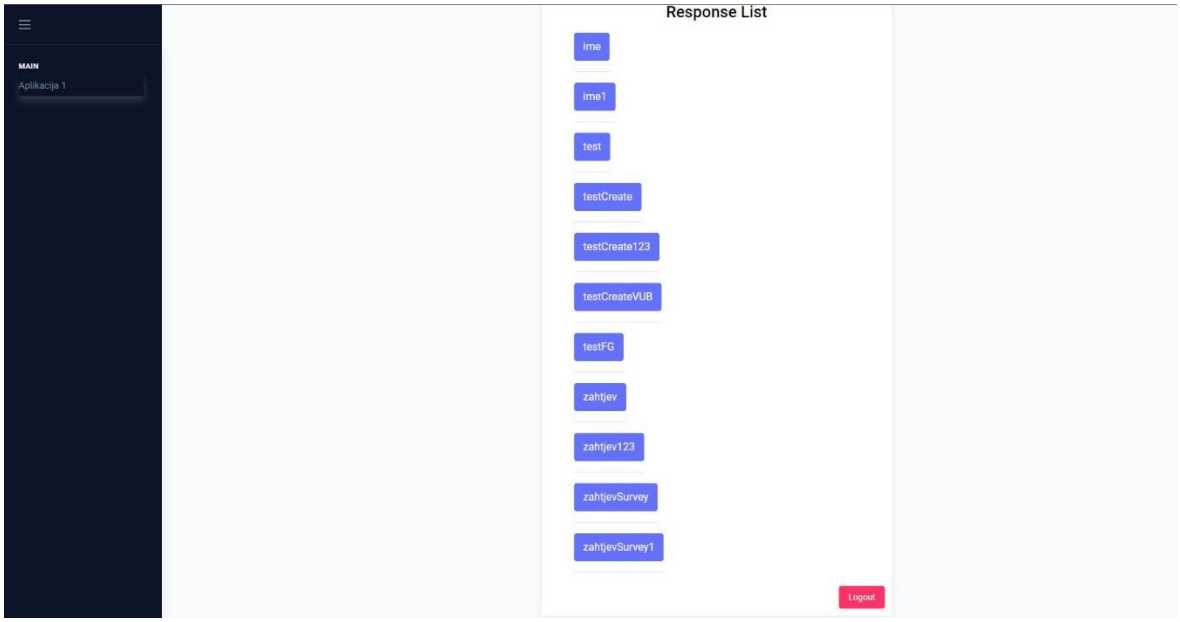
Slika 5.7 - unos podataka u dokument

Stranica */response-create* je svaki put posebnog izgleda ovisno o *JSON* objektu same *SurveyJS* forme. Na njoj se kreira model za stvaranje *SurveyJS* forme koji stvara jednostavnu anketu odnosno dokument za unos podataka. Kada se popuni ta anketa odnosno dokument uzme se *JSON* podatak dobiven popunjavanjem dokumenta te se izvrši *addDocumentResponse* upit za koji je potreban token, ime i sami taj *JSON* podatak.

Programski kod 5.14 – primjer unosa rješenja dokumenta

```
const [addTemplate, { loading, error }] = useLazyQuery(addDocumentTemplateQuery);

  const handleCreateTemplate = async () => {
    try {
      const response = await addTemplate({
        variables: {
          token,
          name,
          body: data,
        },
      });

      if (response && response.data.addDocumentTemplate) {
        console.log('Document template added:',
response.data.addDocumentTemplate);
        navigate('/document-template');
      }
    } catch (error) {
      console.error('Error adding document template:', error);
    }
  };
```

Nakon uspješnog unosa, navigira se na stranicu */response-list*.



Slika 5.8 - lista dokumenata za vidjeti rješenja

Na toj stranici nalazi se ista forma kao i na */document-template* stranici. Razlika između tih stranica je ta što pritiskom gumba za jedan od dokumenata dobivaju se rješenja tj. skup podataka koje je taj korisnik unio za taj dokument.

Taj skup podataka se dobiva izvršavanjem upita *fetchDocumentResponse*.

