RESTful web aplikacija za upravljanje zahtjevima

Paris, Filip

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni prijediplomski studij računarstva

Završni rad

RESTful web aplikacija za upravljanje zahtjevima

Rijeka, rujan 2023.

Filip Paris
0069090358

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni prijediplomski studij računarstva

Završni rad

RESTful web aplikacija za upravljanje zahtjevima

Mentor: doc.dr.sc. Marko Gulić

Rijeka, rujan 2023.

Filip Paris
0069090358

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 13. srpnja 2023.

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

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Filip Paris (0069090358)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **RESTful web aplikacija za upravljanje zahtjevima / RESTful web application for request management**

Opis zadatka:

Razviti RESTful web aplikaciju za upravljanje zahtjevima. Aplikacija mora podržavati uslugu kreiranja zahtjeva, kao i njegovog dodjeljivanja pojedinom tehničaru koji će izvršiti zadani zahtjev. Također, mora postojati mogućnost ažuriranja, brisanja ili zatvaranja zahtjeva. Aplikacija mora podržavati funkcionalnost dodavanja podzahtjeva radi učinkovitijeg praćenja izvršavanja zahtjeva. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Laravel radni okvir uz proizvoljno odabran sustav za upravljanje bazama podataka uz korištenje paketa za realizaciju autentifikacije RESTful aplikacije. Za razvoj klijentskog dijela aplikacije treba koristiti React JavaScript knjižicu uz korištenje Bootstrap JavaScript knjižice za učinkovito renderiranje aplikacije na uređajima s različitim veličinama zaslona.

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

Zadatak uručen pristupniku: 4. rujna 2023.

Mentor:

Doc. dr. sc. Marko Gulić

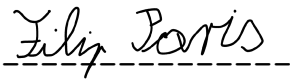
Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.



Ime Prezime

Sadržaj

Popis slika

Popis kodnih isječaka

Poglavlje 1

Uvod

U ovom radu napravljena je RESTful web aplikacija za evidentiranje i rješavanje različitih zahtjeva koje korisnici prijave, npr. nestanak interneta, vode ili struje. Također se može koristiti kao alat za praćenje stanja izvršavanja navedenih zahtjeva. U kontekstu ovog projekta, aplikacija ima ulogu podrške tokom razvoja softverskih proizvoda omogućujući evidentiranje zahtjeva za stvaranjem tih proizvoda.

Svaki generirani zahtjev sadržava naziv kako bi se jasno identificirala njegova svrha. Osim toga, posjeduje opis koji pruža dublje objašnjenje problema koji je potaknuo njegovo otvaranje. Status zahtjeva odražava trenutnu fazu unutar njegovog životnog ciklusa i omogućuje uvid u napredak u njegovom rješavanju. Zahtjev je pokrenut za određenog klijenta zbog njegove potrebe za određenim uslugama. Naposljetku, zahtjev sadrži informacije o tehničarima odgovornim za njegovo rješavanje.

Aplikacija također nudi mogućnost kreiranja podzahtjeva za svaki zatraženi zahtjev, omogućujući precizno definiranje koraka potrebnih za njegovo potpuno dovršavanje. Dodatno, na svaki zahtjev je moguće dodavati komentare kako bi se znalo u kojoj fazi rješavanja se nalazi te ukoliko je došlo do određenih problema. Također, ovisno o ulozi korisnika koji je prijavljen, aplikacija omogućava prikaz i uređivanje popisa tehničara i klijenata.

Za kreiranje ove aplikacije korištene su sljedeće tehnologije: React, Laravel, Bootstrap te MySQL.

Stilizacija pojedinih stranica aplikacije, kao i pojedinih elemenata na svakoj stranici je realizirana pomoću Bootstrapa [1]. Kod korištenja Bootstrapa nije potrebno

ručno pisati veliku količinu CSS koda, već je sve implementirano u klasama koje se koriste na pojedinim elementima stranice. Zbog toga je njegovo korištenje poprilično jednostavno i izrazito učinkovito.

