IZRADA PLATFORME ZA RAZMJENU STUDENATA

Dragičević, Ivan

SVEUČILIŠTE U SPLIT
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Primijenjeno računarstvo

IVAN DRAGIČEVIĆ

Z A V R Š N I R A D

IZRADA PLATFORME ZA RAZMJENU STUDENATA

Split, rujan 2024.

SVEUČILIŠTE U SPLIT
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Primijenjeno računarstvo

Predmet: Programsko inženjerstvo

Z A V R Š N I R A D

Kandidat: Ivan Dragičević

Naslov rada: Izrada platforme za razmjenu studenata

Mentor: mr. sc. Karmen Klarin, viši predavač

Split, rujan 2024.

Sadržaj

Sažetak

U ovom završnom radu opisuje se izrada web aplikacije koja povezuje studente sa fakultetima i omogućuje studentima odlazak na razmjenu i školovanje na drugom sveučilištu. Nakon registracije, studenti mogu pregledavati popis svih fakulteta na kojima je moguća razmjena i mogućnost prijave ukoliko ispunjavaju uvjete za razmjenu, a imaju uvid u pregled svih svojih prijava i njihovog statusa. S druge strane, fakulteti nakon registracije moraju biti odobrene od strane administratora koji nadgleda sve prijavljene fakultete i ima mogućnost nadgledanja i uređivanja. Poslužiteljska strana napravljena je u razvojnom okviru Node.js, dok je korisničko sučelje izrađeno koristeći razvojni okvir React i jezika za izradu web stranica HTML-a i CSS-a. Za pohranu podataka korištena je nerelacijska baza podataka MongoDB. Kôd je napisan koristeći uređivač izvornog kôda Visual Studio Code.

Ključne riječi: MongoDB , Node.js, React, web aplikacija.

Summary

Creating a platform for student exchange

In this paper the creation of a web application that connects students with faculties and enables students to go on exchange and study at another university is described. After registration, students can view the list of all faculties where exchange is possible and the possibility of applying if they meet the conditions for exchange. Also they have an overview of all their applications and their status. On the other hand, companies after registration must be approved by an administrator who oversees all registered faculties and has the ability to monitor and edit. The server side is built in the Node.js framework, while the user interface is built using the React framework and HTML and CSS web development languages. The non-relational MongoDB database was used for data storage. The code was written using the Visual Studio Code source code editor.

Key words: MongoDB, Node.js, React, web application.

1. Uvod

Cilj ovog rada je izrada web aplikacije koja bi olakšala i omogućila studentima studiranje na drugim fakultetima i upoznavanje različitih sredina te stjecanje novih vještina. U današnje vrijeme globalizacije promjena sredine studiranja i življenja su redovna i česta pojava pa samim tim ovakva mogućnost i pomoć studentima je dobrodošla za njihove karijere i stjecanje nekih novih životnih vrijednosti i prijateljstava.

Aplikacija pruža sveobuhvatan skup funkcionalnosti prilagođenih za tri različite korisničke uloge: administratori, studenti i fakulteti.

Administratori mogu učinkovito upravljati platformom kroz različite funkcionalnosti. Imaju mogućnost sigurne prijave i odjave, pristupa bazi podataka i cjelovitog uvida u sve registrirane korisnike i fakultete. Osim toga, administratori mogu ažurirati svoje profile i upravljati informacijama o fakultetu, uključujući ažuriranje ili brisanje podataka o fakultetu. Također su ovlašteni brisati korisničke račune, prijave, studije i recenzije kada je to potrebno, osiguravajući pravilno održavanje i kontrolu nad bazom korisnika platforme.

Studenti se mogu registrirati na platformu, prijaviti se i odjaviti sa stranice. Aplikacija studentima omogućuje traženje fakulteta koji nude razmjenu studenata i filtriranje tih fakulteta prema određenim kriterijima što omogućava lakšu pretragu i pronalazak razmjene koja odgovara svakom studentu individualno. Studenti mogu upravljati svojim profilima ažuriranjem osobnih podataka prema potrebi. Dodatne značajke uključuju mogućnost pregledavanja detaljnih informacija o fakultetima i pregled uvjeta za razmjenu. Proces prijave na razmjenu omogućuje studentima da prilože potrebne značajke te po potrebi poništе ili ažuriraju prijavu. Studenti imaju uvid u sve svoje prijave i mogućnost uređivanja istih objavljivanja recenzija nakon završetka razmjene i uređivanja istih kako bi i budući studenti imali uvid u potencijalnu razmjenu.

Fakulteti koje koriste platformu mogu se registrirati, prijaviti i odjaviti sa stranice. Mogu upravljati svojim studijima za odrađivanje razmjene i upravljati uvjetima koje je potrebno zadovoljiti. Fakulteti imaju mogućnost uvida u popis studenata koji su se prijavili na njihove studije, pristup osnovnim informacijama i značajkama koje su priložili podnositelji zahtjeva, te

mijenjanje statusa prijave na primjer prihvaćanje ili odbijanje prijava. Fakulteti također mogu ažurirati i uređivati svoje profile kako bi njihove informacije bile aktualne i relevantne.

U ovom radu obrađeno je nekoliko poglavlja koja zajedno čine cjelokupnu sliku razvoja web aplikacije za prijavu studenata na studije. U prvom poglavlju detaljno su opisane *frontend* i *backend* tehnologije korištene u razvoju aplikacije. Ove tehnologije omogućuju brzu, stabilnu i sigurnu aplikaciju s modernim korisničkim sučeljem i efikasnom bazom podataka. Drugo poglavlje fokusira se na aplikacijsko rješenje i dizajn baze podataka, gdje je poseban naglasak stavljen na tablice u bazi podataka i njihovu strukturu. Dizajn baze podataka osigurava jednostavno upravljanje korisničkim podacima i prijavama, te omogućuje skalabilnost za buduće nadogradnje. U sljedećem poglavlju prikazan je proces razvoja funkcionalnosti aplikacije, počevši od prijave i registracije korisnika, zatim administrativnih funkcija poput upravljanja korisnicima, studijima, prijavama i recenzijama. Nadalje, funkcionalnosti su prilagođene različitim korisničkim ulogama, uključujući administratore, fakultete i studente. Na primjer, fakulteti imaju mogućnost dodavanja studija i upravljanja prijavama, dok studenti mogu pregledavati dostupne studije, prijavljivati se i upravljati svojim prijavama, a administrator imaju prava na funkcije za upravljanje korisnicima, kao i na sustav recenzija koji pomaže studentima u odabiru studija. Na kraju, u zaključku su sumirani glavni rezultati postignuti tijekom razvoja aplikacije. Istaknuta je važnost daljnjeg razvoja i potencijalnih nadogradnji kako bi se osigurala kontinuirana podrška korisnicima te unaprijedile funkcionalnosti aplikacije, čime bi se omogućilo lakše upravljanje prijavama i studijima u budućnosti.

2. Tehnologije i alati

U ovom poglavlju pobliže su definirane i opisane tehnologije i alati koji su korišteni prilikom izrade aplikacije. Korisničko sučelje izrađeno je pomoću biblioteke React, jezika za izradu web aplikacije HTML i jezika za uređivanje web aplikacija CSS te biblioteke Redux. Poslužiteljski dio aplikacije izrađen je pomoću razvojnog okvira Node.js, baze podataka MongoDB, te web okvira Express.js.

2.1. React

React je popularna JavaScript biblioteka namijenjena za izgradnju korisničkih sučelja, prvenstveno za *single page aplikacije* (*SPA*). Razvila ga je tvrtka Facebook 2013. godine. React omogućuje programerima da kreiraju ponovo upotrebljive *user interface* (*UI*) komponente, što čini razvojni proces učinkovitijim i olakšava održavanje koda.

Značajke Reacta su:

- JSX - Proširenje JavaScript sintakse, te se preporučuje koristiti u razvoju React aplikacije.
- Komponente - Potrebno je razmišljati o povezanim dijelovima kao komponentama. Taj način razmišljanja pomaže u održavanju kôda.
- Biblioteka React implementira jednosmjerni protok podataka. Flux je obrazac zadužen za održavanje jednosmjernosti podataka.

Prednosti biblioteke React su korištenje DOM-a odnosno JavaScript objekta koji poboljšava izvedbu aplikacije jer je brži od običnog DOM-a, čitljivost zbog korištenja komponenti, te korištenje na strani klijenta i poslužitelja. Također, pokriva samo sloj prikaza aplikacije, pa je stoga potrebno odabrati druge tehnologije kako bi se dobio potpuni set alata za razvoj [1].

2.2. Node.js

Node.js je JavaScript *runtime* okruženje za pokretanje web aplikacija izvan klijentskog preglednika i poznat još kao V8 Engine. V8 Engine dizajniran je s ciljem poboljšanja performansi web aplikacija kompajliranjem JavaScripta u izvorni strojni kôd umjesto da ga tumači, te na taj način čini kod puno bržim. Programeri koriste Node.js kako bi izradili web aplikaciju na strani poslužitelja. Idealan je za aplikacije s velikim brojem podataka jer koristi asinkroni model koji je vođen događajima.

Značajke Node.jsa su:

- Izgrađen je na Google Chrome V8 JavaScript Engineu koji ju čini vrlo brzom u izvršavanju kôda.
- Koristi model jedne niti s petljom događaja što znači da se može izvršavati samo jedan po jedan zadatak. Mehanizam događaja pomaže poslužitelju da odgovori bez blokiranja, te ga čini visoko skalabilnim za razliku od poslužitelja koji za obradu zahtjeva stvaraju ograničene niti.
- Asinkroni po prirodi i vođeni događajima - Poslužitelji koji su izrađeni pomoću Node.jsa nikada ne čekaju podatke iz API-ja, te su svi API-i potpuno ne blokirajući. Kako bi poslužitelj primio podatke, te pratio sve odgovore iz prethodnih API zahtjeva, prati mehanizam vođen događajima.
- Nema međuspremnika - Aplikacije izrađene u Node.jsu nikada ne spremaju podatke u međuspremnik.

Node.js se koristi prilikom izrade aplikacija koje obrađuju veću količinu podataka i kod izrade aplikacija koje koriste JSON bazirane API-je i jednostranične web aplikacije [2].

2.3. Express.js

Express.js besplatni je web aplikacijski okvir napravljen za Node.js koji pomaže kod upravljanja rutama i poslužiteljem, te zahtijeva samo JavaScript programski jezik. Bez njega,

developeri bi trebali napisati svoj kôd za izgradnju ruta, dok Express.js nudi minimalizam i skalabilnost [3]. Značajke Express.jsa su:

- Brži razvoj poslužiteljske strane aplikacije - omogućava korištenje mnogih funkcionalnosti u obliku funkcija koje mogu biti iskorištene bilo gdje u programu, te na taj način štede vrijeme.
- Zahtjeva samo JavaScript programski jezik
- Posrednički softver dio je programa koji ima pristup bazi podataka, ostalim posredničkim softverima, te zahtjevima klijenta.
- Rutiranje - odnosi se na način kako će web aplikacija odgovoriti na neki zahtjev preko određenog URL puta, te HTTP zahtjeva.
- Podrška za *templating* - omogućavaju developerima izgradnju i razvoj HTML predložaka za serversku stranu aplikacije.
- Otklanjanje pogrešaka ključno je za razvoj ispravne i korisne web aplikacije, te Express.js to olakšava tako što ima ugrađene mehanizme koji točno ukazuju na mjesto gdje se pogreška nalazi.

2.4. MongoDB

MongoDB je nerelacijska baza podataka koja koristi zbirke i dokumente umjesto tablica i redaka koje se koriste kôd tradicionalnih relacijskih baza podataka. Za povezivanje baze podataka i serverske strane koristi se biblioteka *Mongoose*. *Mongoose* je *objektno relacijsko mapiranje (eng. Object Relational Mapping/Mapper, ORM)* dokumenata objekata baziran na *Node.js*, te također upravlja odnosima između podataka, omogućava provjeru valjanosti sheme, te se koristi i za prevođenje između objekata u kôdu i reprezentacije tih istih objekata u bazi podataka *MongoDB* . Dokumenti su sastavljeni od parova ključ - vrijednost, te su oni osnovna jedinica u bazi podataka *MongoDB*, a zbirke su sastavljene od skupa dokumenata i funkcija. Zbog spremanja podataka u dokumente, *MongoDB* je fleksibilniji i prilagodljiviji situacijama i zahtjevima koji su potrebni klijentima i korisnicima [4].

2.5. HTML

HTML *(*engl. *HyperText Markup Language)* je standardni jezik za izradu i oblikovanje web stranica. Predstavlja osnovu svakog web dokumenta, definirajući strukturu i sadržaj stranice putem oznaka. HTML nije programski jezik, već označni jezik koji omogućuje prikazivanje teksta, slika, veza, obrazaca i drugih elemenata na webu [5].

2.6. CSS

CSS *(*engl. *Cascading Style Sheets)* je stilski jezik koji se koristi za opisivanje izgleda i oblikovanja HTML dokumenata. Pomoću CSS-a, programeri mogu odvojiti sadržaj web stranice od njenog vizualnog izgleda, što omogućuje lakšu promjenu stila stranice bez potrebe za izmjenom osnovne HTML structure [6].Osnovne karakteristike CSS-a:

- Selektori u CSS-u određuju koji HTML elementi će biti stilizirani. Postoji više vrsta selektora: *Element* selektori, *ID* selektori, Klasa selektori
- CSS definira stilove pomoću svojstava i odgovarajućih vrijednosti
- Cascading u CSS-u znači da stilovi mogu biti nasljeđeni ili nadjačani prema specifičnosti selektora i redoslijedu kojim su definirani. Ako dva pravila ciljaju isti element, ono koje je specifičnije ili koje dolazi kasnije u kôdu ima prednost.
- Svaki HTML element se može smatrati kao pravokutna kutija. CSS omogućuje definiranje različitih aspekata te kutije, uključujući obrub, podlogu i sadržaj.
- CSS omogućuje stvaranje responzivnih web stranica koje se prilagođavaju različitim veličinama ekrana koristeći fleksibilne rešetke, rasporede i medijske upite

CSS je ključan za moderni web dizajn, omogućujući stvaranje estetski privlačnih i funkcionalnih web stranica.

