Web servis za komunikaciju s NoSQL modelom u relacijskoj bazi podataka

Juratovac, Ivan

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Bjelovar University of Applied Sciences / Veleučilište u Bjelovaru**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:144:793078

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-09-15**

Repository / Repozitorij:

Repository of Bjelovar University of Applied Sciences - Institutional Repository

VELEUČILIŠTE U BJELOVARU
STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

WEB SERVIS ZA KOMUNIKACIJU S NOSQL MODELOM U RELACIJSKOJ BAZI PODATAKA

Završni rad br. 07/RAČ/2023

Ivan Juratovac

Bjelovar, listopad 2023.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Ivan Juratovac** JMBAG: **0314022914**

Naslov rada (tema): **Web servis za komunikaciju s NoSQL modelom u relacijskoj bazi podataka**

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

2. ZADATAK ZAVRŠNOG RADA BROJ: 07/RAČ/2023

U sklopu završnog rada potrebno je:
1. Razviti relacijski model baze za pohranu JSON podataka.
2. Predložiti i opisati tehnologiju za validaciju ulaznih JSON podataka.
3. Izraditi web servis za komunikaciju s NoSQL modelom u relacijskoj bazi podataka koristeći Node.js tehnologiju
4. Upravljati tokenima za autentikaciju i autorizaciju web servisa za komunikaciju s NoSQL modelom u relacijskoj bazi podataka.
5. Izraditi kolekciju testova web servisa za komunikaciju s NoSQL modelom u relacijskoj bazi podataka

Datum: 18.07.2023. godine

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

Sadržaj

1. UVOD **1**

2. OPISI KORIŠTENIH TEHNOLOGIJA **2**

2.1 PostgreSQL i DBeaver *2*

2.2 Postman i Node.js *4*

3. ODABIR POSTUPKA VALIDACIJE PODATAKA **5**

4. IZRADA BAZE PODATAKA **6**

4.1 Model baze podataka *6*

4.2 NoSQL pristup *8*

5. IZRADA WEB SERVISA **9**

5.1 Ruter *10*

5.2 Kontroler *11*

5.3 Servis *13*

5.4 Spajanje na bazu *13*

6. VALIDACIJA PODATAKA SHEMAMA **15**

6.1 Ajv.js *15*

6.2 Korištenje shema u Node.js rutama *16*

6.3 Univerzalna ruta *17*

7. ZAŠTITA PODATAKA **21**

7.1 Bcrypt *21*

8. AUTORIZACIJA KORISNIKA **25**

9. KORIŠTENJE POSTMANA ZA TESTIRANJE **28**

9.1 Analiza testne skripte *31*

10. ZAKLJUČAK **33**

11. LITERATURA **34**

12. OZNAKE I KRATICE **36**

13. SAŽETAK **37**

14. ABSTRACT **38**

1. UVOD

Tehnologije i metode za upravljanje bazama podataka igraju ključnu ulogu u razvoju modernih aplikacija, pružajući okruženja za skladištenje i organizaciju podataka. Tradicionalne relacijske baze podataka već dugo vrijeme predstavljaju standard za pouzdano i konzistentno pohranjivanje strukturiranih podataka. S druge strane, NoSQL baze podataka nude fleksibilniji pristup za pohranu nestrukturiranih ili polustrukturiranih podataka, kao što su JSON dokumenti. Ovaj završni rad usmjerava se na spajanje navedenih tehnologija kako bi se ostvarila efikasna i skalabilna obrada JSON podataka unutar PostgreSQL baze podataka. Cilj ovog završnog rada je kombiniranje prednosti relacijskih baza podataka, kao što su integritet podataka i SQL upiti, s fleksibilnošću NoSQL pristupa za pohranu i manipulaciju JSON podacima.

U drugom poglavlju se objašnjava kako će se iskoristiti PostgreSQL kao pouzdana osnova za pohranu podataka, dok će se za validaciju JSON podataka koristiti Ajv.js unutar Node.js servera. Uz to, za testiranje i analizu aplikacije koristit će se Postman kao ključni alat za provjeru funkcionalnosti. U trećem poglavlju uspoređuju se metode validacije ulaznih podataka i odabire se metoda koja je najprikladnija za ostvarivanje ciljeva ovog završnog rada. U četvrtom poglavlju opisuje se stvaranje relacijske baze podataka koja ima klasične tablice i tablicu *documents* koja ima stupce s klasičnim tipovima podataka i jedan stupac u kojem može pohranjivati JSON dokumente. U petom poglavlju objašnjava se općenito koje datoteke i funkcionalnosti se nalaze unutar Node.js web servisa i kakva hijerarhija je korištena za raspodjelu zadataka između različitih modula. Šesto poglavlje opisuje načine na koji su iskorištene sheme za validaciju ulaznih podataka na Node.js web servisu prije nego se spreme u bazu podataka. Sedmo poglavlje govori o zaštiti korisničkih podataka korištenjem Bcrypt biblioteke za hashiranje, verificiranje i usporedbu lozinki. Osmo poglavlje pojašnjava korištenje JSON web tokena i dohvaćanje podataka o korisniku iz tablica za autorizaciju korisnika. U devetom poglavlju opisuje se korištenje Postmana za testiranje rada cijele aplikacije korištenjem testnih skripti unutar zahtjeva. Na kraju rada navodi se zaključak u kojem se ukratko opisuju rezultati izrade cijele aplikacije.

2. OPISI KORIŠTENIH TEHNOLOGIJA

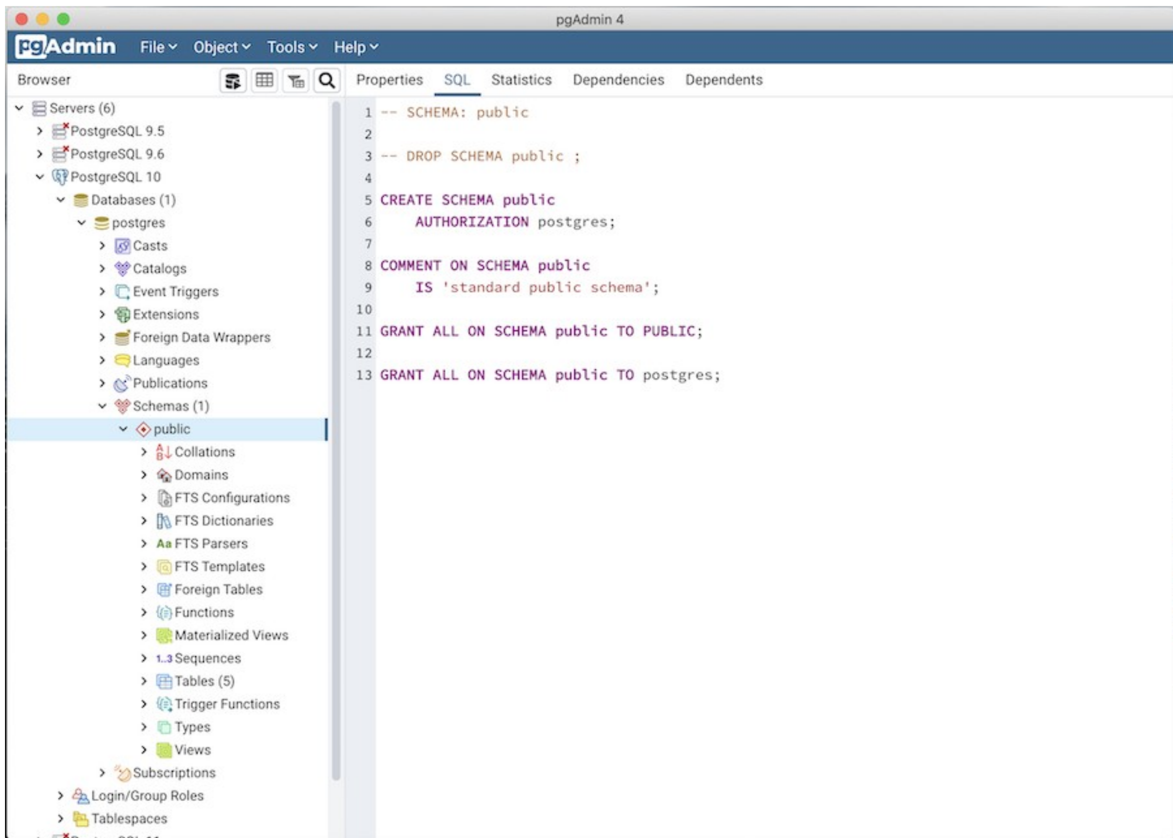
Ovo poglavlje pruža pregled tehnologija koje su implementirane u završnom radu. Koristi se PostgreSQL baza podataka, Node.js web servis i Postman za testiranje rada cijele aplikacije.