Programski kod 5.15 – upiti za dohvaćanje rješenje dokumenta

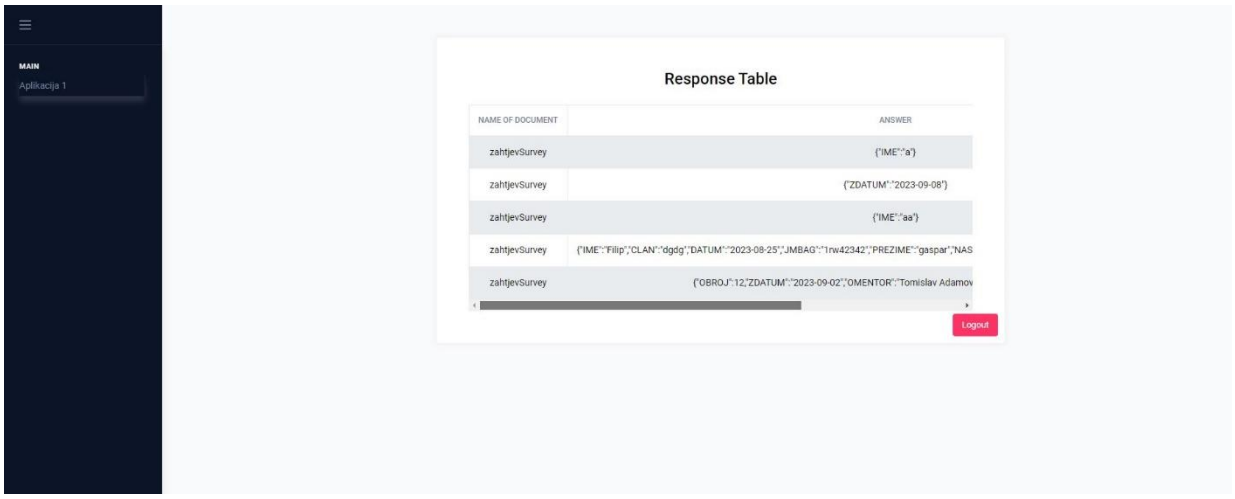
```
const FETCH_DOCUMENT_RESPONSE_QUERY = gql`
  query FetchDocumentResponse($token: String!, $name: String!) {
    fetchDocumentResponse(token: $token, name: $name)
  }
`;

const FETCH_DOCUMENT_TEMPLATES_QUERY = gql`
  query FetchDocumentTemplates($token: String!) {
    fetchDocumentTemplate(token: $token)
  }
`;
```

Programski primjer izvršavanja upita za dohvaćanje rezultata dokumenta.

Programski kod 5.16 – primjer dohvaćanja rezultata dokumenta

```
const [fetchDocumentResponse, { data }] =
useLazyQuery(FETCH_DOCUMENT_RESPONSE_QUERY, {
    onCompleted: (response) => {
      if (response && response.fetchDocumentResponse) {
        global.tableData = {object:response.fetchDocumentResponse}
        localStorage.setItem('tableData', JSON.stringify(global.tableData));
        setLoading(false);
        navigate('/response-table');
      }
    },
  });
```

Nakon uspješnog pritiska gumba navigira se na posljednju stranicu aplikacija za rad s dokumentima */response-table*. Na toj se stranici nalazi *datatables* tablica dokumenata s pripadajućim podacima.



Slika 5.9 - tablica rješenja dokumenata

Podatci o tablici dobiveni su pomoću globalne varijable *tableData* koja je bila prethodno popunjena na prijašnjoj stranici. U tablici se prikazuje ime dokumenta i *JSON* podatak.