3. Struktura razvojnog rješenja i baze podataka

3.1. Aplikacijsko rješenje

Aplikacijsko rješenje razvijeno je korištenjem CRUD (*create-read-update-delete)*. Operacija CRUD je akronim koji označava osnovne operacije nad podacima u bazi podataka:

- **Create (Stvaranje)**: Dodavanje novih zapisa u bazu podataka.
- **Read (Čitanje)**: Dohvaćanje postojećih podataka iz baze podataka.
- **Update (Ažuriranje)**: Promjena postojećih podataka u bazi podataka.
- **Delete (Brisanje)**: Uklanjanje podataka iz baze podataka.

Na primjer, metoda `getUsers` koristi se za dohvaćanje svih korisnika iz baze, dok metoda `postUser` koristi se za stvaranje korisnika i spremanje njegovih podataka u bazi. Metoda `putUser` služi za izmjenu i spremanje podataka korisnika u bazu, a `deleteUser` metoda služi za brisanje korisničkih podataka u bazi podataka.

3.2. Baza podataka

Baza podataka korištena u ovom projektu naziva se `Završni.db`. To je nerelacijska *MongoDB* baza podataka koja se sastoji od tablica koje odgovaraju strukturi modela korištenih u aplikaciji (Slika 1).

Slika 1: Dijagram Zavrsni.db baze podataka

3.2.1. Tablice

Atributi i tipovi atributa tablice `users` su: `id` integer(32), `email` string(256), `password` string(256), `role` string(256), `student_id` object, `university_id` object.

Atributi i tipovi atributa tablice `Student` su: `id` integer(32), `name` string(256), `surname` string(256), `email` string(256), `role` string(256), `registration_check` boolean, `score` number(8).

Atributi i tipovi atributa tablice `University` su: `id` integer(32), `name` string(256), `location` string(256), `email` string(256), `password` string(256), `role` string(256), `student_id` object, `university_id` object.

Atributi i tipovi atributa tablice `Course` su: `id` integer(32), `name` string(256), `university_id` object.

Atributi i tipovi atributa tablice `Prijava` su: `id` integer(32), `name` string(256), `student_id` object, `course_id` object, `description` string(256), `check` boolean, `start` boolean, `end` boolean.

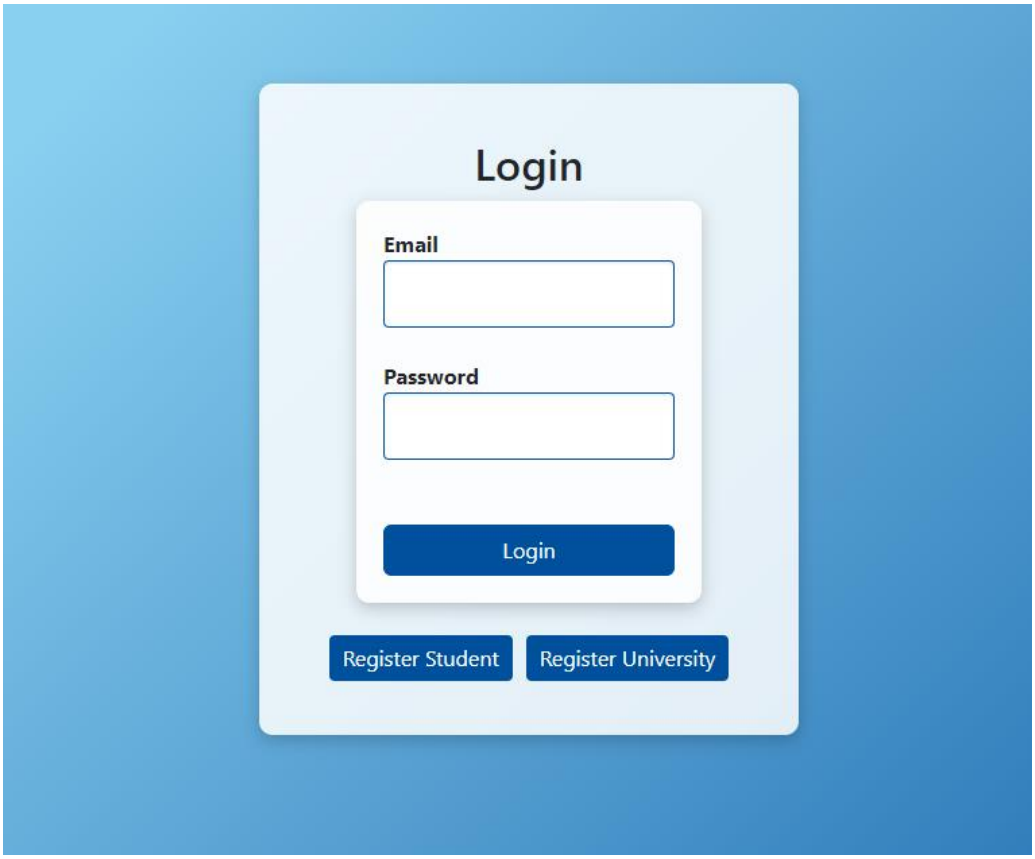
Atributi i tipovi atributa tablice `Location` su: `id` integer(32), `name` string(256), `description` string (256), `image` string(256), `university_ids` object.

Atributi i tipovi atributa tablice `Recension` su: `id` integer(32), `name` string(256), `description` string(256), `student_id` object, `university_id` object, `course_id` object, `prijava_id` object.

4. Funkcionalnosti

4.1. Prijava i registracija

Početna strana aplikacije je prikaz logina i registracije (Slika 2).



Slika 2: Prikaz prijave i registracije

```
router.post('/login',jsonParser,(req, res)=>{
    User.find({email: req.body.email}, function (error, users) {
        if (error || users.length === 0) {
            return res.json({message: "email"});
        }
        if (req.body.email !== users[0].email) {
            return res.json({message: "email"})
        }
        if ( users[0]._id == null){
            return res.status(400).json({ message: error.message });
        }
        bcrypt.compare(req.body.password, users[0].password, function(err, result) {
            if (result) {
                const token = signJwt(users[0]._id);
```

```
                const myuser = {name:users[0].name, role:users[0].role}
                return res.json({accessToken: token, user_id: users[0].id, user_role:
users[0].role,  user_name:users[0].name});

            }
            else {
                return res.json({message: "password"})
            }
          });
    })
});
```

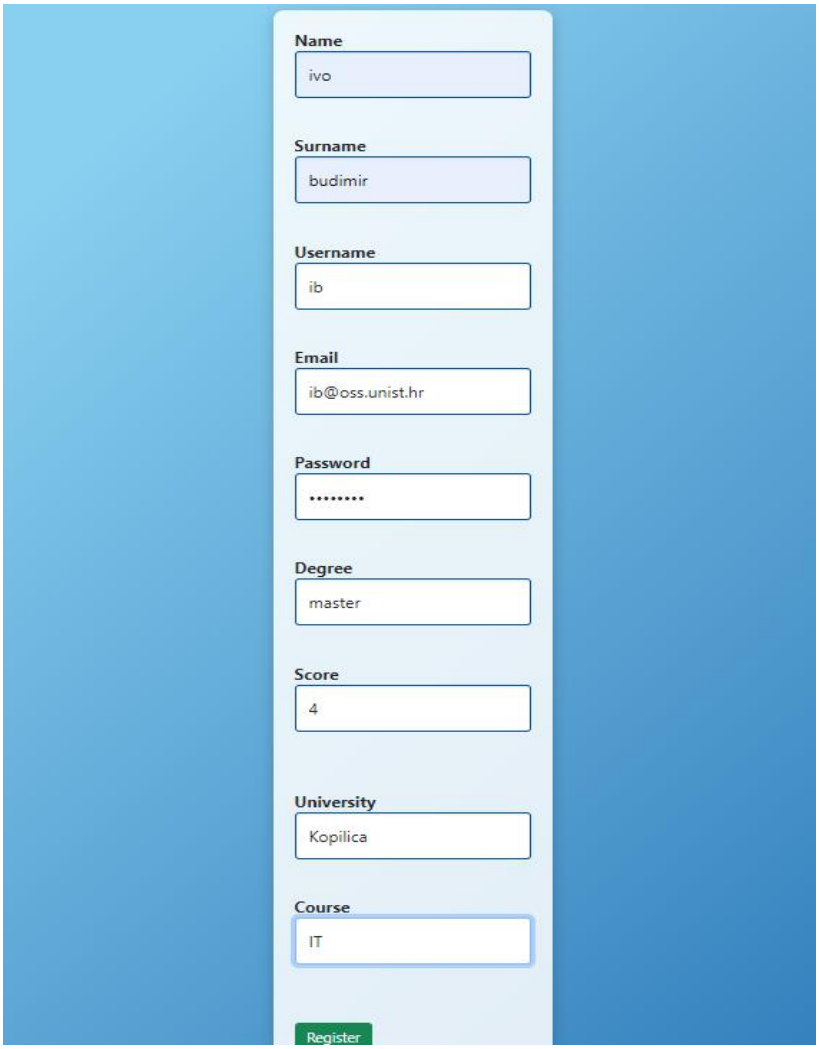
Ispis 1: Metoda za login korisnika

Ovaj kôd (Ispis 1) predstavlja funkciju koja obrađuje zahtjev za prijavu korisnika *bcrypt* bibliotekom za provjeru lozinki. Kada korisnik unese svoje podatke za prijavu, funkcija prvo pretražuje bazu podataka kako bi pronašla korisnika prema unesenoj email adresi. Ako se korisnik ne pronađe ili dođe do pogreške, funkcija vraća odgovor s porukom. U suprotnom, provjerava se podudarnost unesene lozinke s pohranjenom *hashiranom* lozinkom u bazi. Ako je lozinka ispravna, generira se *JSON Web Token* (JWT) za autentifikaciju i korisniku se prijavljuje u sustav. Ovaj postupak osigurava da samo ovlašteni korisnici mogu pristupiti sustavu koristeći ispravne podatke za prijavu. U slučaju da se novi korisnik želi registrirati ima mogućnost za registraciju klikom na dugme `Register Student` ako se radi o studentu ili `Register University` ako se radi o fakultetu.

```
router.post('/user/register/student', jsonParser, (req, res) => {
    console.log("heey");
    console.log("heey");
    console.log(req.body);

    Student.find({email: req.body.email}, function (error, users) {
        if (error || users.length > 0) {
            return res.json({message: "email"});
        }

        const saltRounds = 10; // definiranje salt vrijednosti
        bcrypt.genSalt(saltRounds, function(err, salt) {
            bcrypt.hash(req.body.password, salt, function(err, hash) {
                let user = new Student({ name: req.body.name ,surname: req.body.surname,
username: req.body.username,email: req.body.email, password: hash, role: "student",degree:
req.body.degree, score: req.body.score, course_id:req.body.course_id});
                user.save();
                console.log(user)
                let user2 = new User({ email: user.email, password:user.password, role:
user.role, student_id:user._id});
                user2.save();
                console.log(user2)
                return res.json(user2);
            });
          });
```

Ispis 2:Metoda za registriranje studenta

Ovaj kôd (Ispis 2) definira funkciju koja obrađuje zahtjev za registraciju novog studenta (Slika 3). Prilikom registracije, funkcija najprije provjerava postoji li već korisnik s unesenom email adresom u bazi podataka. Ako takav korisnik već postoji ili dođe do pogreške, nije moguće napraviti novog studenta, a Ako email nije zauzet, koristi se *bcrypt* za *hashiranje* lozinke s definiranim brojem "*salt*" vrijednosti. Nakon toga, stvara se novi objekt Student s podacima koje je korisnik unio, uključujući *hashiranu* lozinku, te se isti sprema u bazu podataka. Paralelno s time, kreira se i objekt *User*, koji pohranjuje osnovne korisničke podatke zajedno s ID-om studenta, kako bi se omogućila integracija s ostalim dijelovima sustava. Na kraju, novi korisnik se sprema u bazu podataka, a funkcija vraća JSON odgovor s podacima o novo registriranom korisniku.