2.1 PostgreSQL i DBeaver

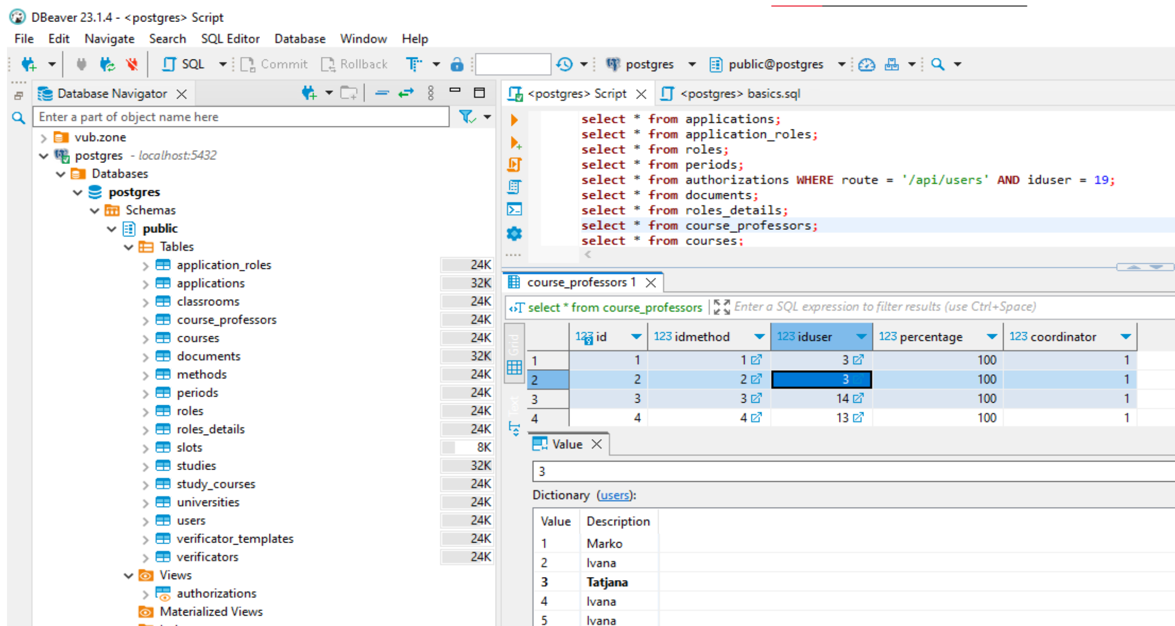
PostgreSQL je besplatan objektno-relacijski sustav baze podataka koji koristi SQL programski jezik i proširuje ga s vlastitim značajkama koje sigurno pohranjuju i skaliraju najsloženija radna opterećenja podataka [1]. Ima snažnu podršku za rukovanje s JSON podacima, uključujući funkcije i operatore za postavljanje upita i upravljanje s JSON dokumentima. Nadalje, PostgreSQL-ov JSONB tip podataka omogućuje učinkovito pohranjivanje i indeksiranje JSON podataka, što ga čini prikladnim izborom za rad s JSON-om u sustavu relacijske baze podataka u ovom završnom radu.

Službeni PostgreSQL instalacijski paket uključuje i pgAdmin, izborni grafički administrativni alat za PostgreSQL koji pruža korisničko sučelje (slika 2.1) za upravljanje bazama podataka, tablicama i upitima. Međutim, ovaj alat ima sučelje koje nije optimalno za rad i postoje problemi s performansama zbog kojih je rad s bazama podataka teži no što treba biti.

Stoga je korišten alat DBeaver. To je besplatan razvojni alat za baze podataka za programere, administratore baza podataka, analitičare i sve koji rade s podacima. Podržava razne SQL baze podataka među kojima je i PostgreSQL [3]. Primjer korisničkog sučelja alata DBeaver nalazi se na slici 2.2.



Slika 2.1: Korisničko sučelje alata pgAdmin [2]



Slika 2.2: Korisničko sučelje alata DBeaver

2.2 Postman i Node.js

Postman je API platforma za izgradnju, testiranje i korištenje API-ja [4]. Moguće je koristiti web aplikaciju ili preuzeti desktop aplikaciju. Postman je organiziran na način da je moguće stvarati više radnih prostora. U svakom radnom prostoru se stvaraju kolekcije, a unutar kolekcija se stvaraju zahtjevi.

Kolekcije služe kako bi se grupirali i organizirali zahtjevi prema resursima na koje utječu. Također, služe tome da se zahtjevi mogu testirati određenim redoslijedom.

U zahtjeve je moguće pisati testne skripte koje očekuju rezultat prema određenim uvjetima napisanim u JavaScript programskom jeziku [5]. JavaScript koji se koristi u testnim skriptama ima specijaliziranu sintaksu koju pruža aplikacija Postman.

Zahtjevi se dijele na GET, POST, PUT i DELETE vrstu zahtjeva. GET zahtjevi dohvaćaju resurse sa ili bez dodatnih parametara u tijelu zahtjeva po kojima se filtriraju ciljani resursi. POST zahtjevi kreiraju nove resurse sa svojstvima opisanim u tijelu zahtjeva. PUT zahtjevi izmjenjuju postojeće resurse sa svojstvima opisanim u tijelu zahtjeva. DELETE zahtjevi brišu resurse koji zadovoljavaju određene uvjete (npr. pošalje se identifikacijski broj resursa za brisanje).

Node.js je besplatno JavaScript okruženje dizajnirano za izgradnju skalabilnih mrežnih aplikacija [6]. U ovom završnom radu, Node.js služi kao web servis ili *middleware*, čija je zadaća obraditi zahtjeve koje šalje Postman i odgovore koji dolaze iz PostgreSQL baze podataka.

Koristi se minimalno i fleksibilno okruženje za izradu web aplikacija u Node.js-u pod nazivom Express koje pruža robustan skup značajki za web i mobilne aplikacije. Express služi za stvaranje robusnog API-ja na brz i jednostavan način korištenjem tankog sloja temeljnih značajki za web aplikacije, bez zaklanjanja značajki Node.js-a [7].

3. ODABIR POSTUPKA VALIDACIJE PODATAKA

Validacija podataka ključna je komponenta svakog sustava za upravljanje bazama podataka, posebice kada se radi s kompleksnim formatima kao što je JSON. U ovom završnom radu, razmotrena su različita rješenja i pristupi kako bi se odabrao najprikladniji način validacije JSON podataka. Prvo je trebalo izabrati na koji način validirati JSON podatke. Osmišljena su dva načina za validaciju JSON podataka: check constraint sa shemom na tablicama u PostgreSQL bazi podataka i Ajv.js validacija podataka shemom unutar Node.js okruženja. U sljedećoj tablici nalazi se usporedba navedenih opcija.

Tablica 3.1: Usporedba načina validacije JSON podataka

PostgreSQL check constraint	Ajv.js
validacija podataka odvija se na razini baze podataka, pružajući centralizirani i dosljedni mehanizam validacije	dosljednost validacije u različitim aplikacijama ili sustavima koji su u interakciji s bazom podataka
validacija unutar baze podataka može biti učinkovitija jer eliminira potrebu za dodatnom mrežnom komunikacijom i iskorištava optimizirano izvršavanje upita baze podataka	više kontrole nad procesom validacije
za ažuriranje logike validacije potrebno je promjeniti shemu baze podataka	ažuriranje logike validacije jednostavno je i ne zahtijeva promjenu sheme baze podataka

U radu se najprije pokušalo validirati JSON podatke unutar PostgreSQL baze podataka koristeći prilagođene funkcije koje su pronađene na GitHub poveznici navedenoj u literaturi [8].