Kao što je spomenuto, Bootstrap tehnologija se bazira na kodu napisanom CSS tehnologijom. CSS [2] zapravo opisuje kako elementi trebaju izgledati na ekranu, tj. koristi se za opisivanje pojedinih elemenata HTML dokumenta. HTML [3] je bazna tehnologija koja se koristi pri izradi web aplikacija. Prilikom početka izrade bilo kakve *web* aplikacije, započinje se upravo s HTML datotekom.

Dinamični prikaz stranica je realiziran pomoću JavaScript biblioteke React [4] izrazito popularne zbog velike raznolikosti i velike količine literature dostupne na internetu. Nudi jednostavan pristup, visoku efikasnost te značajni potencijal za produktivnost, što olakšava izgradnju interaktivnih korisničkih sučelja.

Za omogućavanje različitih funkcionalnosti koje nisu vidljive na samoj aplikaciji jer se izvršavaju u pozadini se koristi programski jezik PHP [5], točnije njegov programski okvir Laravel [6]. To uključuje spajanje na bazu podataka, korištenje upravljača za manipulaciju s podacima te autenifikaciju korisnika za neometano korištenje aplikacije. Za upravljanje i pohranu podataka je korišten lokalno pokrenut MySQL [7] sustav kao server u pozadinskom načinu rada.

Poglavlje 2

Korištene tehnologije

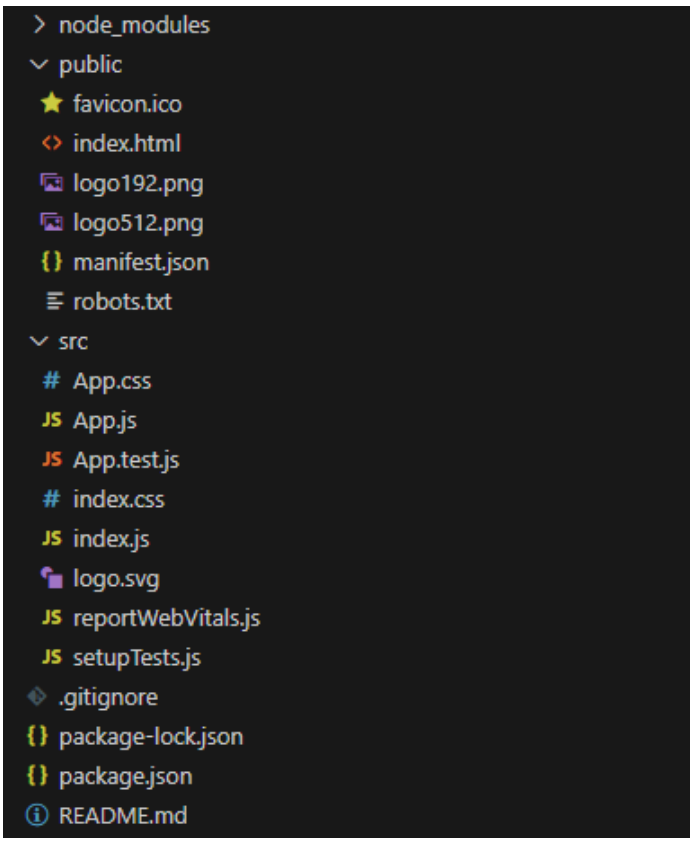
2.1 Opis korištenih tehnologija

2.1.1 React

React [4] je JavaScript knjižnica ponajviše korištena za izgradnju korisničkih sučelja. Glavna značajka ove knjižnice je mogućnost automatskog prikazivanja komponenti prilikom izmjena podataka. Ovo implicira da nije nužno ponovno pokretati program pri svakoj promjeni u kodu, već se promjene primjenjuju prilikom spremanja. Ovakav pristup programiranju omogućava izuzetno visoku efikasnost i produktivnost, budući da se vrijeme ne gubi na neprekidno pokretanje programa.