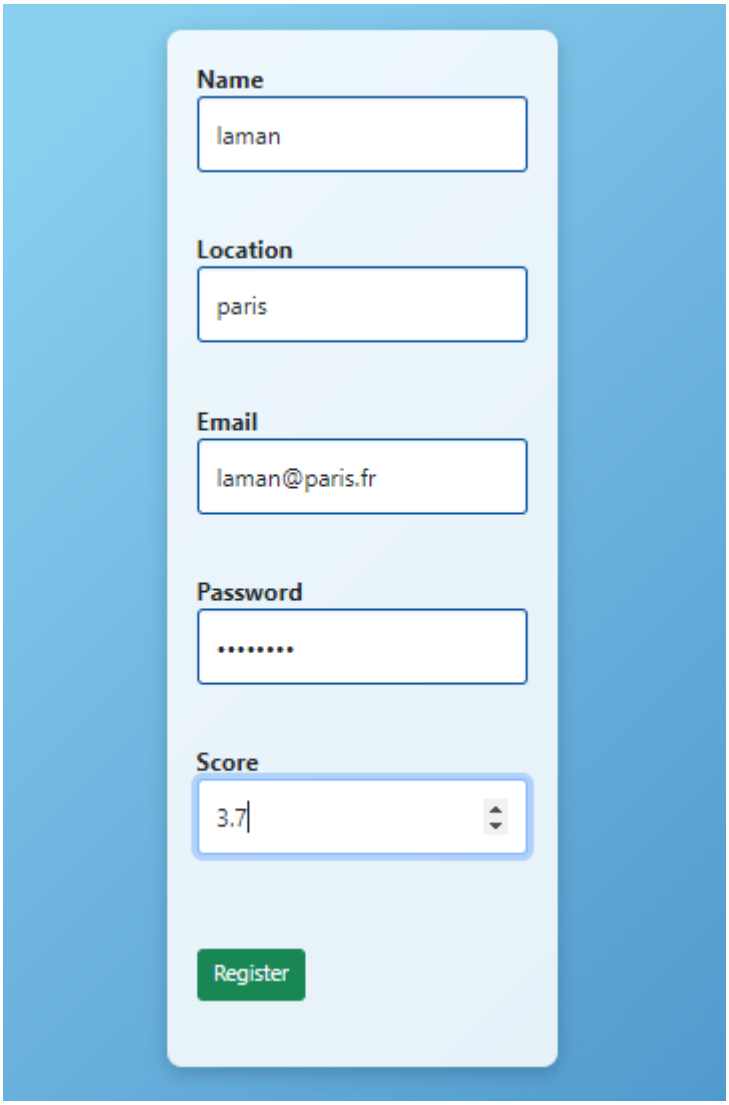
Slika 3: Registracija studenta

```
router.post('/user/register/university', jsonParser, (req, res) => {
    console.log("heey");
    console.log("heey");
    console.log(req.body);

    University.find({email: req.body.email}, function (error, users) {
        if (error || users.length > 0) {
            return res.json({message: "email"});
        }

        const saltRounds = 10; // definiranje salt vrijednosti
        bcrypt.genSalt(saltRounds, function(err, salt) {
            bcrypt.hash(req.body.password, salt, function(err, hash) {
                let user = new University({ name: req.body.name, score:req.body.score , email:
req.body.email,location:req.body.location, password: hash, role: "university"});
                user.save();
                console.log(hash);
                return res.json(user);
            });
          });
```

```
    })
});
```

Ispis 3: Metoda za registraciju fakulteta

Ako se radi o registraciji fakulteta (Slika 4) onda se unesu podaci za registraciju fakulteta, funkcija najprije provjerava postoji li već sveučilište s unesenom email adresom u bazi podataka. Ako takav korisnik već postoji ili dođe do pogreške, javlja se da je nemoguće napraviti novi fakultet. Ukoliko je email adresa slobodna, *bcrypt* se koristi za generiranje "*salt*" vrijednosti i *hashiranje* lozinke koju je korisnik unio (Ispis 3). Zatim se kreira novi objekt "*University*" s unesenim podacima, uključujući *hashiranu* lozinku, te se isti pohranjuje u bazu podataka. Na kraju, funkcija vraća odgovor s podacima o novo registriranom fakultetu. Ovaj postupak osigurava da se svi osjetljivi podaci, poput lozinki, sigurno pohranjuju u bazi podataka.

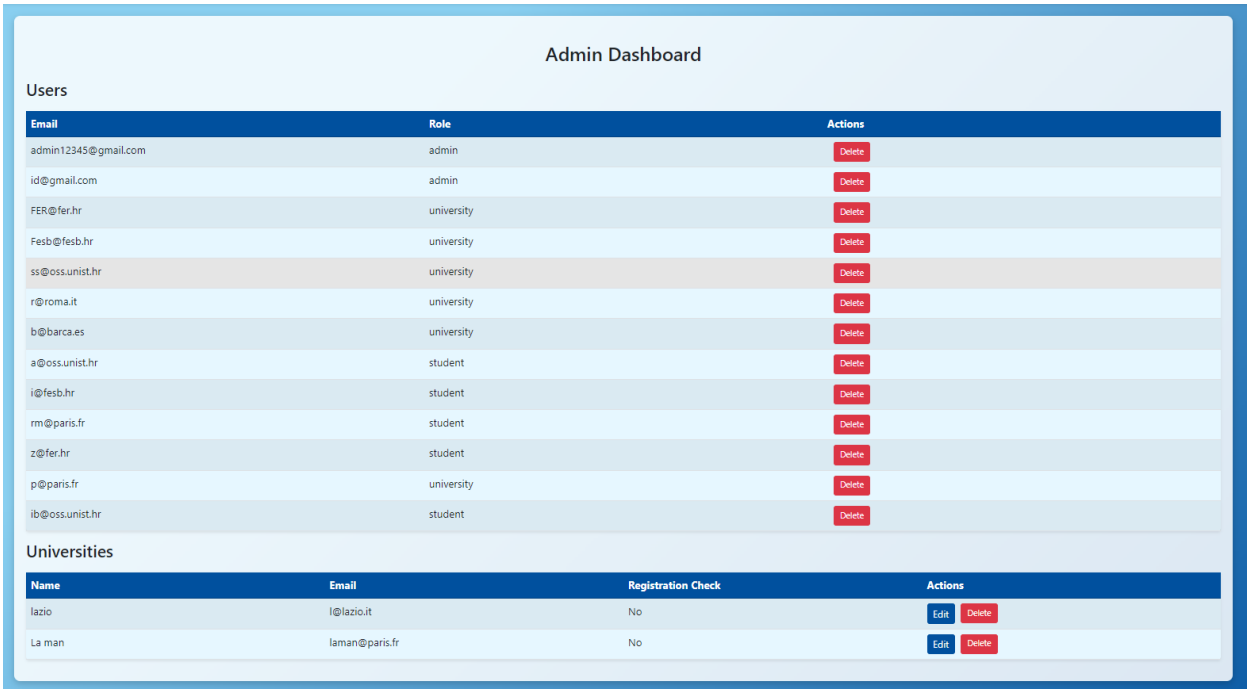


Slika 4: Registracija fakulteta

4.2. Funkcionalnosti: Administrator

4.2.1. Upravljanje korisnicima

Administrator stranice ima uvid u sve registrirane potvrđene i ne potvrđene korisnike stranice (Slika 5).



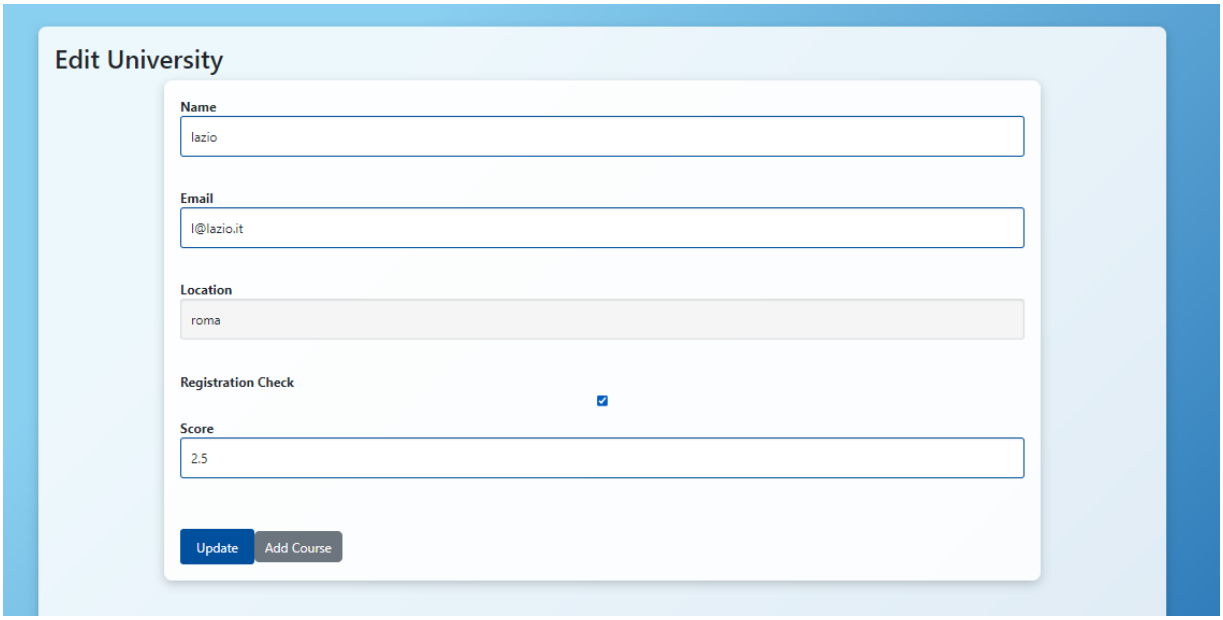
Slika 5: Prikaz administratorskog sučelja

```
useEffect(() => {
        fetch("http://localhost:3000/users", {
            method: "GET",
            headers: { "Content-type": "application/json;charset=UTF-8" }
        })
        .then((resp) => resp.json())
        .then((data) => setUsers(data))
        .catch((err) => console.log(err));

        fetch("http://localhost:3000/universities", {
            method: "GET",
            headers: { "Content-type": "application/json;charset=UTF-8" }
        })
        .then((resp) => resp.json())
        .then((data) => {
            const filteredUniversities = data.filter(university =>
!university.registration_check);
            setUniversities(filteredUniversities);
        })
        .catch((err) => console.log(err));
    }, []);

    const handleDeleteUser = (id) => {
        fetch(`http://localhost:3000/${id}`, {
            method: "DELETE",
            headers: { "Content-type": "application/json;charset=UTF-8" }
        })
        .then((resp) => resp.json())
        .then(() => setUsers(users.filter(user => user._id !== id)))
        .catch((err) => console.log(err));
    };

    const handleDeleteUniversity = (id) => {
```

```
        fetch(`http://localhost:3000/university/${id}`, {
            method: "DELETE",
            headers: { "Content-type": "application/json;charset=UTF-8" }
        })
        .then((resp) => resp.json())
        .then(() => setUniversities(universities.filter(university => university._id !== id)))
        .catch((err) => console.log(err));
    };
```

Ispis 4: Komponenta administratorskog sučelja

Ova metoda (Ispis 4) `get` i putanjom "/users" omogućuje dohvaćanje i prikaz popisa korisnika koji su registrirani i potvrđeni, a metoda `get` i putanjom "/universities" vraća sve fakultete koji se onda filtriraju i ispisuju oni koji su ne potvrđeni. Zbog toga, tablica `Users` ima samo dugme `delete`, a tablica Universities ima i dugme `edit` jer administrator ima mogućnost editiranja registracije fakulteta i protvrđivanja pritiskom na dugme `Registration check` i potvrđivanja promjene (Slika 6).

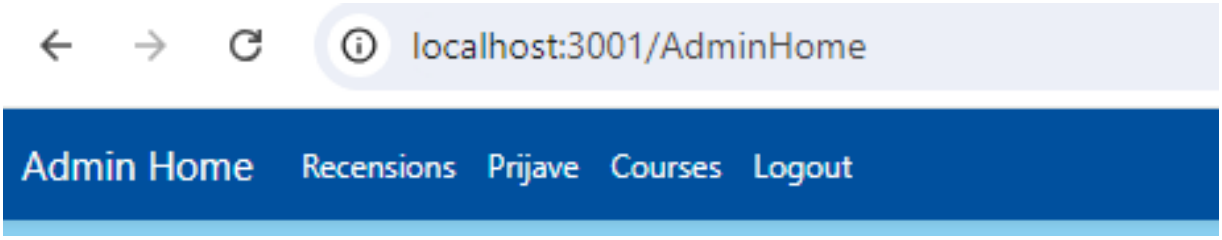


Slika 6: Forma za editiranje fakulteta

Unutar administrativnog sučelja implementirana je navigacijska traka služi kao glavni izbornik za administratora, omogućujući mu brz i jednostavan pristup različitim dijelovima sustava, a sastoji se od (Slika 7):

- **Recensions**: Poveznica koja vodi na stranicu za pregled i upravljanje recenzijama.

- **Prijave**: Poveznica koja vodi na stranicu za upravljanje prijavama, poput prijava studenata ili sveučilišta.
- **Courses**: Poveznica na stranicu za pregled i upravljanje kolegijima.
- **Logout**: Poveznica za odjavu sa stranice.



Slika 7: Navigacijska traka za administratora

4.2.2. Upravljanje studijima

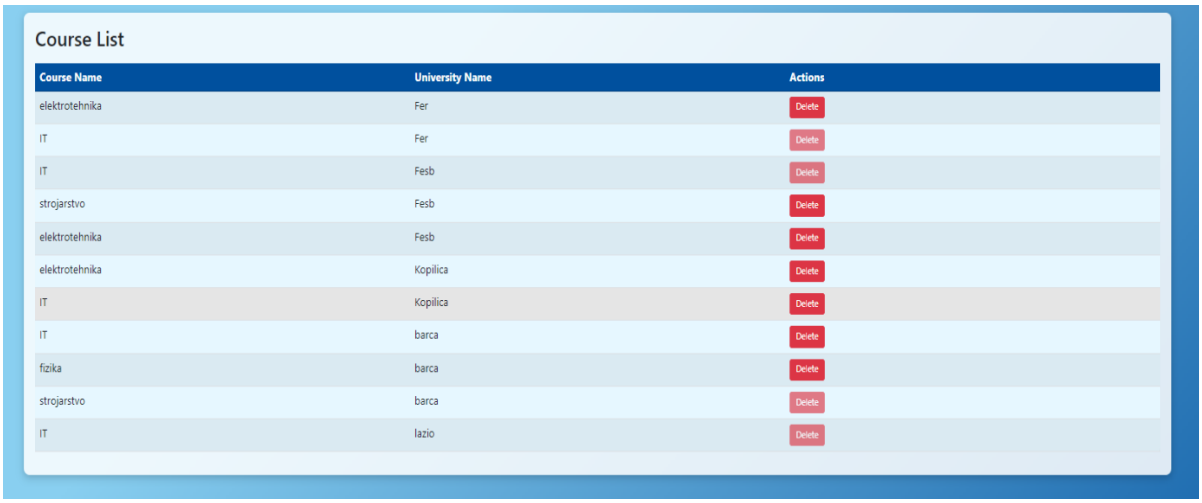
Administrator ima pravo pregleda svih studija na svim fakultetima, a ako je potrebno ima mogućnost brisanja studija.

```
useEffect(() => {
        // Fetch courses
        fetch('http://localhost:3000/api/courses', {
            method: 'GET',
            headers: { 'Content-Type': 'application/json;charset=UTF-8' }
        })
        .then((resp) => resp.json())
        .then((data) => setCourses(data))
        .catch((err) => console.log(err));

        // Fetch prijave
        fetch('http://localhost:3000/api/prijave', {
            method: 'GET',
            headers: { 'Content-Type': 'application/json;charset=UTF-8' }
        })
        .then((resp) => resp.json())
        .then((data) => setPrijave(data))
        .catch((err) => console.log(err));
    }, []);

    const handleDeleteCourse = (id) => {
        fetch(`http://localhost:3000/api/courses/${id}`, {
            method: 'DELETE',
            headers: { 'Content-Type': 'application/json;charset=UTF-8' }
        })
        .then((resp) => resp.json())
        .then(() => setCourses(courses.filter(course => course._id !== id)))
        .catch((err) => console.log(err));
    };
```

```
    const isCourseDeletable = (courseId) => {
        return !prijave.some(prijava => prijava.course_id._id === courseId);
    };
```

Ispis 5: Komponenta za prikaz studija

Ovaj kôd (Ispis 5) omogućuje pregled i upravljanje popisom studija unutar aplikacije. Komponenta koristi Reactov `useState` i `useEffect` za dohvaćanje tih podataka s poslužitelja prilikom prvog učitavanja komponente. Unutar `useEffect` funkcije, izvršavaju se dvije *get* metode. Prva metoda dohvaća podatke o kolegijima i pohranjuje u stanje `courses` pomoću funkcije `setCourses`. Druga metoda dohvaća podatke o prijavama i pohranjuje ih u stanje `prijave`. Funkcija `isCourseDeletable` koristi se za provjeru može li se određeni kolegij obrisati. Ako postoji prijava koja se odnosi na taj kolegij, funkcija će vratiti *false*, čime se sprječava brisanje kolegija koji je već povezan s prijavom. Ako kolegij nije povezan s nijednom prijavom, funkcija vraća *true*, što omogućuje funkciju `handleDeleteCourse` koja omogućava brisanje kolegija (Slika 8).