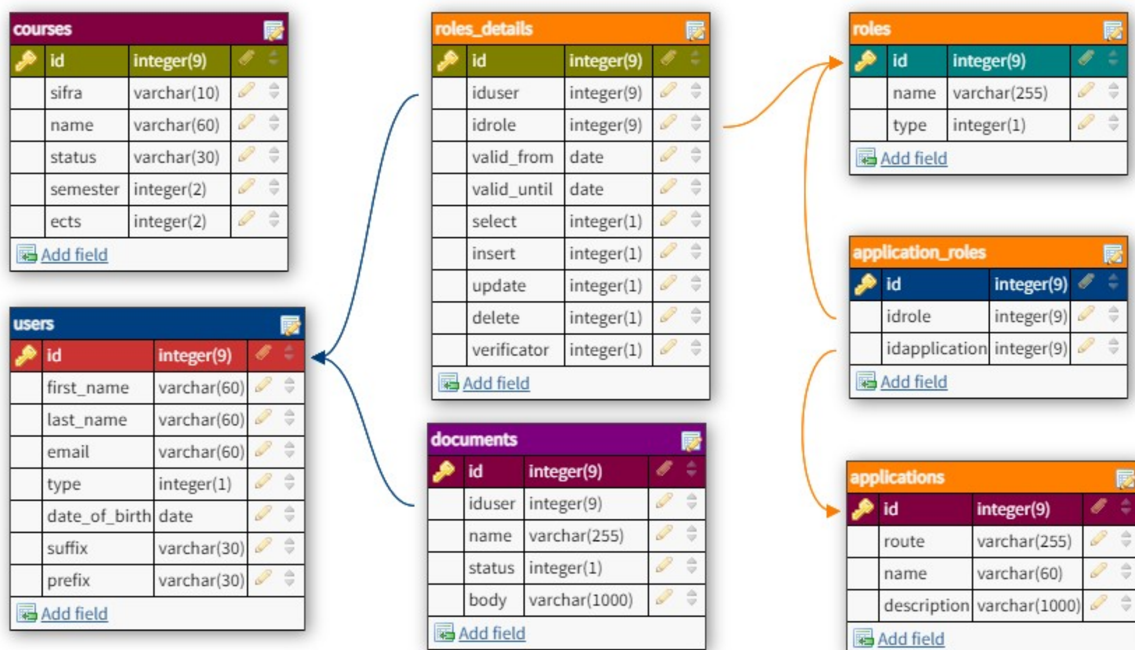
Ideja je bila da se na tablicama u bazi nalaze prilagođeni check constrainti koji ne bi dopuštali unos podataka u tablicu ako format JSON podataka za unos ne odgovara formatu JSON sheme unutar check constrainta. Međutim, nije postojala validacija emaila i implementacija se činila kompleksnija od korištenja Ajv.js, zbog čega je odlučeno koristiti Ajv.js za validaciju podataka.

4. IZRADA BAZE PODATAKA

U ovom poglavlju opisuje se model baze podataka i njegova implementacija u PostgreSQL bazi podataka.

4.1 Model baze podataka

Model baze podataka korišten u ovom završnom radu dio je sustava projekta VUBApp. Iz navedenog projekta iskorišteno je sedam tablica u svrhu izrade ovog završnog rada. Model baze podataka sa sedam tablica prikazan je na slici 4.1.



Slika 4.1: Model baze podataka

U tablici *users* se nalaze podaci o korisnicima koji koriste aplikacije. Spremaju se podaci o imenu i prezimenu korisnika, email, lozinka, broj koji označava tip korisnika (npr. za profesora piše 1), sufiks (npr. za višeg predavača piše v.pred.) i prefiks (npr. za doktora znanosti piše dr.sc.).

U tablicu *documents* se spremaju odgovori na razne forme koje korisnici ispunjavaju vlastitim podacima. U polja se sprema ID korisnika koji je popunio formu (vezano na tablicu users stranim ključem), naziv popunjene forme, status potpisanosti dokumenta i JSON podatak sa svim poljima koje je korisnik ispunio unutar forme.

Tablica *roles_details* određuje koji korisnik ima koju ulogu. Zapis u ovoj tablici određuje koliko dugo vrijede određena prava za korisnika i koju vrstu ruta korisnik smije pozivati. Vrste uloga nalaze se u tablici roles. Primjer uloga su administrator ili korisnik.

U tablici *applications* nalaze se nazivi Node.js ruta te ime i opis aplikacije. Ova tablica povezana je s tablicom roles preko tablice *application_roles* koja sadrži ID zapisa iz tablica *applications* i *roles*.

Tablica *courses* je u ovom završnom radu nepovezana tablica u koju se spremaju podaci o predmetima Veleučilišta u Bjelovaru. Tablica je uvezena iz VUBApp modela baze podataka kako bi poslužila za primjer korištenja univerzalne rute za manipulaciju tablicama /api/table/:tableName.

Pored navedenih tablica stvoren je i pogled (engl. *view*) nazvan *authorizations* koji spaja tablice *applications*, *application_roles* i *roles_details* kako bi prikazao najbitnije podatke za autorizaciju korisnika na jednom mjestu. Upit za kreiranje ovog pogleda nalazi se u programskom kodu 4.1. U upitu se nalaze ID-evi aplikacija, rute, ID-evi korisnika i prava korisnika za korištenje HTTP metoda GET, POST, PUT i DELETE.

Programski kod 4.1: Kreiranje pogleda authorizations

```
CREATE VIEW authorizations AS
SELECT a.id idapplication, a.route, 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;
```

4.2 NoSQL pristup

Pojam NoSQL ima više značenja. Jedno značenje je „non-SQL” gdje se podaci ne spremaju u standardnim relacijskim bazama, nego isključivo u različitim drugim formatima kao što su JSON dokumenti. Alternativno značenje pojma NoSQL je „not only SQL” što se može tumačiti kao model baze podataka s minimalnim korištenjem SQL baza podataka [9].

Pristup koji se koristio u ovom završnom radu je „not only SQL” pristup. Zbog autorizacije korisnika stvorene su tablice *users*, *roles*, *roles_details*, *applications* i *application_roles* u PostgreSQL relacijskoj bazi podataka. Za njihovo stvaranje korišteni su jednostavni CREATE upiti, a za stvaranje relacija između tablica korišteni su jednostavni ALTER upiti za dodavanje stranih ključeva. U ostatku baze podataka ne postoje nikakvi paketi, funkcije ni procedure. Na taj način je ostvaren model baze podataka s minimalnim korištenjem SQL-a.

U modelu postoji tablica *documents* u koju se spremaju dokumenti u JSON formatu. Kontrolu ulaznih podataka ne vrši baza jer se svi podaci nalaze u jednom stupcu. Umjesto toga, kontrola se vrši shemama na Node.js-u.

Više informacija o univerzalnoj ruti i korištenju shema za kontrolu ulaznih podataka nalazi se u poglavlju 6.

5. IZRADA WEB SERVISA

Datoteke u Node.js-u za ovaj završni rad organizirane su u hijerarhiju koja dijeli obavljanje zadataka na tri skupine: ruteri, kontroleri i servisi.

Glavna datoteka za pokretanje nalazi se iznad svih navedenih skupina i zove se *app.js*. U toj datoteci su uvezene sve glavne komponente koje se koriste u aplikaciji kao što su: ruteri, *body-parser* i Express okruženje. Prikaz uvoza komponenti u glavnu datoteku nalazi se u programskom kodu 5.1.

Programski kod 5.1: Uvoz komponenti u glavnu Node.js datoteku

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const usersRouter = require('./routers/usersRouter');
const schemasRouter = require('./routers/schemasRouter');
const tableRouter = require('./routers/tableRouter');
```

Body-parser je funkcija u Express okruženju koja se koristi za parsiranje podataka iz dolaznog zahtjeva u različite formate kao što su JSON i URL-encoded formati. Podaci se nakon parsiranja mogu koristiti u Node.js aplikaciji pomoću *req.body* JavaScript objekta koji se u daljnjem tekstu naziva tijelo zahtjeva. Korištenje *body-parser* funkcije prikazano je programskim kodom 5.2.

Programski kod 5.2: Korištenje body-parser funkcije

```
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
```

Glavna datoteka korištenjem Express okruženja postavlja rutere čiji kod se izvršava za svaki zahtjev koji se odnosi na definirani početni dio URL-a. U programskom kodu 5.3

glavna datoteka koristi rutere pozivanjem metode *use* koja postavlja početni dio URL-a, a zatim daje zadatak ruterima da definiraju ostatak rute.

Programski kod 5.3: Definiranje početnih dijelova ruta