Inicijalizacija projekta se realizira komandom *"npx create-react-app aplikacija"*, pri čemu *aplikacija* predstavlja naziv projekta. Također je bitno napomenuti da *npx* komanda je dostupna jedino ako je na računalu instaliran Node.js [8] verzije 5.2 ili više te je definirana apsolutna putanja do njegove mape u varijablama okruženja. Sadržaj mape početnog React projekta vidljiv je u *Kodnom isječku* 2.1. Inicijalizacijom projekta stvara se struktura direktorija koja uključuje: *node-modules* gdje se lokalno spremaju sve Node.js knjižnice potrebne za ispravan rad aplikacije, *public* mapa koja sadrži datoteke od kojih kreće pregled na internet pregledniku, *src* mapa koji sarži cijelu logiku aplikacije, poput izmjene stranica i prikaza podataka, *package.json* koji sadrži sve knjižnice iz *node-modules* mape i naposljetku *README* datoteka korisna

za opis funkcionalnosti aplikacije.



Kodni isječak 2.1 - Prikaz direktorija inicijalnog React projekta

Daljnji razvoj aplikacije se ostvaruje dodavanjem i izmjenom datoteka i podataka unutar *src* mape. Za postizanje uspješnog upravljanja prijelazima između različitih stranica, potrebno je stvoriti posebnu datoteku koja će djelovati kao *router*. Ova datoteka ima ulogu obrade zahtjeva za promjenom stranica te će kao rezultat takvih zahtjeva odabrati odgovarajuću datoteku koja će poslužiti za prikazivanje nove stranice.

Uz to, potrebno je napraviti datoteku koja će služiti kao *context provider*. Njena osnovna zadaća je dohvaćanje specifičnih informacija prije samog prikaza stranica te obavljanje određenih akcija temeljem tih prikupljenih podataka. U aplikacijama s mogućnosti prijave i registracije korisnika, *context provider* služi za provjeru postojanja tokena u lokalnoj pohrani, odnosno je li korisnik koji pokušava pristupiti

aplikaciji autentificiran ili ne. Ukoliko tokena nema, vrši se redirekcija na početnu stranicu.

2.1.2 Laravel

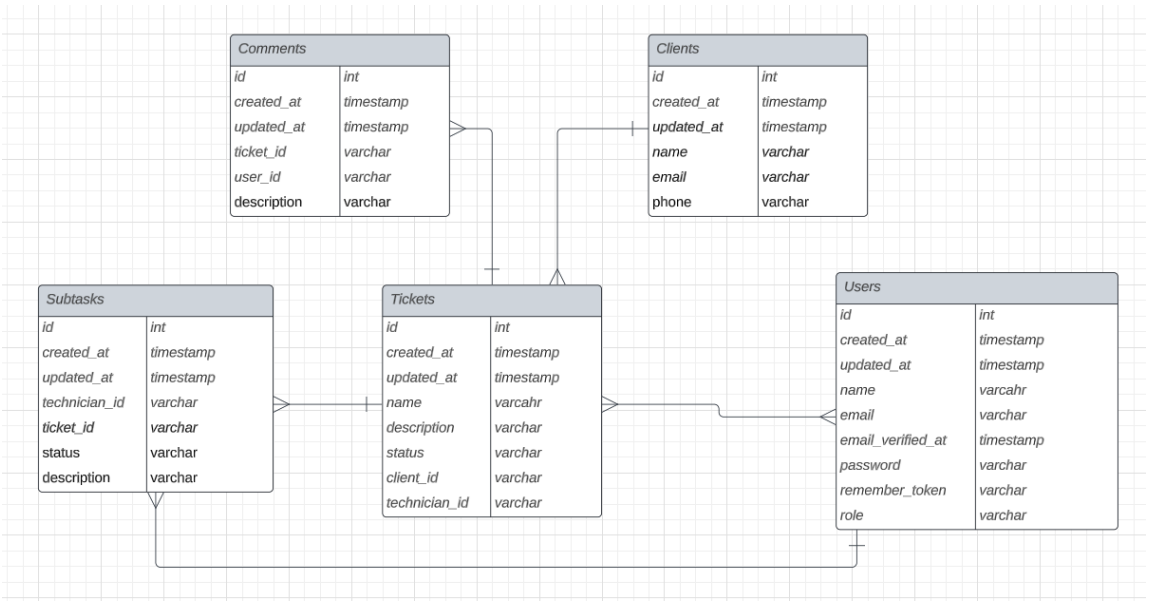
Laravel [6] je programski okvir baziran na PHP-u, skriptnom jeziku opće namjene široko korištenom i otvorenog koda koji je posebno prikladan za web razvoj. Laravel koristi postojeće komponente različitih programskih okvira što pomaže u izradi i razvoju web aplikacija, čineći proces organiziranijim i učinkovitijim.