Slika 8: Prikaz svih studija

4.2.3. Upravljanje prijavama

Administrator ima prava pregledavanja svih prijava za svaki studija. Također, ima mogućnost uređivanja ili brisanja prijava.

```
const AdminPrijave = () => {
    const [prijave, setPrijave] = useState([]);
```

```
useEffect(() => {
    fetchPrijave();
}, []);

const fetchPrijave = () => {
    fetch("http://localhost:3000/api/prijave", {
        method: "GET",
        headers: { "Content-type": "application/json;charset=UTF-8" }
    })
    .then((resp) => resp.json())
    .then((data) => {
        setPrijave(data);
        fetchUniversityNames(data);
    })
    .catch((err) => console.log(err));
};

const fetchUniversityNames = (prijaveData) => {
    prijaveData.forEach(prijava => {
        const courseId = prijava.course_id._id; // Assuming course_id is an object with _id
        fetch(`http://localhost:3000/api/courses/${courseId}`, {
            method: "GET",
            headers: { "Content-type": "application/json;charset=UTF-8" }
        })
        .then((resp) => resp.json())
        .then((courseData) => {
            // Update prijave state with university name
            setPrijave(prevPrijave => (
                prevPrijave.map(p => {
                    if (p._id === prijava._id) {
                        return {
                            ...p,
                            universityName: courseData.university_id.name
                        };
                    }
                    return p;
                })
            ));
        })
        .catch((err) => console.log(err));
    });
};

const handleDeletePrijava = (id) => {
    fetch(`http://localhost:3000/api/prijave/${id}`, {
        method: "DELETE",
        headers: { "Content-type": "application/json;charset=UTF-8" }
    })
    .then(() => {
        setPrijave(prijave.filter(prijava => prijava._id !== id));
    })
    .catch((err) => console.log(err));
};
```

Ispis 6: Komponenta za prikaz svih prijava

Komponenta `AdminPrijave` predstavlja sučelje za upravljanje prijavama i koristi *`useState`* i *`useEffect`* za dohvaćanje i prikaz podataka o prijavama, kao i za njihovo ažuriranje i brisanje (Ispis 6). Na početku, u funkciji `fetchPrijave`, komponenta šalje *get*

zahtjev kako bi dohvatila sve prijave. Kada se podaci uspješno dobiju i pretvore u JSON format, pohranjuju se u stanje prijave pomoću funkcije `setPrijave`. Nakon toga, funkcija `fetchUniversityNames` se poziva kako bi se za svaku prijavu dohvatili podaci o sveučilištu povezani s prijavom. Funkcija `fetchUniversityNames` prolazi kroz sve prijave i za svaku prijavu dohvaća informacije o kolegiju na koji se prijava odnosi. Kada se podaci o kolegiju uspješno dobiju, ažurira se stanje prijave dodajući naziv sveučilišta za svaku prijavu. Ovo ažuriranje se vrši pomoću funkcije `setPrijave`, koja koristi prethodno stanje za generiranje novog popisa prijava s dodanim nazivom sveučilišta. Za brisanje prijava koristi se funkcija `handleDeletePrijava`, koja šalje *delete* zahtjev za brisanje prijave. Nakon uspješnog brisanja, stanje prijave se ažurira kako bi se uklonila izbrisana prijava iz prikazanog popisa, čime se osigurava da korisnik vidi ažurirani popis bez potrebe za ručnim osvježavanjem stranice. Komponenta na ovaj način omogućuje administratoru da pregleda sve prijave, identificira naziv sveučilišta za svaku prijavu, te upravlja prijavama prema potrebi. Ovo omogućuje učinkovitije upravljanje prijavama i osigurava da su podaci uvijek ažurni i relevantni (Slika 9).

Slika 9: Prikaz svih prijava

4.2.4. Upravljanje recenzijama

Administrator ima pravo pregledavanja i mogućnost brisanja svih recenzija.

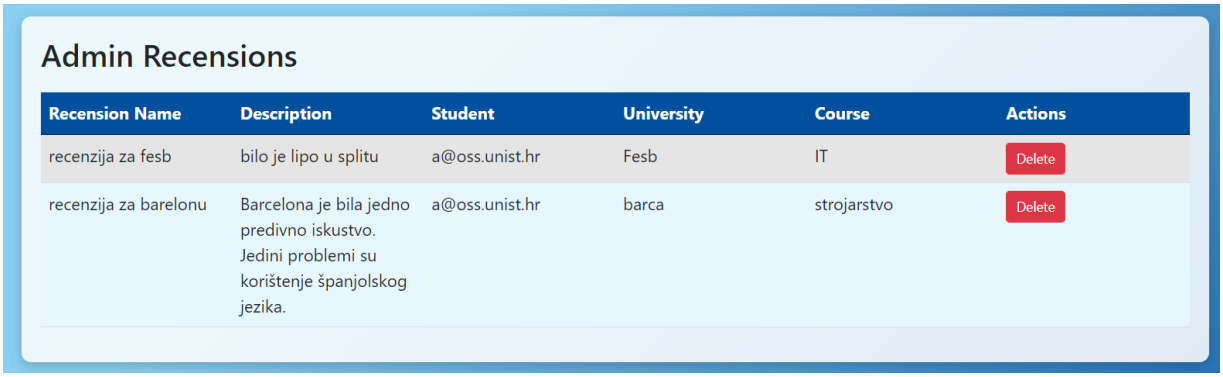
```
const AdminRecension = () => {
    const [recensions, setRecensions] = useState([]);

    useEffect(() => {
        fetchRecensions();
    }, []);
```

```
const fetchRecensions = () => {
    fetch("http://localhost:3000/api/recensions", {
        method: "GET",
        headers: { "Content-type": "application/json;charset=UTF-8" }
    })
    .then((resp) => resp.json())
    .then((data) => setRecensions(data))
    .catch((err) => console.log(err));
};

const handleDeleteRecension = (id) => {
    fetch(`http://localhost:3000/api/recensions/${id}`, {
        method: "DELETE",
        headers: { "Content-type": "application/json;charset=UTF-8" }
    })
    .then((resp) => resp.json())
    .then(() => {
        // Remove the deleted recension from state
        setRecensions(recensions.filter(recension => recension._id !== id));
    })
    .catch((err) => console.log(err));
};
```

Ispis 7: Komponenta za prikaz svih recenzija

Komponenta `AdminRecension` služi za upravljanje recenzijama unutar administrativnog sučelja aplikacije (Ispis 7). Ova komponenta koristi Reactove *useState* i *useEffect hookove* za dohvaćanje, prikazivanje i brisanje recenzija. U početku, komponenta koristi *useEffect* kako bi pozvala funkciju `fetchRecensions` prilikom prvog renderiranja (proces pretvaranja modela u dvodimenzionalnu sliku) komponente. Ova funkcija šalje *get* zahtjev kako bi prikupila sve dostupne recenzije iz baze podataka. Nakon što se podaci uspješno dobiju i obrade u JSON formatu, rezultati se pohranjuju u stanje `recensions` pomoću funkcije `setRecensions`. Ako tijekom dohvaćanja podataka dođe do greške, ona se ispisuje u konzolu radi lakšeg praćenja problema. Za brisanje recenzija koristi se funkcija `handleDeleteRecension`, koja šalje *delete* zahtjev za brisanje određene recenzije. Nakon uspješnog brisanja recenzije, komponenta ažurira stanje `recensions` kako bi izbrisala recenziju iz prikazane liste. Ovo se postiže filtriranjem stanja `recensions` kako bi se uklonile recenzije čiji indetifikator odgovara indetifikatoru izbrisane recenzije. Ova komponenta omogućuje administrator pregledavanje svih recenzijai upravljanje njima kroz funkcionalnost brisanja, čime se osigurava relevantnost i ažuriranost podataka. Također, omogućuje jednostavno i učinkovito upravljanje recenzijama unutar administrativnog sučelja aplikacije (Slika 10).

Slika 10: Prikaz svih recenzija



Slika 10: Prikaz svih recenzija

4.3. Funkcionalnosti: Fakultet

4.3.1. Uređivanje i pregledavanje osobnih podataka

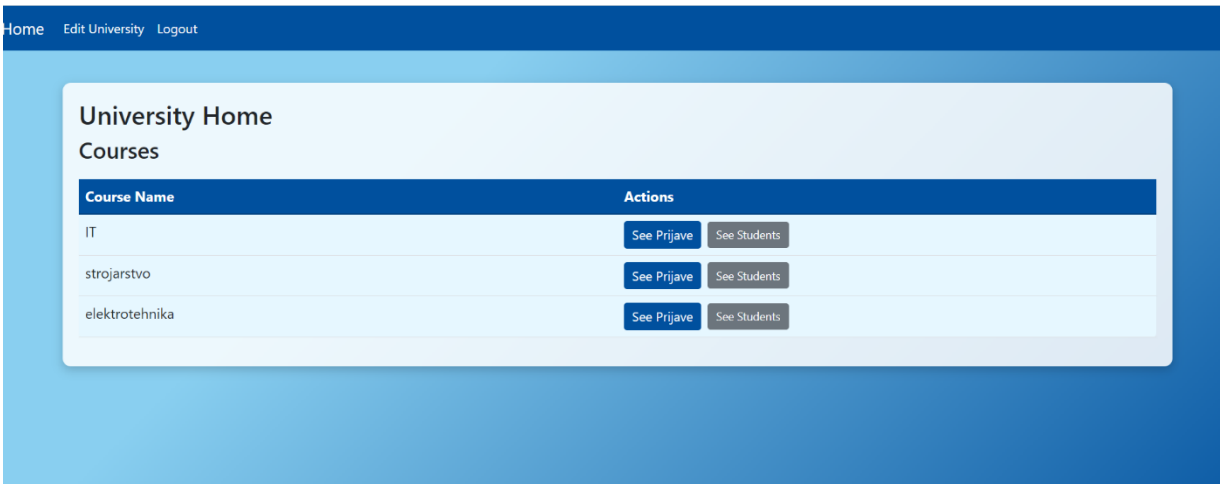
Fakultet ima prikaz svih svojih studija te mogućnost pregledavanja svih prijava i svih studenata koji su se prijavili ili obavili razmjenu za taj studij.

```
useEffect(() => {
        if (userId) {
            fetch(`http://localhost:3000/user/${userId}`, {
                method: 'GET',
                headers: { 'Content-type': 'application/json;charset=UTF-8' }
            })
            .then((resp) => resp.json())
            .then((userData) => {
                if (userData.user.role === 'university') {
                    setUniversityId(userData.university._id);
                    setRole(userData.user.role);
                }
            })
            .catch((err) => console.log(err));
        }
    }, [userId]);

    useEffect(() => {
        if (universityId) {
            fetch('http://localhost:3000/api/courses', {
                method: 'GET',
                headers: { 'Content-type': 'application/json;charset=UTF-8' }
            })
            .then((resp) => resp.json())
            .then((data) => {
                const filteredCourses = data.filter(course => course.university_id._id ===
universityId);
                setCourses(filteredCourses);
            })
            .catch((err) => console.log(err));
        }
    }, [universityId]);
```

```
const handleSeePrijave = (courseId) => {
    navigate(`/prijava/${courseId}`);
};

const handleSeeStudents = (courseId) => {
    navigate(`/student/${courseId}`);
};
```