Programski kod 5.17 – inicijalizacija datatables tablice

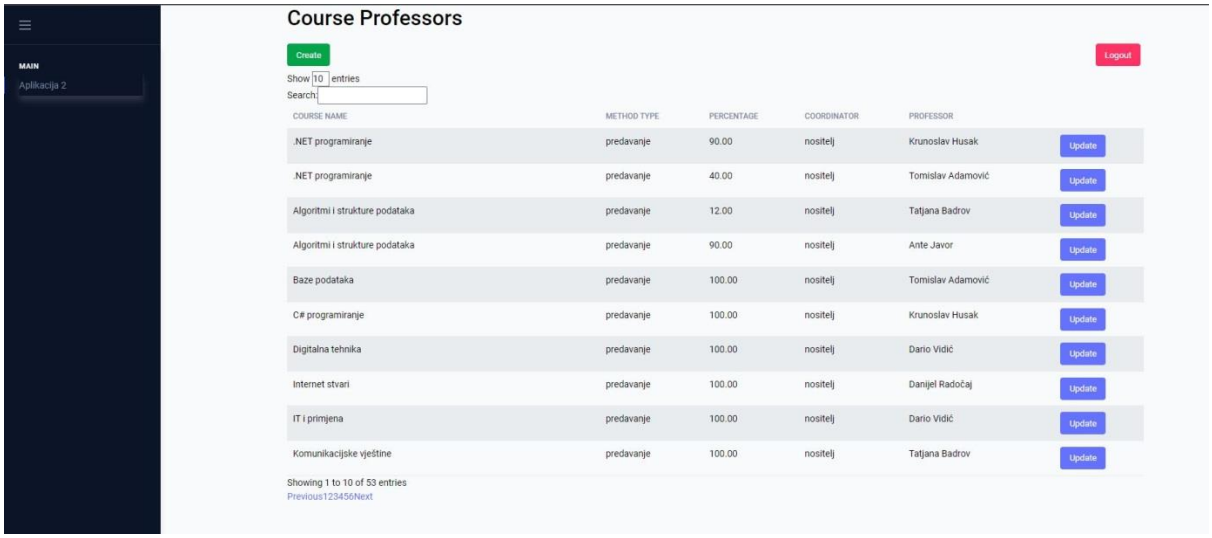
```
useEffect(() => {
    const storedTableData = JSON.parse(localStorage.getItem('tableData'));
    const initialTableData = storedTableData && storedTableData.object
      ? storedTableData.object
      : [{ name: 'Document 1', body: 'Answer 1' }];
    setTableData(initialTableData);

    if (tableRef.current && !dataTableInitialized) {
      setDataTableInstance($(tableRef.current).DataTable());
      setDataTableInitialized(true);
    }
  }, [dataTableInitialized]);

  useEffect(() => {
    if (dataTableInstance) {
      dataTableInstance.clear();
      dataTableInstance.rows.add(tableData || []);
      dataTableInstance.draw();
    }
  }, [dataTableInstance, tableData]);
```

Izgled JSX/HTML dijela tablice prema programskom kodu.

Programski kod 5.20 – JSX/HTML datatables tablica

```
<div className="table-responsive">
                {Array.isArray(tableData) && tableData.length > 0 ? (
                  <table
                    ref={tableRef}
                    className="table table-striped table-bordered"
                  >
                    <thead>
                      <tr>
                        <th>Name of document</th>
                        <th>Answer</th>
                      </tr>
                    </thead>
                    <tbody>
                      {tableData.map((row, index) => (
                        <tr key={index}>
                          <td>{row.name}</td>
                          <td>{JSON.stringify(row.body)}</td>
                        </tr>
                      ))}
                    </tbody>
                  </table>
                ) : (
                  <div>Loading...</div>
                )}
              </div>
```

Povratkom na */home* stranicu i pritiskom na drugu aplikaciju navigira se na stranicu */course-professors*. Na toj stranici se nalazi *datatables* tablica za prikaz kolegija svih profesora.



Slika 5.10 - izgled stranice za dohvaćanje profesore i kolegije

Na stranici se nalaze tri gumba, jedan za odjavu, jedan za kreiranje sloga u tablicu *course_professors* i na svakom retku gumb za ažuriranje tog specifičnog zapisa.

Programski kod 5.19 – upit za dohvaćanje profesorovih kolegija

```
const FETCH_COURSE_PROFESSORS_QUERY = gql`
  query FetchCourseProfessors($token: String!) {
    fetchCourseProfessors(token: $token)
  }
`;
const { loading, data } = useQuery(FETCH_COURSE_PROFESSORS_QUERY, {
    variables: { token },
  });

  useEffect(() => {
    if (data && tableRef.current) {
      if ($.fn.DataTable.isDataTable(tableRef.current)) {
        $(tableRef.current).DataTable().destroy();
      }
      $(tableRef.current).DataTable({
        data: data.fetchCourseProfessors || [],
        columns: [
          { data: 'course_name' },
          { data: 'method_type' },
          { data: 'percentage' },
          { data: 'coordinator' },
          { data: 'professor' },
          { data: 'id', visible: false },
```