Struktura projekta u Laravelu se oslanja na tri glavne komponente: upravljači (eng. controller), modeli (eng. model) i pogledi (eng. view). Upravljači imaju ključnu ulogu u manipulaciji podacima iz baze. Oni omogućuju definiranje različitih operacija koje se mogu izvoditi nad podacima. Modeli predstavljaju strukturu podataka u bazi. Svaki model odgovara jednoj tablici u bazi podataka, a omogućuju pristup i manipulaciju tim podacima. Pogledi su komponente korištene za prikazivanje HTML sadržaja te omogućuju razdvajanje prezentacijskog sloja od logike aplikacije. Pošto je ova aplikacija implementirana kao RESTful, izbjegava se korištenje pogleda iz razloga što se Laravel koristi iskjučivo za razvoj serverske strane aplikacije. Nadalje, bitna značajka koju Laravel uključuje su rute koje se koriste za preusmjeravanje zahtjeva na odgovarajuće upravljačke funkcije. Ovisno o putanji definiranoj putem HTTP protokola, Laravel će odabrati odgovarajuću upravljačku klasu za obradu zahtjeva.

Laravel je moguće koristiti za razvoj cjelokupne aplikacije korištenjem *blade.php* datoteka za prikaz stranica , ali također može biti korišten isključivo za serversku stranu aplikacije koja uspostavlja komunikaciju s klijentskom stranom putem HTTP protokola. Za kreiranje Laravel projekata se koristi Composer [9], alat za upravljanje ovisnostima u PHP-u pomoću kojeg se kroz terminal izrazito jednostavno može inicijalizirati novi projekt. Unutar projekta pruža se niz opcija korištenjem Artisan [10] sučelja. Artisan omogućuje izradu novih klasa, izvođenje migracija tablica u bazu podataka, punjenje tablica s testnim podacima, pokretanje aplikacije lokalno na određenom portu te obavljanje mnogih drugih radnji.

2.1.3 MySQL

MySQL [7] je sustav za upravljanje relacijskim bazama podataka koji je razvio Oracle i koji se temelji na strukturiranom jeziku upita (SQL). Otvorenog je koda što znači da svatko može koristiti i mijenjati softver. Također ga svatko može preuzeti s interneta i koristiti ga bez plaćanja. Ukoliko je potrebno, moguće je raditi izmjene nad izvornim kodom za bolje prilagođavanje potrebama pojedinca. Glavna značajka MySQL sustava je sposobnost uspostavljanja relacija između entiteta, što omogućuje efikasno povezivanje i organizaciju podataka unutar baze. Na taj se način korisnicima omogućuje intuitivno modeliranje složenih odnosa među podacima. Primjer ER dijagrama za ovu aplikaciju na kojoj se mogu vidjeti relacije među entitetima je vidljiv na *slici* 2.1.