```
app.use('/api/users', usersRouter);
app.use('/api/schemas', schemasRouter);
app.use('/api/table', tableRouter);
```

5.1 Ruter

Ruter obrađuje dolazne zahtjeve i usmjerava ih odgovarajućem kontroleru na temelju tražene rute. Ruter definira rute i njihove odgovarajuće funkcije za specifične HTTP metode GET, POST, PUT i DELETE korištenjem Express okruženja. Ove rute se mogu pozvati s klijentske strane ili iz Postmana. U ruterovim metodama se zadaje ostatak URL-a koji definira cijelu rutu za pozivanje, a zatim se pozivaju funkcije u nizu u kojem su napisane. Na programskom kodu 5.4 prikazani su primjeri korištenja ruterovih metoda na način na koji se koristi za većinu ruta u ovom završnom radu. U ovom programskom kodu prva funkcija provjerava ima li korisnik autorizaciju za pozivanje ove rute, a druga funkcija je metoda iz odgovarajućeg kontrolera.

Programski kod 5.4: Definiranje ruta korištenjem rutera

```
// GET /api/schemas/response
router.get('/response',
    auth.authorize,
    schemasController.getResponsesByName
);

// POST /api/schemas
router.post('/',
    auth.authorize,
    schemasController.postSchema
);
```

5.2 Kontroler

Kontroler sadrži logiku za rukovanje određenim rutama. Prima zahtjev od rutera, komunicira s potrebnim servisima i priprema odgovor za vraćanje. U ovom završnom radu kontroler je implementiran kao statička klasa. Statičke klase sadrže metode koje se mogu pozivati bez stvaranja objekta. U kontroleru se nalaze metode koje ruter poziva za svaku rutu. Prednost korištenja klase umjesto zasebnih funkcija je lakši izvoz. Dodavanjem nove funkcije u kontroler može se dogoditi da programer zaboravi dodati tu funkciju u *module.exports* objekt za izvoz. Kada se doda nova metoda u klasu, onda se ona automatski izvozi pošto se izvoženjem cijele klase (primjer programski kod 5.5) izvoze i sve metode koje se u klasi nalaze. Zbog navedenog načina izvoza, moguće je pozivati samo metode (funkcije koje se nalaze unutar klase), a funkcije ne.

Programski kod 5.5: Izvoz klase UsersController

```
module.exports = UsersController;
```

Programski kod 5.6 prikazuje primjer metode unutar kontrolera. Metode unutar kontrolera često prate sličan tijek izvođenja.

Najprije se iz tijela zahtjeva uzimaju parametri koji su potrebni za daljnje izvršavanje metode. Zatim, postoji nekoliko uvjeta koji provjeravaju ulazne parametre i zaustavljaju daljnje izvršavanje koda ako pronađu neku grešku. Tijekom izvođenja metode pozivaju se servisi koji vraćaju grešku ili tražene podatke. Ako su svi ulazni parametri ispravni i sve metode su vratile tražene podatke, onda metoda klijentu vraća odgovor koji se sastoji od statusnog koda i JSON podatka. Za obradu očekivanih i neočekivanih grešaka koristi se *try-catch* blok koji će vratiti odgovarajući statusni kod i JSON poruku s obzirom na tip greške.

Programski kod 5.6: Metoda postSchema u klasi SchemasController

```
static async postSchema(req, res) {
    const { name, body } = req.body;
    if (name == null) {
        return res.status(400).json({
            message: 'Missing required field: name'
        });
    }
    if (body == null) {
        return res.status(400).json({
            message: 'Missing required field: body'
        });
    }

    if (name.match(/[\\\/:\*\?"<>\|]/)) {
        return res.status(400).json({
            ...
        });
    }

    try {
        schemasService.saveSchema(name, body);
        res.json({ message: 'Schema saved successfully.' });
    }
    catch (err) {
        console.log(err);
        if (err.message === 'invalid JSON') {
            res.status(400).json({ message: 'Invalid JSON.' });
        }
        else {
            res.status(500).json({ message: 'Internal server error.' });
        }
    }
}
```

5.3 Servis

Servis izvodi specifične operacije vezane uz određenu domenu ili funkcionalnost. Može komunicirati s bazama podataka, vanjskim API-jima ili obavljati druge zadatke potrebne za ispunjavanje zahtjeva. Servis, kao kontroler, je implementiran kao statička klasa s metodama. Metode u servisu se najčešće sastoje od varijabli *query* i *values*. U varijablu *query* se upisuje tekst upita za bazu podataka, dok u varijablu *values* se dodaje polje s vrijednostima koje se umeću u upit na mjesto gdje se pojavljuje znak dolar i brojka. Zatim se varijable *query* i *values* pošalju datoteci *db.js* u funkciju *query* kao parametri. Ta funkcija vrati redove od kojih se svi redovi, ili samo prvi red, vraćaju nazad kontroleru kao rezultat. Primjer metode u servis klasi nalazi se u programskom kodu 5.7.

Programski kod 5.7: Metoda getUserByEmail u klasi UsersService

```
static async getUserByEmail(email) {
    const query = 'SELECT * FROM users WHERE email = $1';
    const values = [email];
    const { rows } = await db.query(query, values);
    return rows[0];
}
```

5.4 Spajanje na bazu

Datoteka *db.js* je zadužena za spajanje na bazu podataka i direktno dohvaća rezultate prema upitu i parametrima poslanima u funkciju *query*. Iz datoteke *credentials.js* dohvaća podatke za povezivanje s bazom podataka. Povezivanje s bazom podataka odvija se prilikom pokretanja aplikacije. Programski kod 5.8 prikazuje sadržaj datoteke *db.js*.

Programski kod 5.8: Sadržaj datoteke db.js

```
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
};
```

6. VALIDACIJA PODATAKA SHEMAMA

U ovom završnom radu koriste se JSON sheme za validaciju podataka u Node.js-u. JSON shema omogućuje označavanje i provjeru valjanosti JSON dokumenata na temelju određene sheme. Ona određuje dopuštene vrste podataka, formate i ograničenja JSON podataka. Odlučeno je koristiti Ajv.js za validaciju podataka shemama zbog razloga navedenih u poglavlju 3.

6.1 Ajv.js

Ajv.js je paket koji se može instalirati i uvesti u Node.js. Ajv uzima shemu za JSON podatke i pretvara ju u JavaScript kod koji provjerava valjanost podataka u skladu sa shemom. Ajv kompajlira sheme u funkcije i sprema ih u predmemoriju koristeći shemu kao ključ u mapi, tako da se sljedećim korištenjem isti objekt sheme neće ponovno kompajlirati [10].

Cijela funkcionalnost Ajv paketa je grupirana u jednu datoteku nazvanu validation.js. U toj datoteci je definirana klasa *Validation* koja ima metodu *validate*. U toj metodi su implementirane funkcije Ajv paketa koji je uvezen u ovu datoteku.

U programskom kodu 6.1 prikazana je metoda *validate*. Metoda prihvaća dva JSON podatka: cijelu shemu i podatak za provjeru. Ova metoda vraća greške koje se javljaju tijekom provjere podataka ili ništa ako su podaci valjani.

Metoda na početku kompajlira cijelu shemu i sprema ju u varijablu *valid*. Ta varijabla se ponaša kao funkcija koja vraća boolean vrijednost s obzirom na to je li podatak koji se provjerava u skladu sa shemom ili nije. Ako je podatak u skladu sa shemom, onda metoda vraća *null* jer nema nikakvih grešaka. Ako podatak nije u skladu sa shemom, onda metoda vraća polje s formatiranim porukama svih grešaka koje je Ajv.js pronašao.

Programski kod 6.1: Metoda validate u klasi Validation