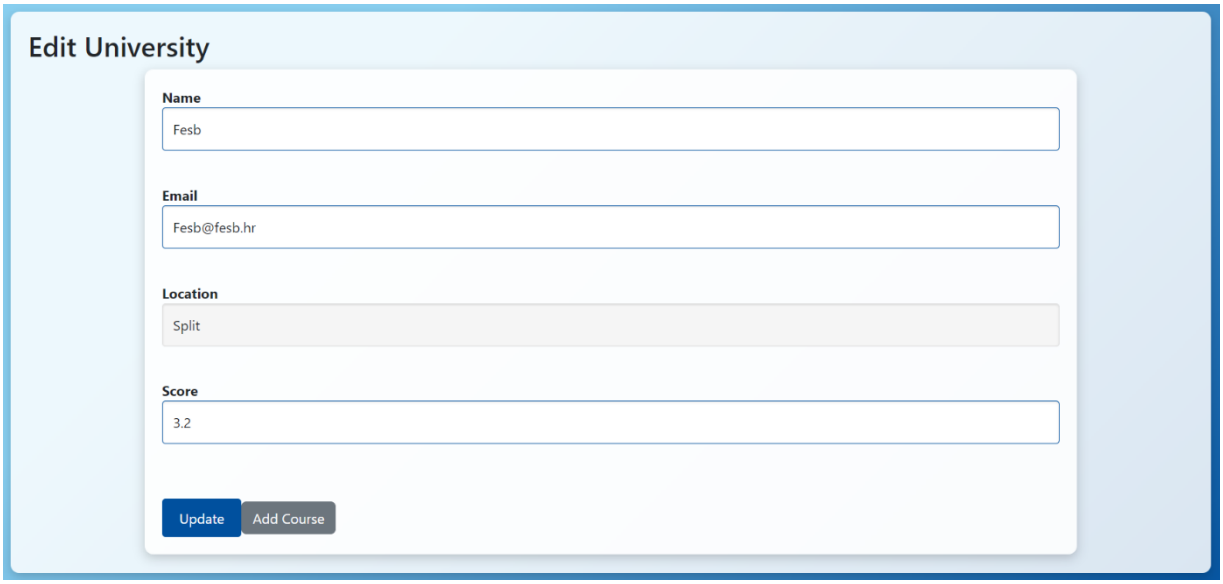
Ispis 8: Prikaz metoda za fakultetsko sučelje

Za prikaz svih studenata i svih prijava na studij, koriste se dvije funkcije (Ispis 8). Prva funkcija *get* pokreće se kada se promijeni vrijednost korisničkog identifikatora, a ako identifikator postoji, šalje se zahtjev za dohvaćanje podataka o korisniku. Ako je uloga povezana sa sveučilištem, postavljaju se identifikator sveučilišta i korisnička uloga u odgovarajuće varijable. Druga funkcija *get* aktivira se kada je postavljen identifikator sveučilišta. Ova funkcija šalje zahtjev za dohvaćanje popisa svih tečajeva koji se onda filtriraju kako bi se prikazali samo oni tečajevi koji pripadaju tom sveučilištu. Filtrirani podaci zatim se spremaju u odgovarajuću varijablu. Dodatno, kôd sadrži dvije funkcije koje omogućuju navigaciju korisnika na različite stranice aplikacije, bilo da se radi o prikazu prijava ili popisu studenata za određeni studij, a ovakav kôd dinamički upravlja prikazom i dostupnošću informacija unutar aplikacije (Slika 11).



Slika 11: Prikaz fakultetskog sučelja

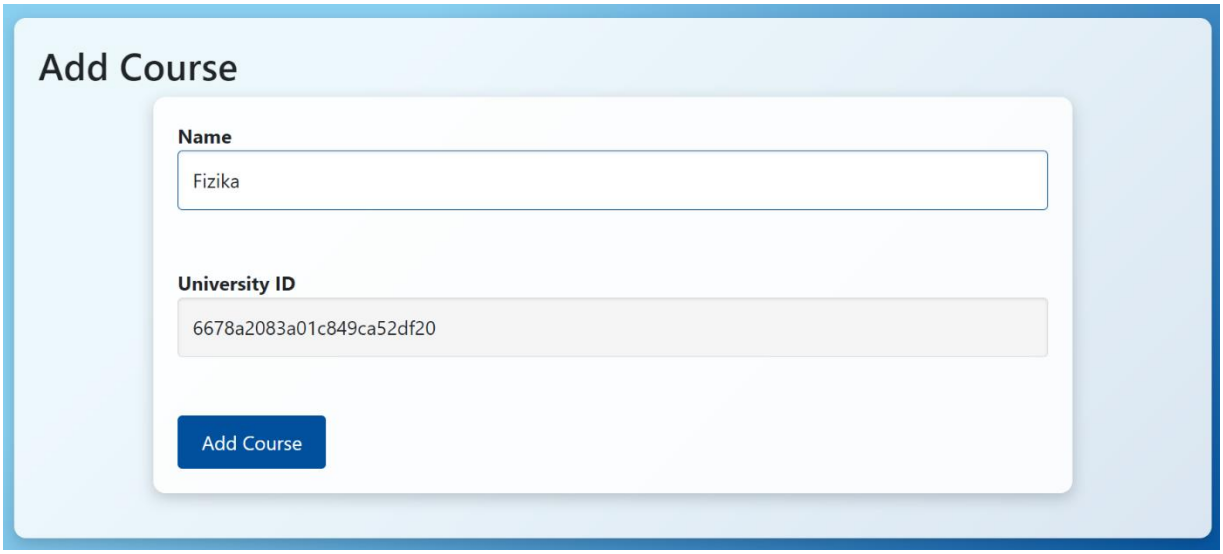
Također, fakulteti imaju mogućnost uređivanja svojih podataka pritiskom na dugme `Edit University`, gdje mogu mijenjati sve svoje podatke koji nisu u zavisnosti o drugima. Takvi podaci su *name*, *email* i *score*, a lokacija se ne može mijenjati jer je ona fiksna za svaki fakultet (Slika 12).



Slika 12: Forma za editiranje fakulteta

4.3.2. Dodavanje studija

Fakulteti imaju mogućnost dodavanja novih studija za svoj fakultet (Slika 13).



Slika 13: Forma za dodavanje studija

```
const AddCourse = () => {
    const navigate = useNavigate();
    const location = useLocation();
    const [courseName, setCourseName] = useState("");
    const [universityId, setUniversityId] = useState("");

    useEffect(() => {
```

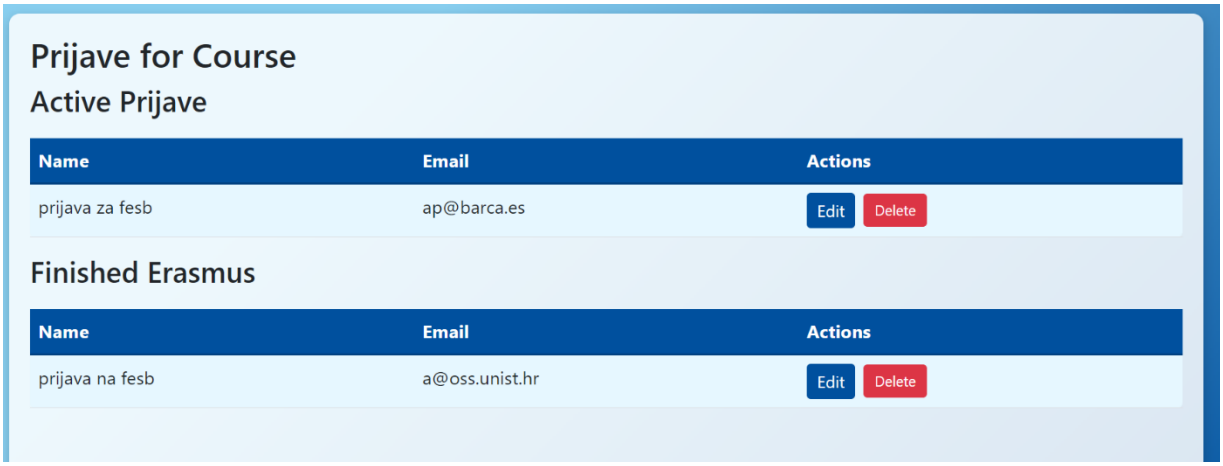
```
        const params = new URLSearchParams(location.search);
        const universityId = params.get('universityId');
        setUniversityId(universityId);
    }, [location]);

    const handleAddCourse = (e) => {
        e.preventDefault();
        fetch("http://localhost:3000/api/courses", {
            method: "POST",
            headers: { "Content-type": "application/json;charset=UTF-8" },
            body: JSON.stringify({
                name: courseName,
                university_id: universityId
            })
        })
        .then((resp) => resp.json())
        .then(() => navigate(`/editUniversity/${universityId}`))  // Redirect to EditUniversity
page after adding the course
        .catch((err) => console.log(err));
    };
```

Ispis 9: Prikaz metode za dodavanje studija

Na početku se koriste `useNavigate` i `useLocation` (Ispis 9) kako bi se omogućila navigacija unutar aplikacije te pristup stvaranju novog studija. Zatim se definiraju dva stanja: `courseName` za praćenje unosa naziva tečaja i `universityId` za pohranu identifikatora sveučilišta koji se ne može editirati jer je uvijek vezan za fakultet koji stvara novi studij. *UseEffect* funkcija dohvaća parametar `universityId` iz URL-a, koristeći `URLSearchParams` i postavlja ga u *state*. Ovo omogućuje komponenti da zna kojem sveučilištu pripada novi tečaj. Glavna funkcija `handleAddCourse` poziva se kada korisnik podnese obrazac za dodavanje tečaja i ona sprječava zadano ponašanje obrasca te šalje POST zahtjev na poslužitelj kako bi se novi tečaj dodao u bazu podataka. Nakon uspješnog dodavanja tečaja, korisnik se preusmjerava na stranicu za uređivanje sveučilišta putem `navigate` funkcije.

4.3.3. Upravljanje prijavama

Fakultet ima uvid u sve prijave studenata za željenu razmjenu na vlastitim studijima. Otvaranjem svih prijava, vidljivo je da se prijave dijele na aktivne i završene prijave. Aktivne prijave sadrže sve prijave koje su u stanju odobravanja od strane fakulteta ili se razmjena događa u datom trenutku. Fakultet ima opciju uređivanja i brisanja svih prijava (Slika 14).



Slika 14: Prikaz svih prijava za određeni studij

```
const EditPrijava = () => {
    const { id } = useParams();
    const navigate = useNavigate();
    const [prijava, setPrijava] = useState(null);
    const [check, setCheck] = useState(false);
    const [start, setStart] = useState(false);
    const [end, setEnd] = useState(false);
    const [studentCourse, setStudentCourse] = useState('');
    const [error, setError] = useState(null);

    useEffect(() => {
        fetch(`http://localhost:3000/api/prijave/${id}`, {
            method: 'GET',
            headers: { 'Content-type': 'application/json;charset=UTF-8' }
        })
        .then((resp) => resp.json())
        .then((data) => {
            setPrijava(data);
            setCheck(data.check);
            setStart(data.start);
            setEnd(data.end);

            fetch(`http://localhost:3000/student/course/${data.student_id._id}`, {
                method: 'GET',
                headers: { 'Content-type': 'application/json;charset=UTF-8' }
            })
            .then((resp) => resp.json())
            .then((studentData) => {
                if (studentData && studentData.course_id) {
                    setStudentCourse(studentData.course_id.name);
                }
            })
            .catch((err) => {
                console.log(err);
                setError(err.message);
            });
        })
        .catch((err) => {
            setError(err.message);
            console.log(err);
        });
    }, [id]);
```

```
    const handleCheckChange = () => {
        setCheck(!check);
    };

    const handleStartChange = () => {
        setStart(!start);
    };

    const handleEndChange = () => {
        setEnd(!end);
    };

    const handleSaveChanges = () => {
        const updatedData = {
            check,
            start,
            end
        };

        fetch(`http://localhost:3000/api/prijave/${id}`, {
            method: 'PUT',
            headers: { 'Content-type': 'application/json;charset=UTF-8' },
            body: JSON.stringify(updatedData)
        })
        .then((resp) => resp.json())
        .then((data) => {
            console.log('Updated Prijava:', data);
            navigate('/admin/prijave'); // Redirect to AdminPrijave after saving changes
        })
        .catch((err) => {
            console.error('Error updating Prijava:', err);
            setError(err.message);
        });
    };
```

Ispis 10: Komponenta za editiranje prijava

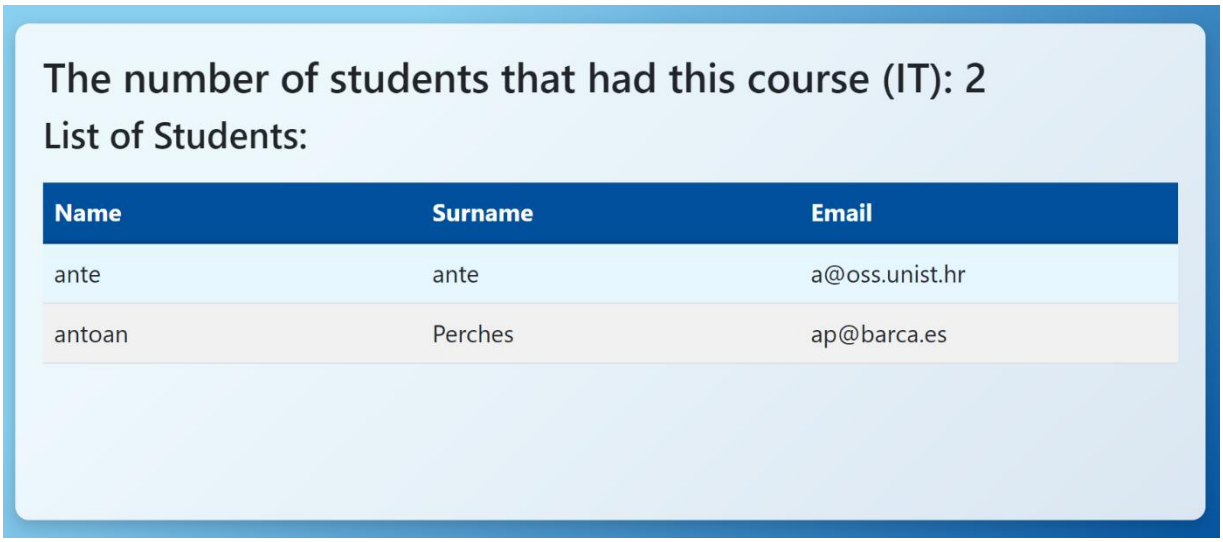
Komponenta `EditPrijava` (Ispis 10), koja omogućuje uređivanje postojećih prijava u sustavu. Komponenta koristi nekoliko *hookova* kao što su `useParams`, `useNavigate`, i `useState` za upravljanje stanjem, navigacijom i dohvatom parametara nakon čega se dobiveni podaci pohranjuju u state varijable poput `prijava`, dok se polja kao što su `check`, `start` i `end` inicijaliziraju na temelju dohvaćenih podataka. Istovremeno, komponenta šalje dodatni *get* zahtjev kako bi dohvatila informacije o tečaju studenta povezanog s prijavom i prikazala ih u odgovarajućem polju. Fakultet ima mogućnost da izmijeni vrijednosti za `check`, `start` i `end` kroz *checkboxove*. Kada fakultet klikne na gumb za spremanje promjena, funkcija `handleSaveChanges` šalje *put* zahtjev s ažuriranim podacima na server. Nakon uspješnog spremanja promjena, korisnik se preusmjerava na stranicu s popisom svih prijava. Ukoliko je fakultet označio opcije `Check` i `Start` onda se prijava postavlja kao aktivna, a ako je

označena i opcija `End` onda je razmjena završena i student ima mogućnost ostaviti recenziju na odrađeni studij (Slika 15).