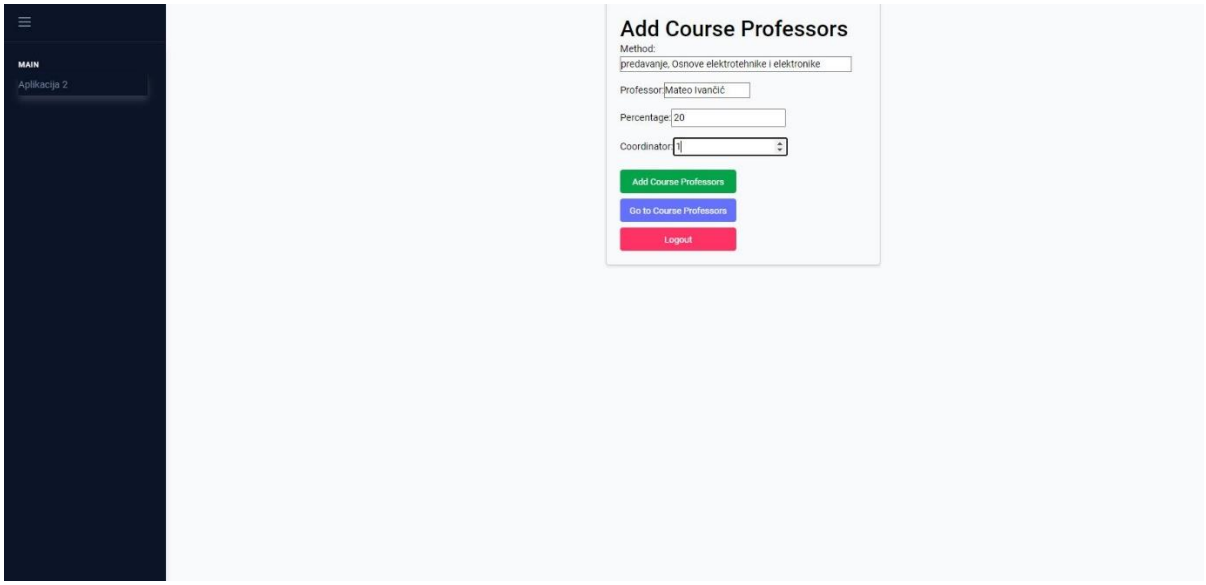
```
          {
            data: null,
            render: function (data, type, row) {
              return `<button class="btn btn-sm btn-primary update-button" data-
id="${row.id}">Update</button>`;
            },
          },
        ],
      });
```

Kreiranje gumba za ažuriranje na svakom retku prikazan je u sljedećem programskom kodu.

Programski kod 5.20 – kreiranje gumba za ažuriranje

```
$(tableRef.current).on('click', '.update-button', function () {
        const id = $(this).data('id');
        global.courseProfessorID = id;
        localStorage.setItem('courseProfessorID', global.courseProfessorID);
        setSelectedCourseProfessorID(id);
      });
```

U tablici su prikazani samo podatci o kolegiju, metodi nastave, koji profesor vodi predmet, njegov postotak i je li izvođač ili nositelj. Nije prikazan stupac s *ID-jevima*. Upit koji kreira tablicu je *fetchCourseProfessors*. Pritskom gumba za kreiranje novog sloga navigira se na stranicu */course-professors-create*.



Slika 5.11 - unos novog predmeta za profesora

Na toj stranaici se nalaze dva padajuća izbornika s podatcima o profesoru i metodi nastave i ostala polja za unos. Padajući izbornici pune se upitima koji se pozovu svakim osvježavanjem stranice. Ta dva upita su *fetchMethods* i *fetchProfessors*.