```
static validate(schema, data) {
    try {
        var valid = ajv.compile(schema);
    }
    catch (err) {
        console.log(err);
        return 'Invalid schema';
    }

    if (!valid(data)) {
        console.log(valid.errors);
        let errors = [];
        for (let error of valid.errors) {
            if (error.keyword == 'additionalProperties') {
                errors.push(`must NOT have additional property '${
                    error.params.additionalProperty
                }'`);
            }
            else if (error.instancePath == '') {
                errors.push(error.message);
            }
            else {
                errors.push(`${error.instancePath} ${error.message}`);
            }
        }
        return errors;
    }
    return null;
}
```

6.2 Korištenje shema u Node.js rutama

U Node.js-u je stvorena ruta /api/schemas za dohvaćanje i spremanje shema na Node.js server te ruta /api/schemas/response za dohvaćanje i spremanje odgovora koji su u skladu sa shemama na PostgreSQL bazu podataka.

Ruta POST /api/schemas prihvaća dva parametra: naziv sheme i tijelo sheme. Ta ruta će stvoriti novu shemu na Node.js serveru ukoliko je naziv sheme u skladu s pravilima naziva datoteke operativnog sustava i ukoliko je JSON tijelo sheme ispravno formatirano.

Ruta GET /api/schemas prihvaća opcionalan parametar: naziv sheme. Ako je parametar poslan, onda kontroler vraća shemu koja sadrži taj naziv. Ako parametar nije poslan, onda kontroler vraća nazive svih shema koje klijent može dohvatiti.

Ruta POST /api/schemas/response prihvaća dva parametra: naziv sheme i odgovor na tu shemu u obliku JSON podatka. Kontroler dohvaća shemu s odgovarajućim nazivom i provjerava valjanost odgovora sa shemom koju je dohvatio, uz uvjet da je poslani naziv sheme valjan. Ako provjera nije uspješna, onda se klijentu vraća poruka u kojoj se opisuju greške o poslanom odgovoru. Ako je provjera uspješna, onda se u tablicu *documents* na bazi podataka sprema odgovor i klijentu se šalje poruka da je zahtjev uspješno izvršen.

Ruta GET /api/schemas/response prihvaća jedan parametar: naziv sheme prema kojoj su odgovori poslani. Klijentu se iz baze podataka dohvaćaju svi odgovori prema navedenoj shemi ako ta shema postoji.

6.3 Univerzalna ruta

Univerzalna ruta služi dohvaćanju, unosu i mijenjanju podataka iz bilo koje tablice. Klijent mora imati autorizaciju za akciju i za tablicu kako bi joj mogao pristupiti. Također, navedena tablica mora postojati u bazi, a na Node.js serveru mora postojati shema s istim imenom kao i ciljana tablica kako bi klijentov zahtjev bio uspješno izvršen.

U glavnoj datoteci i u ruteru je ruta definirana kao /api/table/:tableName. Kada se u ruti nalazi dvotočka onda je desno od te dvotočke definiran naziv parametra koji se može dohvatiti iz zahtjeva, odnosno objekta *req* pristupanjem atributu *params*. U programskom kodu 6.2 prikazana je definicija rute za ažuriranje podataka u tablici. Na isti način su definirane rute za dohvaćanje i za kreiranje podataka. U njima se odvija autorizacija korisnika, provjerava se valjanost podataka shemama i izvršava se akcija nad tablicom.

Programski kod 6.2: Definiranje rute /api/table/:tableName za ažuriranje podataka

```
router.put('/:tableName',
    auth.authorize,
    tableController.validateTable,
    tableController.updateRow
);
```

U programskom kodu 6.3 prikazana je metoda *validateTable*. Metoda prihvaća objekt *req* popunjen dolaznim zahtjevom, objekt *res* za slanje odgovora i objekt *next* za prelazak u iduću funkciju koja je definirana na ruti. Iz objekta *req.params* stvara se varijabla *tableName* koja ima vrijednost teksta na poziciji *:tableName* u ruti /api/table/:tableName. Prekida se izvršavanje koda i vraća se poruka da nema navedene sheme ako ona ne postoji na Node.js serveru. Izvršava se provjera valjanosti podataka korištenjem sheme koja ima naziv tablice i koja pripada odgovarajućoj HTTP metodi. Ako ima grešaka, one se prikazuju klijentu i zaustavlja se izvršavanje koda. Ako nema grešaka, prijelazi se u iduću funkciju na ruti.

Programski kod 6.3: Metoda validateTable u klasi TableController

```
class TableController {
    static async validateTable(req, res, next) {
        const { tableName } = req.params;
        if (!Object.keys(tableSchema).includes(tableName)) {
            return res.status(404).json({
                message: `No schema found for table ${tableName}.`
            });
        }
        const errors = ajv.validate(tableSchema[tableName][req.method],
                                    req.body);
        if (errors != null) {
            return res.status(400).json({ message: errors });
        }
        next();
    }
    ...
}
```

Za korištenje prethodno navedene metode potrebno je uvesti datoteku *tableSchema.js*. Programski kod 6.4 prikazuje ovu datoteku. U njoj se postavlja izvoz shema koje su potrebne za pristup određenim tablicama. Sheme za tablice koje se ne nalaze u ovom izvozu nije moguće pozivati. Na taj način se štite tablice koje ne bi trebale biti dostupne univerzalnom rutom.

Programski kod 6.4: Izvoz shema za univerzalnu rutu

```
...
const courses = require('./coursesSchema');
...

const schemas = {
  ...
  courses,
  ...
};

module.exports = schemas;
```

Svaka shema za tablicu koja se koristi u univerzalnoj ruti mora imati podsheme koje definiraju ulazne parametre za određenu akciju na tablici. U ovom završnom radu nije napravljena nijedna ruta za brisanje podataka iz tablica, tako da su u shemama definirane podsheme za GET, POST i PUT akcije. Podsheme se definiraju kao konstantni JavaScript objekti koji se zajedno izvoze kako bi im datoteka *tableSchema.js* mogla pristupati.

Unutar podsheme definirano je kakav mora biti tip podatka, koji su obavezni atributi i smije li podatak imati dodatnih atributa koji nisu navedeni pod ključem *properties*. Unutar ključa *properties* navedena su pravila koja svaki atribut mora poštivati. Neka od tih pravila su: tip podatka, veličina teksta, minimalna vrijednost, maksimalna vrijednost i slično. Primjer navedenih pravila i izvoz podshema nalazi se u programskom kodu 6.5.

Programski kod 6.5: Podsheme u shemi za tablicu courses

```
const getSchema = { ... };

const createSchema = {
    "type": "object",
    "properties": {
        "sifra": {
            "type": "string",
            "pattern": '^[A-Z]+[0-9]*$',
            "maxLength": 10
        },
        "name": {
            "type": "string",
            "maxLength": 60
        },
        "status": {
            "type": "string",
            "pattern": '^[0-9]$'
        },
        "semester": {
            "type": "integer",
            "minimum": 1,
            "maximum": 6
        },
        "ects": {
            "type": "integer",
            "minimum": 0,
            "maximum": 11
        }
    },
    "required": ["sifra", "name", "status", "semester", "ects"],
    "additionalProperties": false
};

const updateSchema = { ... };

module.exports = {
    GET: getSchema,
    POST: createSchema,
    PUT: updateSchema
};
```

7. ZAŠTITA PODATAKA

U ovom završnom radu nije moguća samostalna registracija korisnika. Nove korisnike registriraju postojeći korisnici koji imaju ulogu administratora. Na taj način se štiti aplikacija od dolaska novih korisnika koji nemaju i neće imati nikakvu autorizaciju. Međutim, i dalje mora postojati prijava.

Korisnici se mogu prijaviti u aplikaciju korištenjem rute /api/users/login. Za prijavu je potrebno unijeti email i lozinku. Email i ostali podaci se u tablicu *users* spremaju u istom obliku u kojem su uneseni, a lozinka se sprema u hashiranom obliku. Hashiranje se koristi za pretvaranje ulaznih podataka lozinke u niz znakova, slova i brojeva fiksne duljine. Čak i najmanja promjena u ulaznim podacima rezultira potpuno drugačijom izlaznom hash vrijednošću. Za hashiranje lozinki u Node.js-u korišten je paket Bcrypt.

7.1 Bcrypt

Bcrypt je biblioteka koja koristi sol i faktor rada za hashiranje lozinki. Sol je niz znakova koji se dodaje lozinki pri hashiranju kako bi se otežalo njeno probijanje. Faktor rada služi za produljivanje vremena potrebnog da se lozinka hashira, usporedi i verificira kako bi se onemogućili napadi grubom silom (engl. *brute force attacks*) jer svaki pokušaj pogađanja lozinke traje dulje od klasičnih metoda zaštite [11].

Prilikom početnog postavljanja aplikacije, prvog korisnika potrebno je dodati izravno u bazu podataka koristeći INSERT SQL upit. Međutim, ne može se unijeti obična tekstualna lozinka jer to narušava sigurnost aplikacije, a ni prijava neće raditi jer se unesene lozinke automatski hashiraju. Zato je potrebno prethodno hashirati korisnikovu lozinku korištenjem Bcrypta korištenjem datoteke *setup.js*. Pokretanje ove datoteke obavlja se naredbom „node setup.js" u terminalu. Terminal zatim traži unos lozinke nakon čega ispiše hashiranu verziju lozinke koja se kopira i koristi u INSERT upitu u bazi podataka. Primjer pokretanja datoteke *setup.js* nalazi se na slici 7.1. Iz primjera je vidljivo da korištenje iste lozinke rezultira drugačijom hash vrijednošću zbog načina na koji Bcrypt koristi hash funkcije.