Slika 15: Forma za editiranje prijave

4.3.4. Pregled liste studenata

Fakultet ima mogućnost pregledavanja svih studenata koji su prošli kroz određeni studij na tom fakultetu (Slika 16).



Slika 16: Prikaz liste studenata na studiju

```
useEffect(() => {
        fetch(`http://localhost:3000/api/courses/${id}`, {
            method: 'GET',
            headers: { 'Content-type': 'application/json;charset=UTF-8' }
        })
        .then((resp) => resp.json())
        .then((course) => {
            setCourseName(course.name);
        })
        .catch((err) => console.log(err));

        fetch(`http://localhost:3000/api/prijave`, {
            method: 'GET',
            headers: { 'Content-type': 'application/json;charset=UTF-8' }
        })
        .then((resp) => resp.json())
        .then((data) => {
            const enrolledStudents = data.filter(prijava => prijava.course_id._id === id)
                                         .map(prijava => prijava.student_id);
            setStudents(enrolledStudents);
        })
        .catch((err) => console.log(err));
    }, [id]);
```

Ispis 11: Metoda za prikaz studenata koji su prošli kroz studij

Ova metoda (Ispis 11) koristi `useEffect` funkciju za dohvaćanje podataka s poslužitelja kada se komponenta učita ili kada se promijeni vrijednost varijable `id`. Prvi zahtjev dohvaća podatke o specifičnom tečaju na temelju identifikatora `course_id`. Kada podaci stignu, `courseName` se postavlja u odgovarajuću *state* varijablu pomoću funkcije `setCourseName`. Drugi zahtjev dohvaća sve prijave iz baze podataka. Nakon što podaci stignu, oni se filtriraju kako bi se pronašle prijave koje odgovaraju zadanom indetifikatoru `course_id`. Zatim se iz tih prijava izvlače podaci o studentima koji su prijavljeni na tečaj, a ti studenti se pohranjuju u *state* varijablu `setStudents`. Ako dođe do greške tijekom bilo kojeg od ovih zahtjeva, greška se ispisuje u konzolu.

4.4. Funkcionalnosti: Student

4.4.1. Pregled fakulteta i studija

Studentsko sučelje je prikaz svih fakulteta i njihovih studija, sadrži i opciju filtriranja studija po nazivu ili fakultetu.

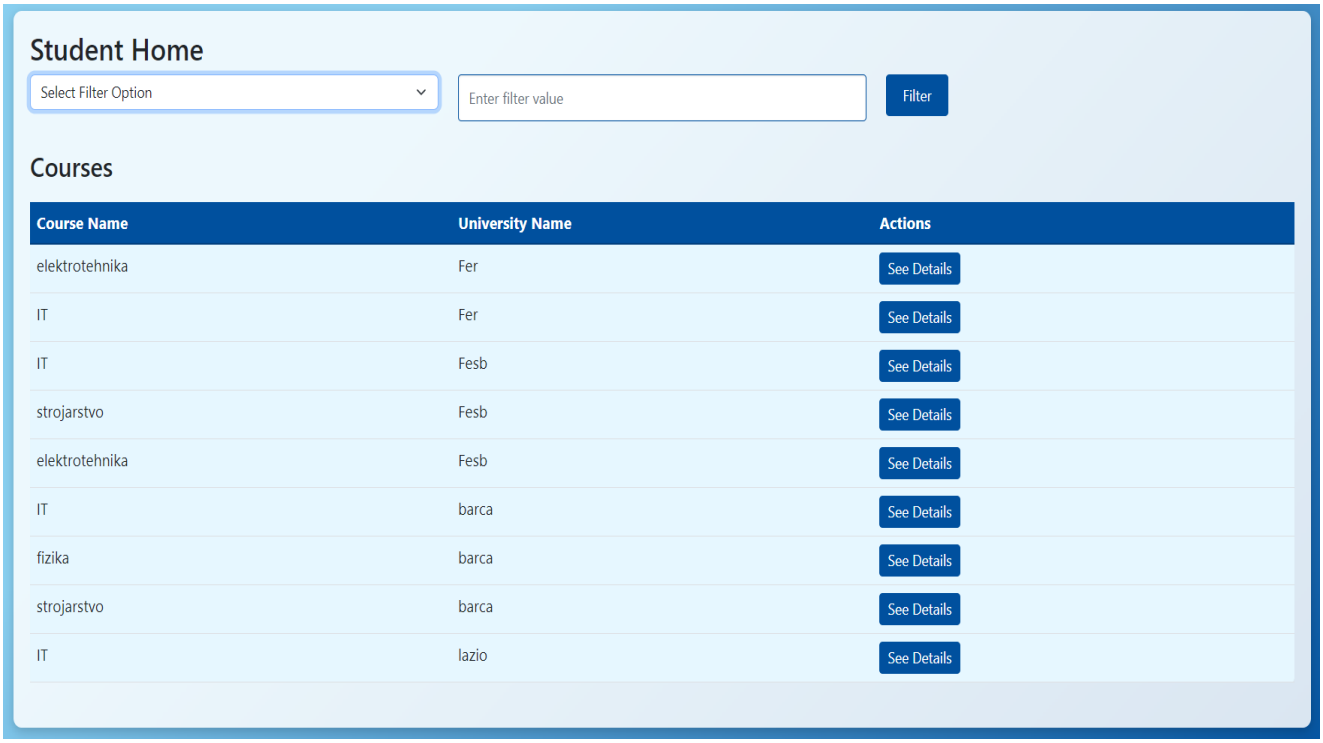
```
useEffect(() => {
        if (userId) {
```

```
            fetch(`http://localhost:3000/user/${userId}`, {
                method: 'GET',
                headers: { 'Content-type': 'application/json;charset=UTF-8' }
            })
            .then((resp) => resp.json())
            .then((userData) => {
                const studentId = userData.student._id;

                fetch(`http://localhost:3000/student/${studentId}`, {
                    method: 'GET',
                    headers: { 'Content-type': 'application/json;charset=UTF-8' }
                })
                .then((resp) => resp.json())
                .then((studentData) => {
                    const studentCourseId = studentData.course_id;

                    fetch(`http://localhost:3000/api/courses/${studentCourseId}`, {
                        method: 'GET',
                        headers: { 'Content-type': 'application/json;charset=UTF-8' }
                    })
                    .then((resp) => resp.json())
                    .then((courseData) => {
                        const universityId = courseData.university_id._id;

                        fetch('http://localhost:3000/api/courses', {
                            method: 'GET',
                            headers: { 'Content-type': 'application/json;charset=UTF-8' }
                        })
                        .then((resp) => resp.json())
                        .then((data) => {
                            const filteredCourses = data.filter(course => {
                                return course.university_id._id.toString() !==
universityId.toString();
                            });
                            setUniversityId(universityId);
                            setCourses(filteredCourses);
                            setFilteredCourses(filteredCourses); // Initialize filtered courses
with all courses
                        })
                        .catch((err) => console.log(err));
                    })
                    .catch((err) => console.log(err));
                })
                .catch((err) => console.log(err));
            })
            .catch((err) => console.log(err));
        }
    }, [userId]);

    const handleFilter = () => {
        if (filterOption === "university") {
            const filtered = courses.filter(course =>
course.university_id.name.toLowerCase().includes(filterValue.toLowerCase()));
            setFilteredCourses(filtered);
        } else if (filterOption === "courseName") {
            const filtered = courses.filter(course =>
course.name.toLowerCase().includes(filterValue.toLowerCase()));
            setFilteredCourses(filtered);
        } else {
            setFilteredCourses(courses);
        }
    };
```

```
    const handleFilterOptionChange = (e) => {
        setFilterOption(e.target.value);
    };

    const handleFilterValueChange = (e) => {
        setFilterValue(e.target.value);
    };

    const handleCourseClick = (courseId) => {
        navigate(`/course/${courseId}?userId=${userId}`);
    };
```

Ispis 12: Komponenta za prikaz studentskog sučelja

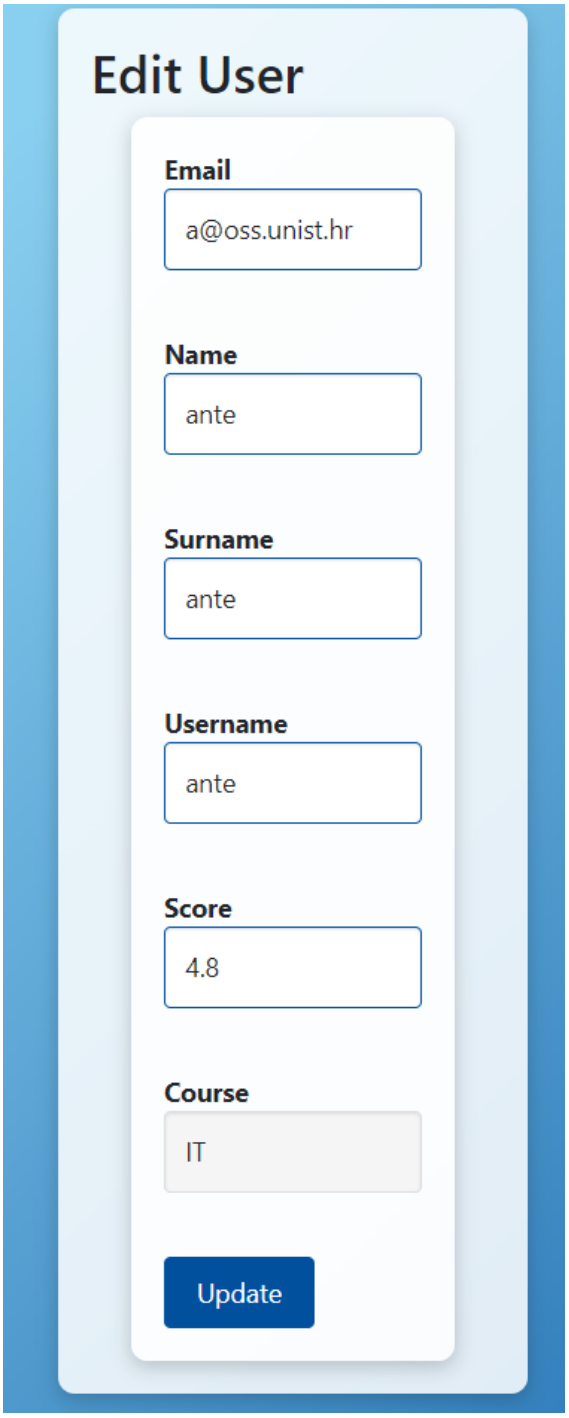
Student ima mogućnost pregledavanja svih studija koji se ne nalaze na fakultetu na kojem student trenutno studira na način da API poziv dohvaća sve dostupne tečajeve, koji se potom filtriraju kako bi se izbacili oni povezani sa studentovim fakultetom (Ispis 12). Filtrirani tečajevi spremaju se u listu `filteredCourses`, a ta lista inicijalno postaje osnovna lista studija. Kako bi student lakše pretraživao željenu lokaciju za razmjenu koristi se Funkcija `handleFilter` koja omogućuje korisniku filtriranje tečajeva prema odabranom kriteriju, bilo prema fakultetu ili nazivu studija. Kada student izabere neku od opcija tada funkcije `handleFilterOptionChange` i `handleFilterValueChange` ažuriraju stanje filtera. Na kraju, student klikom na neki tečaj pokreće funkciju `handleCourseClick` koja omogućuje navigaciju prema stranici specifičnog tečaja, uz uključivanje `userId` u URL kako bi se osigurao kontekst studenta (Slika 17).



Slika 17: Prikaz studentskog sučelja

4.4.2. Izmjena vlastitih podataka

Svaki student ima mogućnost izmjene vlastitih podataka, ali samo onih koji nisu definirani s određenim uvjetima (Slika 18).