Programski kod 5.21 – upiti za kreiranje novog sloga tablice course_professors

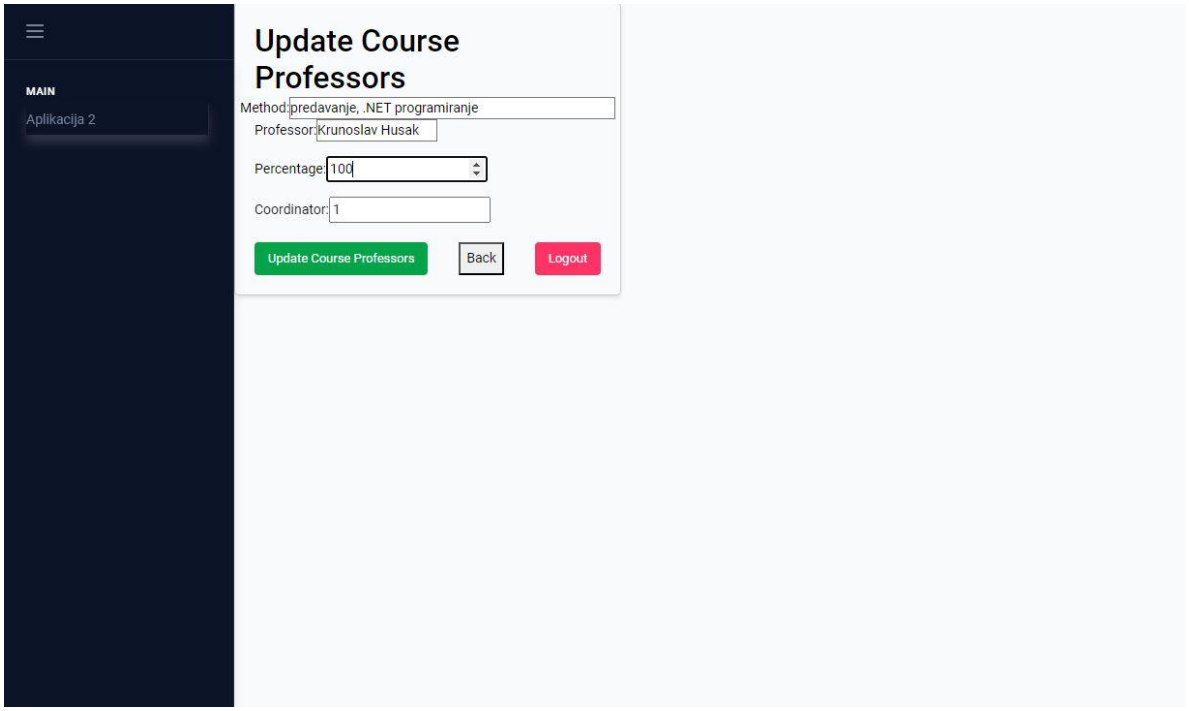
```
const FETCH_METHODS_QUERY = gql`
  query FetchMethods($token: String!) {
    fetchMethods(token: $token)
  }
`;

const FETCH_PROFESSORS_QUERY = gql`
  query FetchProfessors($token: String!) {
    fetchProfessors(token: $token)
  }
`;
const { data: methodsData, loading: methodsLoading } =
useQuery(FETCH_METHODS_QUERY, {
    variables: { token },
  });

  const { data: professorsData, loading: professorsLoading } =
useQuery(FETCH_PROFESSORS_QUERY, {
    variables: { token },
  });
```

Kada se popuni forma i pritisne gumb, dohvate se podatci iz svih polja i aktivira se upit *addCourseProfessors*.

Programski kod 5.22 – izvršavanje unosa

```
const handleAddCourseProfessors = () => {
    addCourseProfessors({
      variables: {
        token,
        idmethod: parseInt(idmethod),
        iduser: parseInt(iduser),
        percentage: parseInt(percentage),
        coordinator: parseInt(coordinator),
      },
    });
    navigate('/course-professors');
  };
```

Nakon uspješnog kreiranja, vraća se nazad na stranicu */course-professors*. Za ažuriranje jednog reda podatka pritisne se na gumb *Update*. Taj gumb navigira na stranicu */course-professors-update.* Ta stranicu ima identičnu formu kao i stranica */course-professors-create.*



Slika 5.12 - ažuriranje predmet profesora

Jedina razlika je kada se popuni forma i pritisne gumb za ažuriranje, izvrši se upit updateCourseProfessors te vraća se nazad na stranicu „/course-professors“ .