```
PS D:\VSC Projects\Geneva NodeJS> node setup.js
Enter password: asdf
$2a$10$gnTLILarnV2Rs04D8xmCReyi761JEKWa5/UfxUu13FvgKTqbEjvnS
PS D:\VSC Projects\Geneva NodeJS> node setup.js
Enter password: asdf
$2a$10$g7rARYO54irUA93vpqEWnehWA3e7dg2tilrrbfzhiKuhnkycg0wnG
PS D:\VSC Projects\Geneva NodeJS>
```

Slika 7.1: Početno hashiranje lozinke u terminalu

Sada je moguće prijaviti se novostvorenim korisnikom. Metoda u kontroleru *UsersController* za prijavu korisnika prikazana je u programskom kodu 7.1. Ova metoda na početku provjerava jesu li ulazni podaci u skladu sa shemom za prijavu. Zatim se iz tijela zahtjeva izvlače email i lozinka koji se koriste za dohvat korisnika kako bi se provjerilo postoji li korisnik s tim emailom u bazi podataka. Nakon toga, koristi se Bcrypt za usporedbu unesene lozinke s hashiranom lozinkom dohvaćenom iz baze podataka. Ako ove provjere prođu, korisniku se generira token i vraćaju mu se podaci o njegovoj autorizaciji.

Programski kod 7.1: Algoritam za prijavu korisnika

```
static async login(req, res) {
    const errors = ajv.validate(usersSchema.createSchema, req.body);
    if (errors != null) {
        return res.status(400).json({ message: errors });
    }

    const { email, password } = req.body;
    try {
        const user = await usersService.getUserByEmail(email);
        if (user == null) {
            throw new Error('Invalid credentials');
        }
        const match = await bcrypt.compare(password, user.password);
        if (!match) {
            throw new Error('Invalid credentials');
        }
        const token = config.generateToken(user);
        const authorizations = await authorizationService
                                    .getUserAuthorizations(user.id);
        let routes = [];
        for (const authorization of authorizations) {
            routes.push(authorization.route);
        }
        let role = {
            get: authorizations[0].get,
            put: authorizations[0].put,
            post: authorizations[0].post,
            del: authorizations[0].del
        }
        res.json({
            data: {
                token: token,
                routes: routes,
                role: role
            }
        });
    }
    catch (err) {
        ...
    }
}
```

Prijavljeni korisnici mogu registrirati nove korisnike korištenjem rute /api/users/register. Prilikom registracije korisnika lozinka se hashira korištenjem metode *hash* iz paketa Bcrypt. Ova metoda je iskorištena u servisu *usersService* za stvaranje novog korisnika u bazi podataka koji ima email i hashiranu lozinku što je prikazano u programskom kodu 7.2.

Programski kod 7.2: Stvaranje novog korisnika

```
static async createUser(email, password) {
    const query = 'INSERT INTO users (email, password) VALUES ($1, $2)';
    const hash = await bcrypt.hash(password, 10);
    const values = [email, hash];
    const { rows } = await db.query(query, values);
    return rows[0];
}
```

8. AUTORIZACIJA KORISNIKA

U prethodnom programskom kodu spominje se generiranje tokena. Za generiranje tokena koristi se datoteka *jwtConfig.js* čiji se sadržaj nalazi u programskom kodu 8.1. Ova datoteka koristi paket JWT koji služi za spremanje informacija o prijavljenom korisniku unutar niza znakova koji se zove token. Podaci o korisniku se šifriraju i spremaju u token korištenjem metode *sign*. Ova metoda prihvaća podatke o korisniku, tajnu hash vrijednost i dodatne parametre kao što je vrijeme isteka tokena. Tajna hash vrijednost se nalazi u datoteci *.env* i koristi se za dodatnu zaštitu korisničkih podataka. Za dohvaćanje podataka iz tokena koristi se metoda *verify*. Ona će izbaciti grešku ako je rok trajanja tokena istekao.

Programski kod 8.1: Generiranje i verifikacija 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
};
```

Nakon prijave, korisnik može koristiti sve izložene rute koje se nalaze na Node.js serveru. Međutim, pri pozivanju svake rute potrebno je obaviti dodatnu autorizaciju

korisnika. Za autorizaciju korisnika se poziva metoda *authorize* u klasi *Authorization* u datoteci *authorization.js*. Na početku ove metode se provjerava ispravnost tokena ako je poslan što je prikazano programskim kodom 8.2.

Programski kod 8.2: Provjera tokena pri autorizaciji korisnika

```
const config = require('./config/jwtConfig');
const usersService = require('./services/usersService');
const authorizationService = require('./services/authorizationService');
const authorizationSchema = require('./schemas/static/authorizationSchema');
const ajv = require('./validation');

class Authorization {
    static async authorize(req, res, next) {
        const authHeader = req.headers.authorization;
        if (!authHeader) {
            return res.status(401).json({message: 'No authorization header'});
        }

        const token = authHeader.split(' ')[1];
        if (!token) {
            return res.status(401).json({ message: 'No token provided' });
        }

        try {
            const user = config.verifyToken(token);
            if (!user) {
                return res.status(401).json({ message: 'Invalid token' });
            }
            ...
```

Nakon toga se iz baze podataka dohvaćaju autorizacije korisnika prema ruti koja se koristi i prema ID-u korisnika. Za to se koristi metoda *getAuthorizationByRoute* iz servisa *authorizationService*. Korisniku se šalju odgovarajuće poruke ako nema prava za korištenje navedene rute ili navedene HTTP metode. Ako je autorizacija uspješna, onda se u zahtjev spremaju detalji o korisniku koji se mogu iz zahtjeva dohvatiti kada je to potrebno i prijelazi se na iduću funkciju na ruti pozivanjem funkcije *next*. Sve od navedenog nalazi se u programskom kodu 8.3 koji je nastavak na programski kod 8.2.

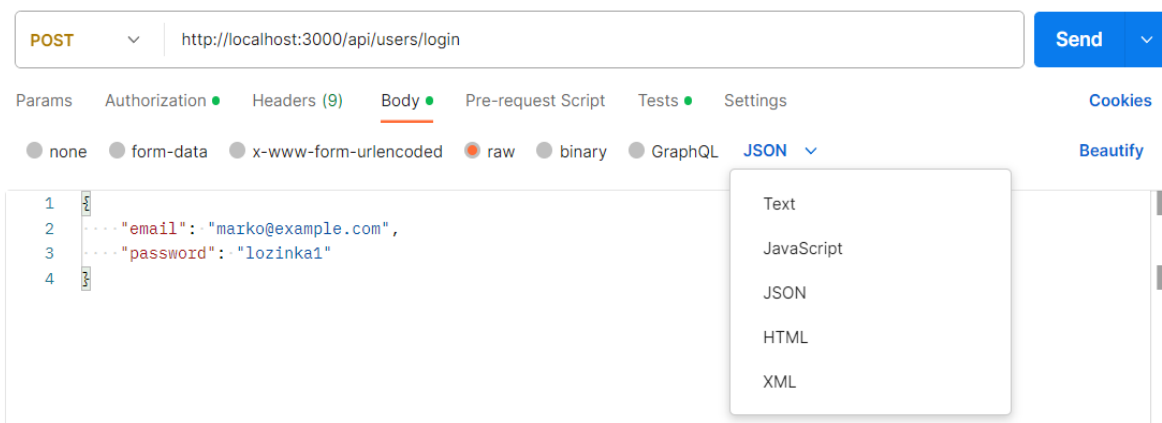
Programski kod 8.3: Provjera autorizacije korisnika

```
let authorization;
const endpoint = authorizationService.extractEndpoint(req.originalUrl);
if (endpoint == '/api/table') {
    authorization = await authorizationService
                          .getAuthorizationByRoute('/api/table', user.id);
    authorization.route = '/api/table';
}
else {
    authorization = await authorizationService
                          .getAuthorizationByRoute(req.originalUrl, user.id);
}

if (authorization == null) {
    return res.status(401).json({
        message: `You are unauthorized to call route ${req.originalUrl}`
    });
}

if (authorization[req.method.toLowerCase()] === 0) {
    return res.status(401).json({
        message: `Missing ${req.method} permission`
    });
}

req.user = user;
next();
```

9. KORIŠTENJE POSTMANA ZA TESTIRANJE

U ovom poglavlju se opisuje korištenje Postmana za testiranje ruta na Node.js serveru uz pomoć kolekcija, mapa, zahtjeva i testnih skripti.