Slika 18: Prikaz forme za editiranje korisnika

```
                    <label>Name</label>
                    <input
                        type="text"
                        className="form-control"
                        value={name}
                        onChange={(e) => setName(e.target.value)}
```

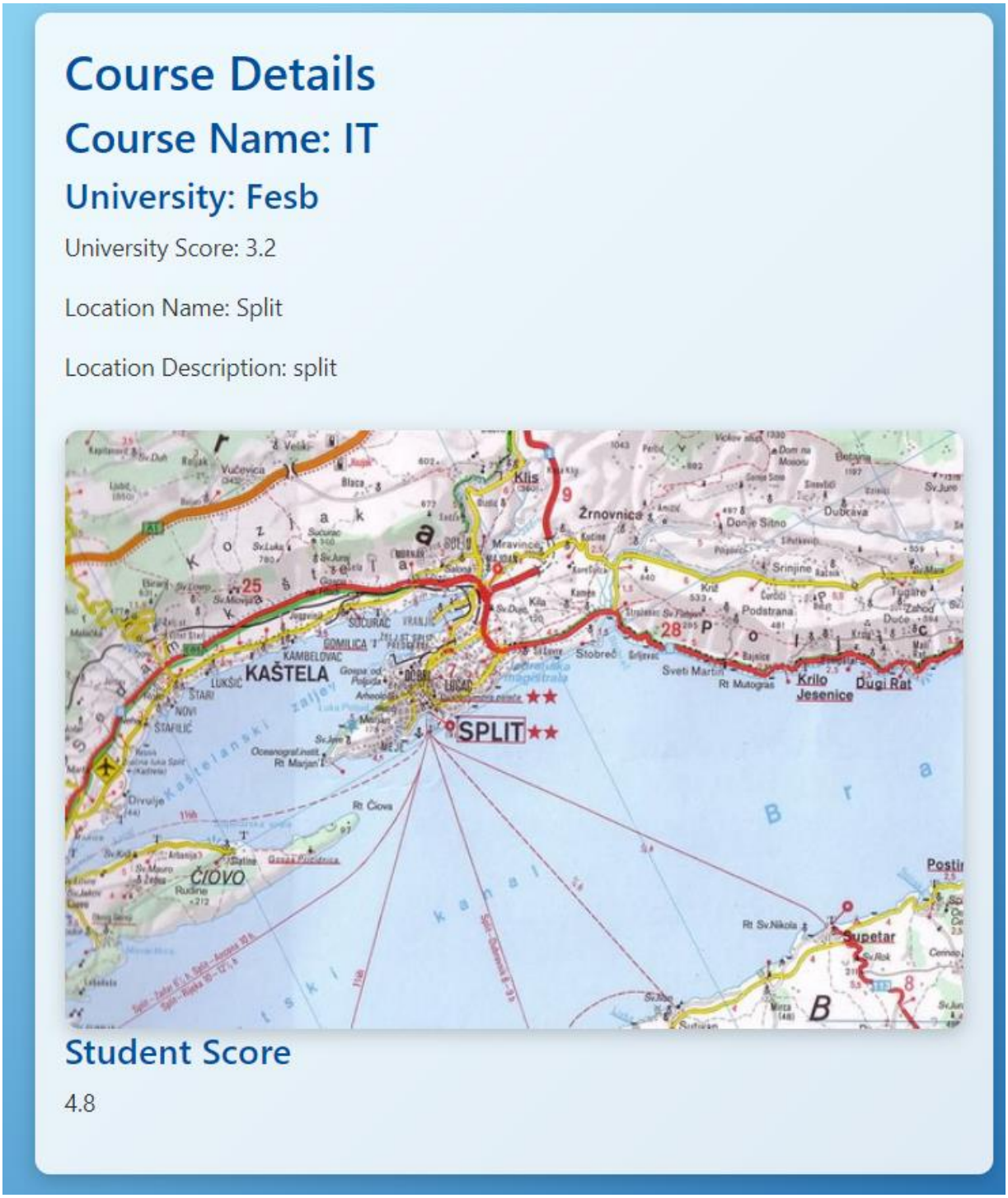
```
                    required
                /><br/>
                <label>Surname</label>
                <input
                    type="text"
                    className="form-control"
                    value={surname}
                    onChange={(e) => setSurname(e.target.value)}
                    required
                /><br/>
                <label>Username</label>
                <input
                    type="text"
                    className="form-control"
                    value={username}
                    onChange={(e) => setUsername(e.target.value)}
                    required
                /><br/>
                <label>Score</label>
                <input
                    type="number"
                    className="form-control"
                    value={score}
                    onChange={(e) => setScore(e.target.value)}
                /><br/>
                {course && (
                    <>
                        <label>Course</label>
                        <input
                            type="text"
                            className="form-control"
                            value={course.name}
                            readOnly
                        /><br/>
                    </>
                )}
```

Ispis 13: HTML elementi za prikaz forme za editiranje korisnika

Studentu se prikazuje niz ulaznih polja unutar obrazaca za unos podataka kao što su ime, prezime, korisničko ime, bodovi i studij. Svako polje je povezano s odgovarajućom vrijednosti stanja, a promjene u tim poljima ažuriraju odgovarajuće stanje pomoću funkcija kao što su `setName`, `setSurname`, `setUsername` i `setScore` (Ispis 13). Osim toga, ako postoji podatak o studiju prikazuje se i dodatno polje za prikaz naziva studija, ali ovo polje je samo za čitanje, što znači da korisnik ne može mijenjati njegovu vrijednost. Ovaj način postavljanja omogućuje jednostavno prikupljanje i prikazivanje informacija specifičnih za studente unutar korisničkog sučelja.

4.4.3. Pregledavanje određenog studija

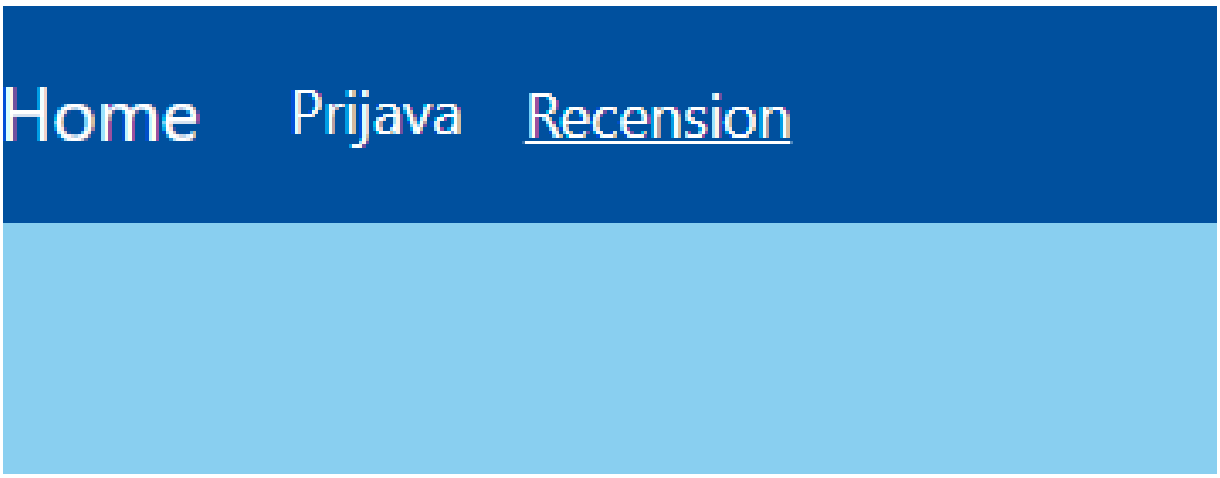
Student ima mogućnost pregledavanja specifičnog studija na željenom fakultetu kako bi mogao vidjeti sve informacije o lokaciji na kojoj se nalazi fakultet i uvjetima koji su potrebni kako bi student ispunio prijavu za razmjenu (Slika 19).



Slika 19: Prikaz detalja određenog studija

Nakon pregledavanja uvjeta i detalja za željeni studij, student ukoliko ispunjava kriterij za prijavu ima mogućnost slanja prijave za taj studij klikom na dugme `Prijava`, a ako prije

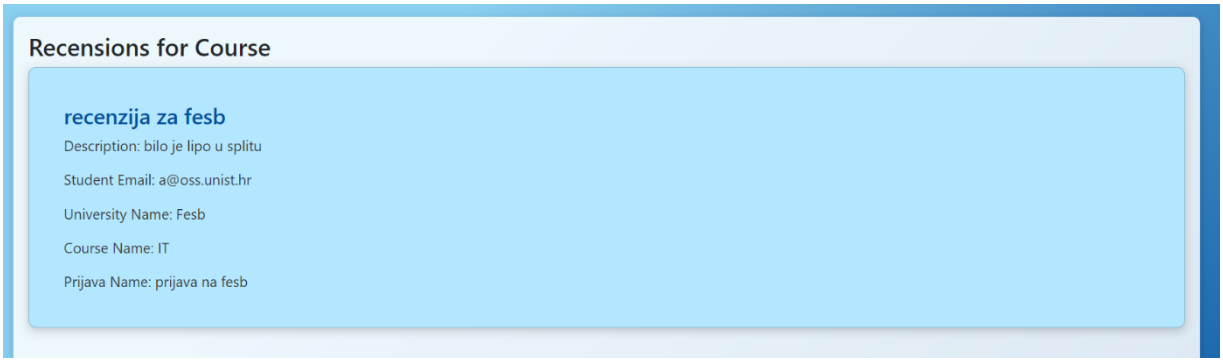
prijave želi vidjeti iskustva drugih studenata onda može kliknuti na dugme `Recension` (Slika 20).



Slika 20: Prikaz navigacijske trake

4.4.4. Pregledavanje recenzija za studij

Svaki student ima pravo na pregledavanje recenzija o nekom studiju bez obzira da li zadovoljava uvjete za prijavu na taj studij ili ne (Slika 21).



Slika 21: Prikaz recenzije za određeni studij

Ukoliko je student završio razmjenu za određeni studij onda ima priliku ostaviti recenziju kako bi olakšao i pomogao sljedećim studentima ili onima koji se razmišljaju oko prijavljivanja na razmjenu.

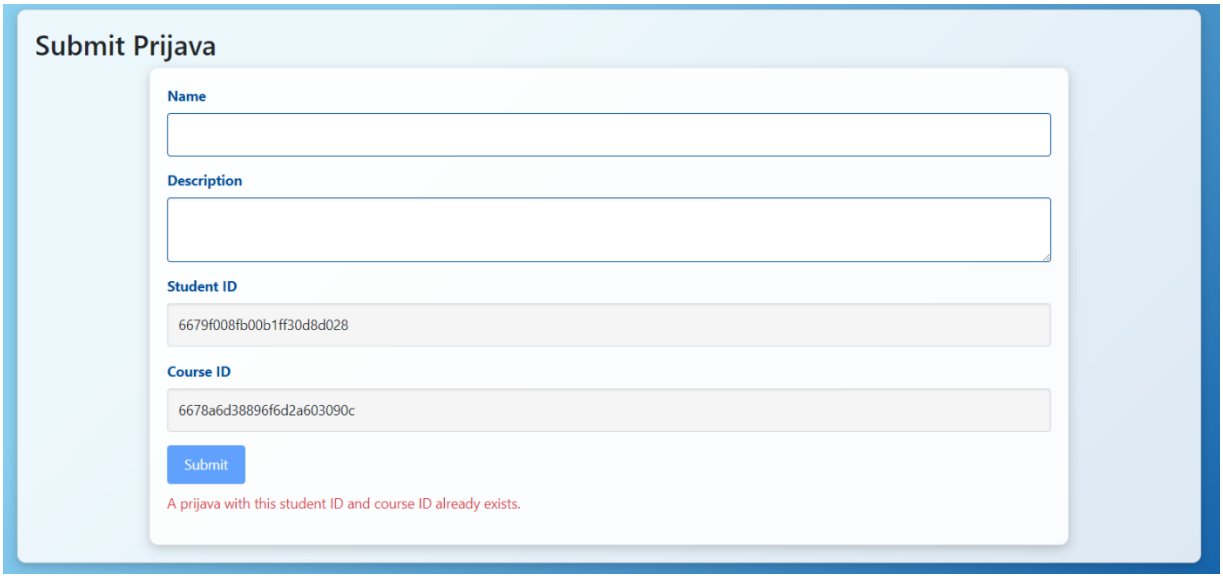
4.4.5. Prijava

Student ima pravo ispunjavanja prijave na izabrani studij ako zadovoljava postavljene uvjete.

```
<form onSubmit={handleSubmit}>
                    <div className="mb-3">
                        <label htmlFor="name" className="form-label">Name</label>
                        <input
                            type="text"
                            className="form-control"
                            id="name"
                            value={name}
                            onChange={(e) => setName(e.target.value)}
                            required
                        />
                    </div>
                    <div className="mb-3">
                        <label htmlFor="description" className="form-label">Description</label>
                        <textarea
                            className="form-control"
                            id="description"
                            value={description}
                            onChange={(e) => setDescription(e.target.value)}
                            required
                        />
                    </div>
                    <div className="mb-3">
                        <label htmlFor="student_id" className="form-label">Student ID</label>
                        <input
                            type="text"
                            className="form-control"
                            id="student_id"
                            value={studentId || ''}
                            readOnly
                        />
                    </div>
                    <div className="mb-3">
                        <label htmlFor="course_id" className="form-label">Course ID</label>
                        <input
                            type="text"
                            className="form-control"
                            id="course_id"
                            value={courseId || ''}
                            readOnly
                        />
                    </div>
                    <button type="submit" className="btn btn-primary"
disabled={isDuplicate}>Submit</button>
                    {isDuplicate && <p className="text-danger">A prijava with this student ID and
course ID already exists.</p>}
                </form>
```

Ispis 14: HTML elementi za prikaz forme prijave

Forma prijave sastoji se od 4 četiri polja (Ispis 14), a student ispunjava samo polja `Name` i `Description` jer polja `Student ID` i `Course ID` se automatski popunjavaju sa studentovim podacima i podacima studija (Slika 22).