Programski kod 5.23 – ažuriranje podatka o kolegiju profesora

```
const [updateCourseProfessors] = useLazyQuery(UPDATE_COURSE_PROFESSORS_QUERY);

  const handleUpdateCourseProfessors = () => {
    updateCourseProfessors({
      variables: {
        id: selectedCourseProfessorID,
        token,
        idmethod: parseInt(idmethod),
        iduser: parseInt(iduser),
        percentage: parseInt(percentage),
        coordinator: parseInt(coordinator),
      },
    });
    navigate('/course-professors');
  };
```

Na taj način se kreira klijentski dio završnog rada za rad s GraphQL tehnologijom. Za ovaj završni rad bilo je potrebno još napraviti isti klijentski dio, ali bez korištenja GraphQL tehnologije.

5.2 Klijentski dio bez GraphQL dijela

Za izradu ovog dijela potrebno je maknuti sve Apollo i GraphQL *npm* pakete. Spremanje podataka i tokena na svim stranicama izvodi se na isti način kao i na GraphQL klijentskom dijelu. Jedna od razlika između klijentskog dijela s i bez GraphQL na primjer u index.js datoteci je ta da se ne koriste Apollo tagovi/elementi.

Programski kod 5.24 – primjer Root elementa

```
const Root = () => {
  return (
    <div>
      <Provider store={store}>
        <PersistGate loading={null} persistor={persistor}>
          <Router>
            <Routes>
              …
          </Router>
        </PersistGate>
      </Provider>
    </div>
  );
};
```

Stranice imaju istu logiku navigiranja s jedne na drugu stranicu kao i u *GraphQL* klijentskom dijelu, mapiranja podataka, itd. Razlike između tih stranicu u programskom kodu su u tome što GraphQL klijentski dio koristi GraphQL upite, mutacije, itd. dok klijentski dio bez GraphQL koristi direktno pozivanje samih ruta. Middleware za oba klijentska dijela je identičan. U sljedećem programskom kodu je prikazana prijava.

Programski kod 5.25 – prijava bez korištenja GraphQL tehnologije

```
const handleSubmit = async event => {
    event.preventDefault();

    try {
      const response = await axios.post('http://localhost:3000/api/users/login', {
        email,
        password
      });

      if (response.data) {

        if (response.data.data.role && response.data.data.role.length > 0) {
            …
        }

        dispatch(setToken(response.data.data.token));
        navigate('/home');
      }
    } catch (error) {
      console.error('Error:', error.message);
    }
  };
```

U sljedećem rogramskom kodu prikazana je stranica */document-template* bez korištenja GraphQL tehnologije.

Programski kod 5.26 – pozivanje svih dokumenata bez GraphQL tehnologije

```
useEffect(() => {
    const fetchDocumentTemplates = async () => {
      try {
        const response = await axios.get('http://localhost:3000/api/schemas', {
          headers: { Authorization: `Bearer ${token}` },
        });

        if (response.data) {
          setDocumentData(response.data);
        }
      } catch (error) {
        console.error('Error:', error.message);
      }
    };
```

```
    fetchDocumentTemplates();
  }, [token]);

  const handleButtonClick = async (template) => {
    setLoading(true);

    try {
      const config = {
        headers: { Authorization: `Bearer ${token}` },
        params: { name: template },
      };

      const response = await axios.get('http://localhost:3000/api/schemas',
config);

      if (response.data) {
        global.surveyData = response.data;
        setLoading(false);
        navigate('/response-create');
      }
    } catch (error) {
      console.error('Error:', error.message);
      setLoading(false);
    }
  };
```

Na kraju programski kod stranicu za */course-professors* stranicu bez korištenja GraphQL tehnologije.

Programski kod 5.27 – pozivanje kolegija profesora bez korišenja GraphQL-a

```
const handleAddCourseProfessors = async () => {
    try {
      await axios.post(
        'http://localhost:3000/api/courseProfessors',
        {
          idmethod: parseInt(idmethod),
          iduser: parseInt(iduser),
          percentage: parseInt(percentage),
          coordinator: parseInt(coordinator),
        },
        {
          headers: { Authorization: `Bearer ${token}` },
```

```
        }
      );
      navigate('/course-professors');
    } catch (error) {
      console.error('Error adding course professors:', error);
    }
  };
```

Kao što je vidljivo, jedine razlike su u upitima jer se koristi *axios*, a ne GraphQL. U sljedećem poglavlju objasnit će se testiranje i analiza usporedbe dvaju klijentskih dijela završnog rada.