Na samom početku rada s aplikacijom potrebno je prijaviti se kao korisnik. Prijavljuje se korištenjem HTTP metode POST na ruti /api/users/login. U zahtjevu se unutar izbornika *Body* odabire vrsta tijela koja će se slati. Sheme koje se koriste u Node.js serveru najbolje razaznaju brojeve od teksta ako se u Postmanu koristi običan JSON podatak. Zbog toga se koristi opcija *raw* i u dodatnom izborniku sa strane se odabere JSON format kao što je prikazano na slici 9.1.



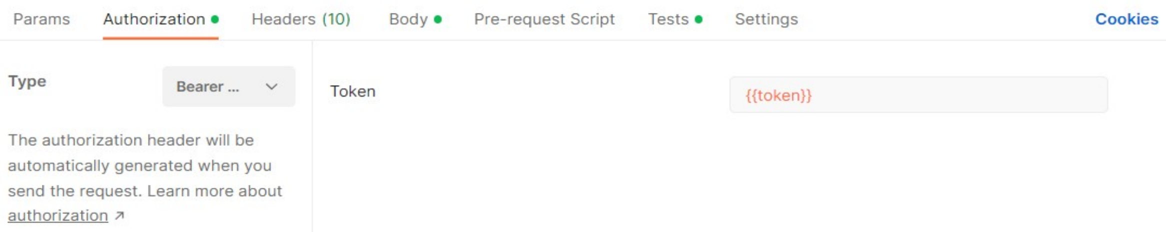
Slika 9.1: Zahtjev za prijavu korisnika

Ako je sve uspješno, server će vratiti JSON podatak koji sadrži token, koje rute korisnik smije pozivati i koje uloge korisnik posjeduje. Za svaki idući zahtjev potrebno je u izbornik *Authorization* upisivati token kako bi autorizacija korisnika bila uspješna. Da se ne bi moralo za svaki zahtjev posebno upisivati token, u zahtjev za prijavu je postavljena testna skripta koja, osim testiranja ispravnosti dobivenih podataka, sprema dobiveni token u globalnu varijablu. Ova testna skripta se upisuje unutar izbornika *Tests*, a programski kod 9.1 prikazuje dio skripte koji sprema token.

Programski kod 9.1: Spremanje tokena u globalnu varijablu

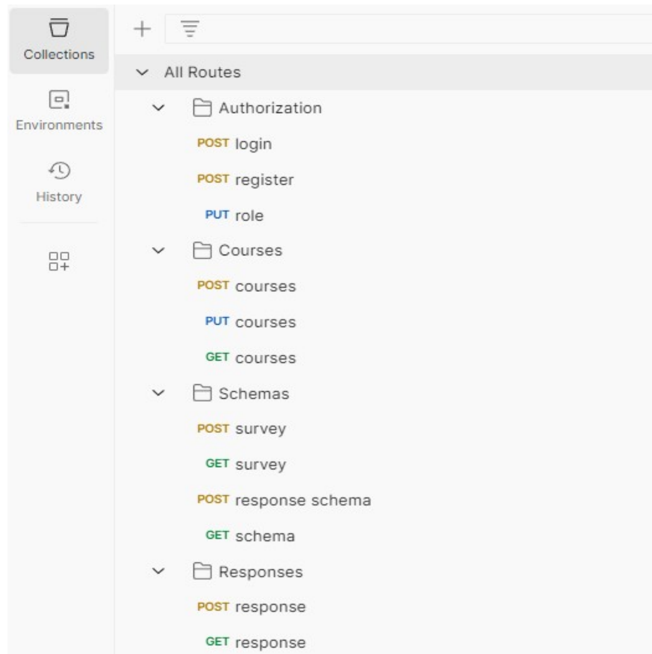
```
const responseJson = pm.response.json();
...
pm.globals.set("token", responseJson.data.token);
```

Na svakom zahtjevu na kojem je potrebno definirati token, u izborniku *Authorization* se odabere *Bearer Token* i u polje se unese ime globalne varijable unutar dvije vitičaste zagrade kao što je prikazano na slici 9.2. Na taj način Postmanu se daje do znanja da na to mjesto mora dohvatiti vrijednost navedene varijable. Ovakve varijable moguće je koristiti na bilo kojem mjestu unutar zahtjeva (npr. ruta ili tijelo zahtjeva).



Slika 9.2: Unos tokena korištenjem globalne varijable

Jedan od glavnih temelja Postmana je rad unutar kolekcija. Svaki zahtjev se mora nalaziti unutar neke kolekcije. U ovom završnom radu korištena je jedna kolekcija koja sadrži sve glavne rute, a podijeljena je na mape. Unutar kolekcija moguće je stvarati mape koje će sadržavati skupinu zahtjeva što je prikazano na slici 9.3. Ovakva struktura ne pomaže samo kod organizacije radnog prostora, već i kod planiranja testiranja više zahtjeva u nizu. Moguće je pokrenuti kolekciju koja će izvršavati sve zahtjeve koji se u njoj nalaze i ispisivati rezultate testova. Također je moguće pokretati i mape ako se žele testirati samo određeni dijelovi aplikacije. Primjer rezultata pokretanja cijele kolekcije nalazi se na slici 9.4.



Slika 9.3: Kolekcija s mapama i zahtjevima

All Routes - Run results
Run Again
Automate Run
New Run
Ran today at 15:22:59 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	1s 910ms	46	58 ms

All Tests Passed (37) Failed (4) Skipped (5) View Summary

POST register
http://localhost:3000/api/users/register
409 Conflict 8 ms 306 B

PASS No server errors
FAIL Status code is 200 | AssertionError: expected 409 to equal 200
FAIL Received data object | AssertionError: expected { message: 'User already exists' } to have property 'data'
FAIL Created user | AssertionError: expected {} to have property 'user'
FAIL Created user roles | AssertionError: expected {} to have property 'role'

PUT role
http://localhost:3000/api/users/role
200 OK 137 ms 438 B

PASS No server errors
SKIP Error handled for not finding user
PASS Received roles

POST courses
http://localhost:3000/api/table/courses
200 OK 72 ms 343 B

PASS No server errors
PASS Received course ID
PASS Received course code

Slika 9.4: Rezultati testova nakon pokretanja kolekcije

Iz primjera je vidljivo da postoje tri moguća rezultata svakog pojedinog testa: PASS, FAIL i SKIP. U prijevodu, testovi mogu biti uspješni i neuspješni, a moguće ih je i preskočiti u slučaju da njihov očekivan rezultat nije relevantan s obzirom na ulazne podatke.

U testnim skriptama se koristi *pm* objekt za spremanje svih podataka vezanih uz odgovore, zahtjeve, okruženja i testove [12]. Postman testovi koriste sintaksu biblioteke *Chai Assertion* koja pruža opcije za optimizaciju čitljivosti testova. Korištenjem ove biblioteke moguće je pisati testove korištenjem riječi iz engleskog jezika koje tvore rečenicu [13].

9.1 Analiza testne skripte

Analizirat će se programski kod 9.2 koji služi kao primjer testne skripte u kojoj se koristi većina metoda iz ostalih testnih skripti. Ova testna skripta nalazi se na zahtjevu za dohvaćanje redaka iz tablice *courses* korištenjem univerzalne rute.

Na početku koda deklariraju se dvije varijable u koje se spremaju podaci u JSON obliku: odgovor servera i tijelo zahtjeva. U prvom testu se provjerava je li došlo do interne greške servera. Očekuje se da dolazni HTTP kod nije jednak 500 koji označava prethodno navedenu grešku.

Drugi i treći test koriste ternarni operator da odluče hoće li se test izvršiti ili preskočiti s obzirom na uvjet koji ispituje postojanje svojstva ID unutar tijela zahtjeva. Testovi su napisani na ovaj način jer se očekivanja dolaznih rezultata mijenjaju s obzirom na to koji su ulazni podaci poslani prema serveru.

Ovi testovi koriste funkciju *expectCourseProperties* koja za objekt *course* očekuje da postoje sva svojstva iz tablice *courses* s ispravnim tipom podatka.