Slika 22: Prikaz forme za prijavu

```
useEffect(() => {
        const duplicate = existingPrijave.some(prijava => prijava.student_id._id === studentId &&
prijava.course_id._id === courseId);
        setIsDuplicate(duplicate);
    }, [existingPrijave, studentId, courseId]);

    const handleSubmit = (e) => {
        e.preventDefault();

        const prijava = {
            name,
            student_id: studentId,
            course_id: courseId,
            description
        };
```

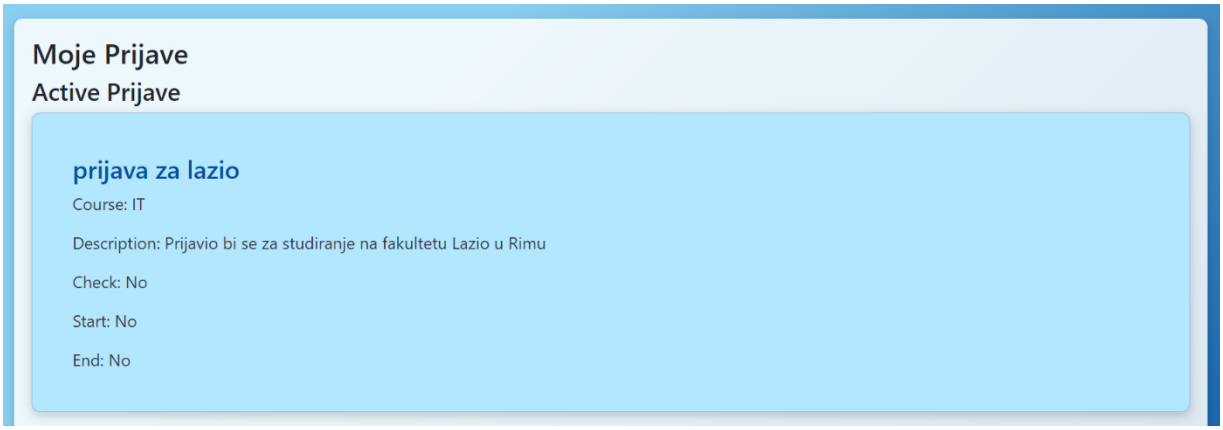
Ispis 15: Metoda za slanje prijave

Ukoliko se student već prijavio za razmjenu, javlja se poruka s greškom i onemogućuje pritiskanje dugmeta `Submit` jer se pokreće metoda `useEffect` svaki put kada se ažuriraju `existingPrijave`, `studentId` ili `coursed` (Ispis 15). Ako ne postoji poruka s greškom onda se pokreće funkcija `handleSubmit` i kreira se *post* zahtjev za slanje obrazca. Ako je

slanje uspješno, u konzolu se ispisuje potvrda, a korisnik se preusmjerava na stranicu StudentHome. U slučaju pogreške prilikom slanja, ona se ispisuje u konzoli.

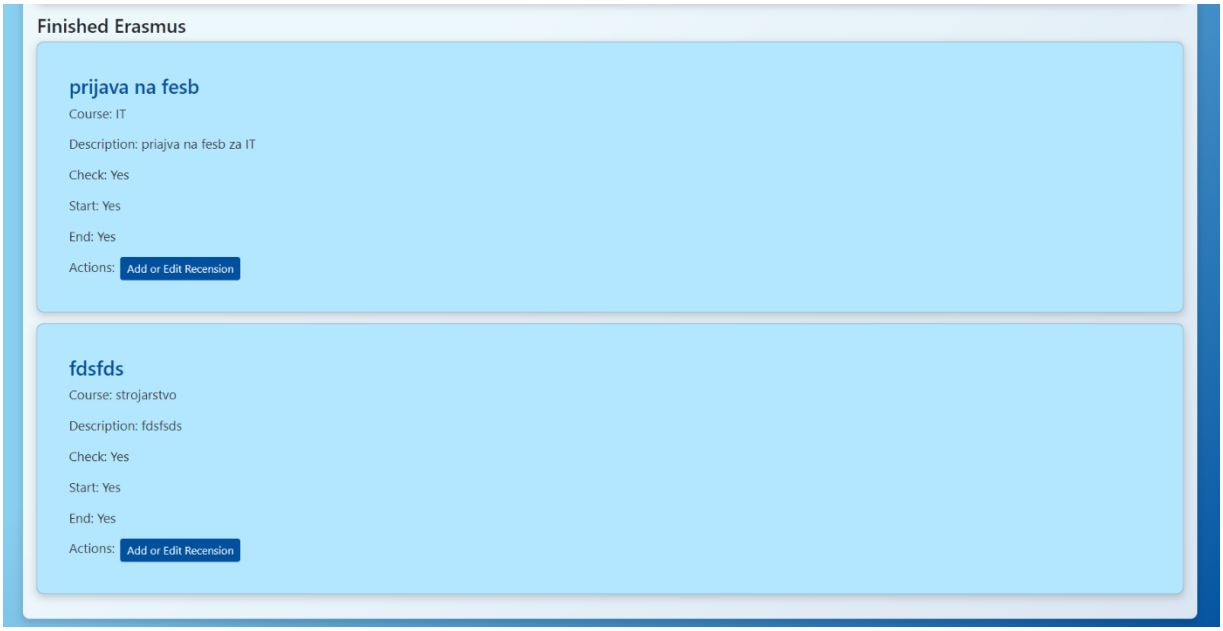
4.4.6. Pregled vlastitih prijava i mogućnost ažuriranja

Svaki student ima mogućnost pregledavanja vlastitih prijava i ažuriranja istih. Prijave studenta dijele se na aktivne, a to su one kod kojih razmjena još nije počela ili one gdje je student trenutno na razmjeni (Slika 23).



Slika 23: Prikaz aktivnih prijava

Osim aktivnih prijava postoje i završene prijave, a to su one kod kojih je student već obavio razmjenu i uz prijavu ima mogućnost ostavljanja i editiranja recenzije (Slika 24).



Slika 24: Prikaz završenih prijava

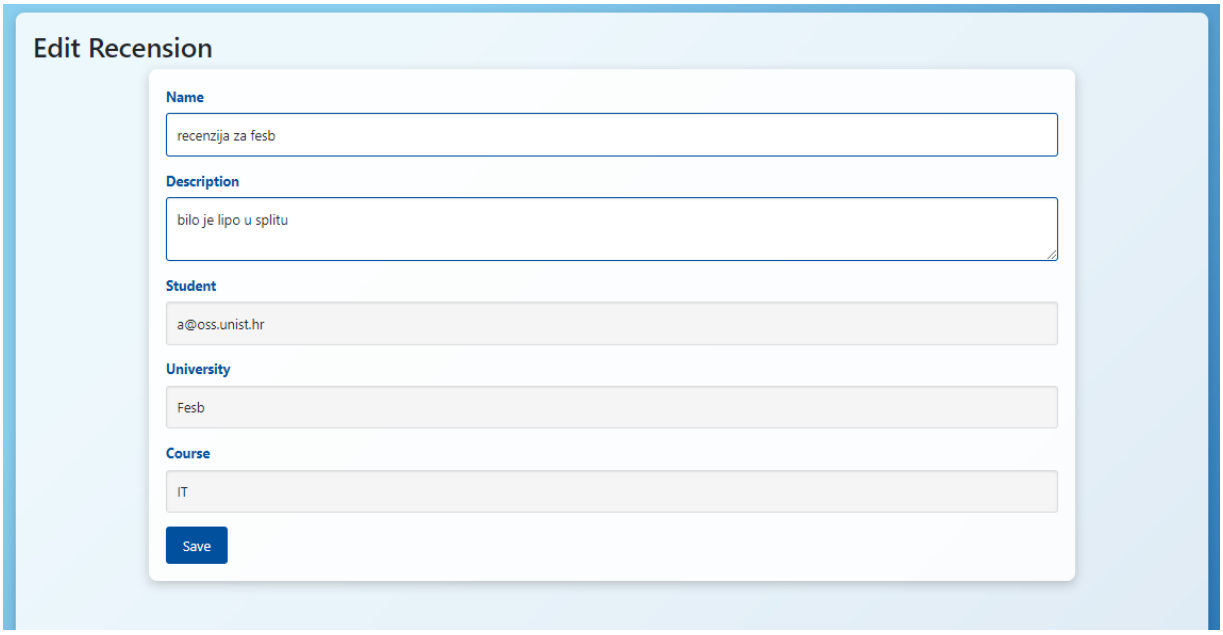
```
const MojePrijave = () => {
    const location = useLocation();
    const navigate = useNavigate();
    const [user, setUser] = useState(null);
    const [student, setStudent] = useState(null);
    const [prijave, setPrijave] = useState([]);
    const [activePrijave, setActivePrijave] = useState([]);
    const [finishedErasmus, setFinishedErasmus] = useState([]);
    const [recensions, setRecensions] = useState([]);

    useEffect(() => {
        const params = new URLSearchParams(location.search);
        const userId = params.get('userId');

        if (userId) {
            fetch(`http://localhost:3000/user/${userId}`, {
                method: 'GET',
                headers: { 'Content-type': 'application/json;charset=UTF-8' }
            })
            .then((resp) => resp.json())
            .then((userData) => {
                setUser(userData.user);
                if (userData.user.role === 'student') {
                    setStudent(userData.student);
                }
            })
            .catch((err) => console.log(err));
        }
    }, [location]);

    useEffect(() => {
        fetch('http://localhost:3000/api/prijave', {
            method: 'GET',
            headers: { 'Content-type': 'application/json;charset=UTF-8' }
        })
        .then((resp) => resp.json())
```

```
        .then((data) => {
            setPrijave(data);
        })
        .catch((err) => console.log(err));
    }, []);

    useEffect(() => {
        fetch('http://localhost:3000/api/recensions', {
            method: 'GET',
            headers: { 'Content-type': 'application/json;charset=UTF-8' }
        })
        .then((resp) => resp.json())
        .then((data) => {
            setRecensions(data);
        })
        .catch((err) => console.log(err));
    }, []);

    useEffect(() => {
        if (student) {
            const filteredPrijave = prijave.filter(prijava => prijava.student_id._id ===
student._id);
            const active = filteredPrijave.filter(prijava => !prijava.end);
            const finished = filteredPrijave.filter(prijava => prijava.end);
            setActivePrijave(active);
            setFinishedErasmus(finished);
        }
    }, [prijave, student, recensions]);

    const handleAddOrEditRecension = (studentId, prijavaId, courseId) => {
        const existingRecension = recensions.find(recension => recension.student_id._id ===
studentId && recension.prijava_id === prijavaId);
        if (existingRecension) {
            navigate(`/edit/recension/${existingRecension._id}`);
        } else {
            navigate(`/recension?student_id=${studentId}&prijava_id=${prijavaId}&course_id=${cour
seId}`);
        }
    };
```

Ispis 16: Komponenta za prikaz korisnikovih prijava

Komponenta `MojePrijave` (Ispis 16) je odgovorna za prikaz i upravljanje prijavama studenta, uključujući njihovo filtriranje na aktivne i završene prijave te povezivanje s recenzijama. Unutar komponente koristi se nekoliko *hookova*, uključujući `useState` za upravljanje stanjem i `useEffect` za pokretanje API poziva i ažuriranje stanja kada se promijeni određeni uvjet. Na početku se koristi `useEffect` za dohvaćanje korisničkih podataka temeljem `userId`, a onda drugi `useEffect` pokreće *get* zahtjev za dohvaćanje svih prijava i pohranjuje te prijave u stanje `prijave`. Ključni `useEffect` u ovoj komponenti filtrira prijave koje su povezane s prijavljenim studentom, a prijave se dijele na aktivne i

završene. Funkcija `handleAddOrEditRecension` omogućuje korisniku da doda novu recenziju ili uredi postojeću (Slika 25).



Slika 25: Prikaz forme za editiranje recenzija

5. Zaključak

U ovom je diplomskom radu opisan proces izrade web stranice za razmjenu studenata koja omogućuje povezivanje fakulteta i samih studenata koji se mogu prijaviti za odlazak odrađivanja semestra nekog studija na drugom fakultetu. Najveći izazov u izradi aplikacije je bio povezivanje svih tehnologija i usklađivanje njihovih verzija kako bi bili kompatibilni. Za izradu aplikacije izabran je React koji ima složeniju strukturu, veći opseg mogućnosti i kompatibilan je s Node.jsom u svim verzijama i bibliotekama zbog čega je izabran umjesto Vue.jsa. Za razliku od Reacta, Vue.js je puno jednostavniji, ali zbog toga ima mogućnost boljeg upravljanja *cacheom* i podacima, što ga čini nešto bržim u prikazu sučelja. Za izradu *backenda* izabran je Node.js zbog svoje jednostavnosti kod izrade CRUD operacija koje su ključ za uspješnu izradu aplikacije jer omogućuje spremanje podataka u bazu podataka i prikazivanje podataka na sučelju.

Ovakva platforma sada studentima omogućuje jednostavan pristup prilikama za razmjenu i stjecanje novih znanja u području njihovog interesa, što je posebno važno za njihov daljnji napredak i profesionalni razvoj. Kroz ovu web aplikaciju mogu pregledavati popis dostupnih razmjena i recenzija na određene studije na koje možda imaju želju otići. Dok fakulteti mogu učinkovito upravljati svojim podacima za razmjenu, pregledavati prijave studenata i odabrati najprikladnije kandidate za razmjenu kako bi omogućili nova iskustva i znanja nekima od njih. Ovakav sustav omogućuje fakultetima da razvijaju i dobiju nove informacije od studenata koji nisu iz iste države ili istog grada, a studentima omogućuje neka nova znanja i vještine, što može doprinijeti njihovom rastu i razvoju. Administratoru aplikacije omogućen je uvid u sve podatke na stranici i može upravljati korisnicima kao i podacima vezanima za prijave, studije i recenzije.

Za poboljšanje ove aplikacije najvažnija je povratna informacija korisnika, koja sadrži odgovore za buduću nadogradnju i poboljšanje sustava. Aplikacija bi u budućnosti trebala staviti prioritet na povećanju sigurnosti, a to se može postići uvođenjem višefaktorske autentifikacije i strožim pravilima za izradu korisničkog računa. Na kraju, ova aplikacija zadovoljava trenutne potrebe tržišta, ali postoji mogućnost poboljšanja kroz otvorenu komunikaciju s korisnicima i ulaganje u razvoj i marketing aplikacije.

Literatura

1. https://www.popwebdesign.net/sta-je-react.html (posjećeno 06.08.2024)
2. https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs (posjećeno 06.08.2024)
3. https://expressjs.com/en/5x/api.html (posjećeno 06.08.2024)
4. https://www.techtarget.com/searchdatamanagement/definition/MongoDB (posjećeno 06.08.2024)
5. https://developer.mozilla.org/en-US/docs/Web/HTML (posjećeno 06.08.2024)
6. https://developer.mozilla.org/en-US/docs/Web/CSS (posjećeno 06.08.2024)