6. TESTIRANJE I ANALIZA PERFORMANSI

U ovom poglavlju bit će objašnjeno kako je provedeno testiranje performansi između dva načina dohvaćanja podataka te će se pružiti konačna analiza tih testova.

Testiranje je provodeno tako da je štoperica bila aktivirana, a zatim su izvršeni pojedinačni upiti i pozivi. Za svaki upit i poziv zasebno, testiran je odziv servera, a rezultati su zabilježeni u tablici koja prikazuje rezultate testiranja performansi i odziva prema serveru:

Stranice	GraphQL upiti	SQL Pogledi
Register	121 milisekundi	89 milisekundi
Login	117 milisekundi	106 milisekundi
Document-template	14 milisekundi	14 milisekunde
Document-template-create	21 milisekundi	8 milisekunda
Document-template(samo jedan dokument)	21 milisekunda	14 milisekunda
fetchDocumentResponse	99 milisekundi	14 milisekundi
addDocumentResponse	22 milisekundi	10 milisekundi
fetchCourseProfessors	40 milisekundi	13 milisekundi
fetchMethods	13 milisekundi	8 milisekundi
fetchCourseProfessors (jedan profesor)	17 milisekundi	15 milisekundi
fetchProfessors	13 milisekundi	9 milisekundi

addCourseProfessors	24 milisekundi	16 milisekundi
updateCourseProfessors	19 milisekundi	10 milisekundi

U prosjeku, GraphQL upiti se izvršavaju za 41,65 milisekundi, dok SQL Pogledi, odnosno upiti bez korištenja GraphQL-a, zahtijevaju 25,08 milisekunde. To ukazuje da su GraphQL upiti u prosjeku sporiji za 16,57 milisekunde. Na temelju dobivenih rezultata može se zaključiti da zbog dodatne kompleksnosti koju GraphQL sustav donosi, izvršavanje određenih akcija prema bazi podataka može usporiti React aplikaciju za do 17 milisekundi. U okviru ovog završnog rada, ta razlika se smatra beznačajnom i teško primjetnom. Međutim, kod korištenja GraphQL sustava na bazama podataka ili sustavima s većom količinom podataka, ovaj problem bi mogao postati značajniji i zahtijevati dodatne napore za poboljšanje performansi.

Iz perspektive običnog korisnika koji samo koristi klijentski dio aplikacije i unosi podatke, prosječno produženje od 16,57 milisekundi u manjim aplikacijama vjerojatno ne bi bilo primjetno. Međutim, u većim aplikacijama ovo produženje vremena moglo bi postati primjetno. S perspektive programera, rad s GraphQL sustavom zahtijeva dodatne napore kako bi se poboljšale performanse cijele aplikacije. Također, izrada middlewarea s GraphQL-om može trajati duže zbog dodatne kompleksnosti koju ovaj sustav donosi, dok je izrada aplikacija bez korištenja GraphQL-a nešto jednostavnija.

7. ZAKLJUČAK

Kada se izrađuju web aplikacije, postoje različiti pristupi za njihovu izradu. Klasičan pristup uključuje pozivanje API ruta koje koriste SQL Poglede i GraphQL tehnologiju. Prilikom izrade middlewarea s upotrebom GraphQL tehnologije, važno je razmotriti vrste podataka koje će aplikacija koristiti i stvaranje odgovarajućih GraphQL čvorova i elemenata. Za korištenje GraphQL tehnologije na klijentskoj strani, često se koristi Apollo Client tehnologija, koja olakšava integraciju GraphQL tehnologije u klijentsku stranu aplikacije.

Kada se uspoređuju ova dva pristupa izradi web aplikacija, potrebno je također obratiti pažnju na performanse aplikacije. To uključuje analizu odaziva nakon izvršavanja GraphQL upita, odgovora nakon poziva određenih ruta, ukupno trajanje izrade aplikacije, složenost sustava i druge faktore. GraphQL tehnologija dodaje određenu složenost u izradi aplikacija, a njeno utjecanje na vrijeme odaziva može se primijetiti, posebno kod velikih količina podataka. Najveća korist od upotrebe GraphQL tehnologije obično dolazi do izražaja u složenim sustavima s mnogo tablica i kompleksnim vezama između njih.