Drugi test će se izvršiti ako nije poslan ID u tijelu zahtjeva. Ovaj test očekuje da će u odgovoru servera dobiti niz objekata gdje svaki objekt predstavlja jedan redak u tablici *courses*. Korištenjem petlje provjerava se ispravnost svakog dobivenog objekta pozivajući funkciju *expectCourseProperties*.

Treći test će se izvršiti samo ako se nalazi svojstvo ID unutar tijela zahtjeva. Ovaj test očekuje da se u odgovoru nalazi samo jedan objekt u polju. Dobiveni objekt se

provjerava funkcijom *expectCourseProperties*. Na kraju testa nalazi se još jedna provjera koja očekuje da je vrijednost ID-a iz tijela zahtjeva jednak ID-u unutar dobivenog objekta kako bi se utvrdilo da je dobiveni objekt onaj koji je i zatražen od servera.

Programski kod 9.2: Testna skripta za dohvaćanje podataka iz tablice courses

```
const response = pm.response.json();
const requestJSON = JSON.parse(pm.request.body);

pm.test("No server errors", function () {
    pm.expect(pm.response.code).to.not.equal(500);
});

(("id" in requestJSON) ? pm.test.skip : pm.test)
("Received an array of courses", () => {
    pm.expect(response).to.be.an("array");
    pm.expect(response.length).to.be.above(0);
    for (let i = 0; i < response.length; i++) {
        expectCourseProperties(response[i]);
    }
});

(("id" in requestJSON) ? pm.test : pm.test.skip)
("Received one course", () => {
    pm.expect(response.length).to.equal(1);
    expectCourseProperties(response[0]);
    pm.expect(requestJSON.id).to.equal(response[0].id);
});

function expectCourseProperties(course) {
    pm.expect(course).to.have.property("id").that.is.a("number");
    pm.expect(course).to.have.property("sifra").that.is.a("string");
    pm.expect(course).to.have.property("name").that.is.a("string");
    pm.expect(course).to.have.property("status").that.is.a("string");
    pm.expect(course).to.have.property("semester").that.is.a("number");
    pm.expect(course).to.have.property("ects").that.is.a("number");
}
```

10. ZAKLJUČAK

Ovaj rad istraživao je integraciju PostgreSQL baze podataka, Ajv.js validacije JSON podataka, i Postman alata za testiranje. Cilj rada bio je stvoriti aplikaciju koja kombinira prednosti relacijskih baza i NoSQL tehnologija za pohranu i obradu raznolikih podataka, uz osiguranje njihove valjanosti i sigurnosti.

Rezultati istraživanja ukazuju na uspješnu implementaciju integracije ovih tehnologija. Postignuta je efikasna i skalabilna aplikacija sposobna za pouzdano pohranjivanje i obradu JSON podataka unutar PostgreSQL baze. Korištenje Ajv.js paketa omogućilo je strogu validaciju podataka prije pohrane, čime je osigurana dosljednost i točnost. Postman je pružio ključni alat za testiranje i analizu funkcionalnosti aplikacije.

Iako su postignuti uspješni rezultati, važno je napomenuti da aplikacija i dalje nosi određena ograničenja. Primjena ovog rješenja može se ograničiti na projekte koji zahtijevaju integraciju različitih tipova podataka, posebice JSON formata. Dodatno, učinkovita uporaba ovog rješenja zahtijeva odgovarajuće stručno znanje i resurse za održavanje i podršku aplikacije.

Ovaj rad pruža temelj za daljnji razvoj i istraživanje u području upravljanja podacima i razvoja aplikacija. Integracija različitih tehnologija otvara mogućnosti za širok spektar primjena u modernom računarstvu i informacijskoj tehnologiji, čime se postiže efikasnije i fleksibilnije rješenje za raznovrsne potrebe skladištenja i obrade podataka.

11. LITERATURA

[1] PostgreSQL Global Development Group. PostgreSQL: About [Online]. 2023. Dostupno na: https://www.postgresql.org/about/. (17.08.2023.)

[2] PostgreSQL Global Development Group. Desktop Deployment – pgAdmin 4 7.5 documentation [Online]. 2023. Dostupno na: https://www.pgadmin.org/docs/pgadmin4/development/desktop_deployment.html. (17.08.2023.)

[3] DBeaver Community. Dbeaver Community | Free Universal Database Tool [Online]. 2023. Dostupno na: https://dbeaver.io/. (17.08.2023.)

[4] Postman Inc. Postman API Platform [Online]. 2023. Dostupno na: https://www.postman.com/. (19.08.2023.)

[5] Postman Inc. Scripting in Postman | Postman Learning Center [Online]. 2023. Dostupno na: https://learning.postman.com/docs/writing-scripts/intro-to-scripts/. (19.08.2023.)

[6] OpenJS Foundation. About | Node.js [Online]. 2023. Dostupno na: https://nodejs.org/en/about. (19.08.2023.)

[7] OpenJS Foundation. Express – Node.js web application framework [Online]. 2023. Dostupno na: https://expressjs.com/. (29.08.2023.)

[8] Gavinwahl. Postgres-json-schema [Online]. 2023. Dostupno na: https://github.com/gavinwahl/postgres-json-schema/blob/master/postgres-json-schema—0.1.1.sql.

[9] IBM. What are NoSQL Databases? | IBM [Online]. 2023. Dostupno na: https://www.ibm.com/topics/nosql-databases. (23.08.2023.)

[10] Evgeny Poberezkin. Getting started | Ajv JSON schema validator [Online]. 2023. Dostupno na: https://ajv.js.org/guide/getting-started.html. (28.08.2023.)

[11] Coda Hale. How To Safely Store A Password [Online]. 2023. Dostupno na: https://codahale.com/how-to-safely-store-a-password/. (07.09.2023.)

[12] Postman Inc. Postman JavaScript Reference | Postman Learning Center [Online]. 2023. Dostupno na: https://learning.postman.com/docs/writing-scripts/script-references/postman-sandbox-api-reference/. (11.09.2023.)

[13] Postman Inc. Test script examples | Public Workspace | Postman API Network [Online]. 2023. Dostupno na: https://www.postman.com/randyroman-qa/workspace/public-workspace/documentation/18313954-c1da2f21-a7b8-469e-89f4-5e182b7e27e6. (11.09.2023.)

12. OZNAKE I KRATICE

API – *Application Programming Interface* (aplikacijsko programsko sučelje)

HTML – *HyperText Markup Language*

JS – *JavaScript*

JSON – *JavaScript Object Notation*

JWT – *JSON Web Token*

SQL – S*tructured Query language* (strukturni upitni jezik)

URL – *Uniform Resource Locator* (usklađeni lokator sadržaja)

VUB – Veleučilište u Bjelovaru

13. SAŽETAK

Naslov: Web servis za komunikaciju s NoSQL modelom u relacijskoj bazi podataka

Ovaj završni rad govori o spajanju NoSQL modela i relacijske baze podataka. Obavlja se validacija JSON podataka korištenjem shema i Ajv.js paketa u Node.js web servisu. Validirani JSON podaci spremaju se u relacijsku bazu podataka PostgreSQL. Koristi se Postman za testiranje rada aplikacije. U Postmanu se koriste kolekcije, mape, zahtjevi i testne skripte. Prilikom registracije i prijave u aplikaciju obavlja se hashiranje lozinki radi zaštite korisničkih podataka. Korisnike se autorizira korištenjem JSON web tokena i tablica koje sadrže prava na akcije i na rute koje korisnik smije koristiti. Korištena je hijerarhija ruter-kontroler-servis za raspodjelu zadataka unutar Node.js web servisa.

Ključne riječi: validacija, shema, token, hash, autorizacija, test.

14. ABSTRACT

Title: Web service for communication with NoSQL model in a relational database

This final thesis is about merging the NoSQL model and the relational database. Validation of JSON data is performed using schemas and the Ajv.js package in the Node.js web service. Validated JSON data is stored in the PostgreSQL relational database. Postman is used to test the operation of the application. Collections, folders, requests and test scripts are used in Postman. When registering and logging into the application, passwords are hashed to protect user data. Users are authorized using JSON web tokens and tables that contain rights to actions and routes that the user is allowed to use. A router-controller-service hierarchy was used to distribute tasks within the Node.js web service.

Keywords: validation, scheme, token, hash, authorization, test.

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.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 26. 9. 2023.	Ivan Juratovac	Juratovac

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

Ivan Juratovac
____
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.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 26. 9. 2023.

Juratovac
____
potpis studenta/ice