S druge strane, tehnologija SQL Pogleda obično se koristi u manje složenim sustavima gdje nema velikog broja tablica i složenih struktura podataka. GraphQL tehnologija se bolje koristi u složenijim sustavima kako bi pojednostavila proces izrade aplikacija za programere.

8. LITERATURA

[1] GraphQL Foundation, Meta, GraphQL (Learn), 2023
https://graphql.org/learn/

[2] Stephen Grider, GraphQL with React: The Complete Developers Guide, srpanj 2023
https://www.udemy.com/course/graphql-with-react-course/

[3] Apollo, ApolloGraphQL, Introduction to Apollo Client, 2023
https://www.apollographql.com/docs/react/

[4] DBeaver, DBeaver, Wiki, 2023,
https://github.com/dbeaver/dbeaver/wiki

[5] NodeJS, OpenJS Foundation, 2023
https://nodejs.org/en/about

9. OZNAKE I KRATICE

QL – Query Language

SQL – Structured Query Language

VUB – Veleučilište u Bjelovaru

JS – JavaScript

Gql -GraphQL

NPM – NodeJS Package Manager

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

JSX – JavaScript XML

10. SAŽETAK

Naslov: Optimizacija pristupa podacima pomoću GraphQL i SQL pogleda u React aplikaciji

U ovome završnome objašnjeno je funkcioniranje i korištenje GraphQL tehnologije naspram korištenja SQL Pogledi tehnologije kod izrada React aplikacija. Objašnjeno je kako se stvaraju čvorovi u GraphQL tehnologiji i izrađuju upiti. Objašenja je baza podataka korištena u završnom radu te svaki pojedini čvor. Objašnjen je i sustav izrade React aplikacije koristeći Apollo Client tehnologiju, kako se pomoću te tehnologije izvršavaju GraphQL upiti. Objašnjena je izrada React aplikacija, prelazak između jedne na drugu stranicu, spremanje podataka poput tokena, umjesto korištenja GraphQL tehnologije način poziva ruta. Na kraju, je objašnjeno testiranje i analiza usporedbe obaju tehnologija te su prikazani i analizirani rezultati tih tehnologija.

Ključne riječi: GraphQL, React, SQL Pogledi, performanse, Apollo Client

11. ABSTRACT

Title: Optimizing Data Access Using GraphQL and SQL Views in a React Application

In this final thesis functionality and the usage of the GraphQL technology is explained against using the SQL Views technology when making React applications. It is explained how are nodes and queries made in a GraphQL technology. The database is also explained in this final thesis and every node used in this final thesis. The production system of making React applications using Apollo Client technology is explained, how with the help of the technology the GraphQL queries are being executed. The making of React applications is explained, like navigating from one page to the other, storing dana like tokens and instead of using GraphQL technology how are the routes called. In the end, testing and analyzing results of the comparison between both technologies is shown and analyzed.

Keywords: GraphQL, React, SQL Views, performance, Apollo Client

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

ocument subtitle]

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 27.10.2023.	Filip Gašpar	Filip Gašpar

U skladu s čl. 58, st. 5 Zakona o visokom obrazovanju i znanstvenoj djelatnosti, Veleučilište u Bjelovaru dužno je u roku od 30 dana od dana obrane završnog rada objaviti elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru u nacionalnom repozitoriju.

Suglasnost za pravo pristupa elektroničkoj inačici završnog rada u nacionalnom repozitoriju

Filip Gašpar
ime i prezime studenta/ice

Dajem suglasnost da tekst mojeg završnog rada u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu bude pohranjen s pravom pristupa (zaokružiti jedno od ponuđenog):

a) Rad javno dostupan
b) Rad javno dostupan nakon _____ (upisati datum)
c) Rad dostupan svim korisnicima iz sustava znanosti i visokog obrazovanja RH
d) Rad dostupan samo korisnicima matične ustanove (Veleučilište u Bjelovaru)
e) Rad nije dostupan.

nent subtitle]

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 27.10.2023.

Filip Gašpar
potpis studenta/ice