Slika 2.1 ER dijagram opisanog sustava

2.1.4 Bootstrap

Bootstrap [1] je besplatni CSS okvir otvorenog koda usmjeren na kreiranje responzivnih komponenti na klijentskoj strani. Sadrži predloške dizajna temeljene na HTML-u, CSS-u i JavaScriptu te se u pojedine komponente stranice implementira kroz klasne nazive. Glavna prednost korištenja Bootstrapa nad ručnim pisanjem CSS koda je mogućnost puno bržeg i efikasijeg stiliziranja stranica.

Implementacija Bootstrapa u projekt je moguća dohvaćanjem paketa kroz terminal ili definiranjem poveznice s internet preglednika na kojoj se nalazi kompletan Bootstrap paket. Odabir metode ovisi o tehnologijama koje se primjenjuju u projektu.

Za primjer se uspoređuje razlika obrasca koji se sastoji isključivo od HTML koda i onog koji dodatno implementira Bootstrap. Na *slici* 2.2. vidljiv je obrazac koji je izrađen samo pomoću HTML koda, bez dodatnih stilskih prilagodbi.

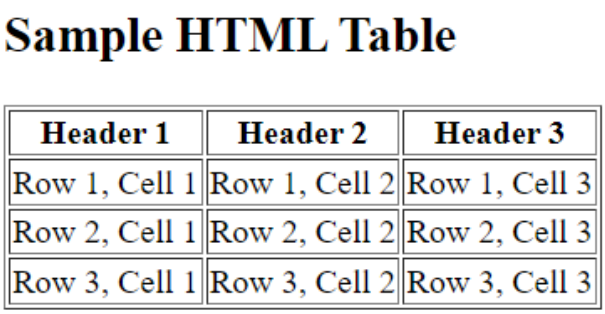
Slika 2.2 Jednostavni HTML obrazac

Na *slici* 2.3 prikazan je jednaki obrazac, ali koji implementira Bootstrap za stliziranje pojedinih komponenti.

Slika 2.3 HTML obrazac koja implementira Bootstrap

Razlika među navedenim obrascima je prilično značajna, što jasno ilustrira kako jednostavna stilizacija putem Bootstrapa može značajno transformirati izgled.

Kao dodatni primjer, jasno se uočava stilistička razlika između tablice koja je kreirana isključivo pomoću HTML-a i one koja je implementirana putem Bootstrapa. Na *slici* 2.4 je vidljiva tablica pisana isključivo HTML kodom.



Slika 2.4 Jednostavna HTML tablica

Slika 2.5 reprezenitra kako tablica izgleda uz vrlo jednostavnu implementaciju Bootstrapa. Kao što je vidljivo, dizajn tablice izgleda puno modernije i ugodnije oku u odnosu na tablicu sa *slike* 2.4.

Sample Table with Bootstrap

Header 1	Header 2	Header 3
Row 1, Cell 1	Row 1, Cell 2	Row 1, Cell 3
Row 2, Cell 1	Row 2, Cell 2	Row 2, Cell 3
Row 3, Cell 1	Row 3, Cell 2	Row 3, Cell 3

Slika 2.5 HTML tablica koja implementira Bootstrap

Za prethodni primjer biti će istaknuta razlika u kodu korištenom za generiranje tablica. U *Kodnom isječku* 2.2 prikazan je kod korišten za izradu generičke tablice korištenjem isključivo HTML-a.

```
<table border="1">
    <tr>
        <th>Header 1</th>
        <th>Header 2</th>
        <th>Header 3</th>
    </tr>
    <tr>
        <td>Row 1, Cell 1</td>
        <td>Row 1, Cell 2</td>
        <td>Row 1, Cell 3</td>
    </tr>
    <tr>
        <td>Row 2, Cell 1</td>
        <td>Row 2, Cell 2</td>
        <td>Row 2, Cell 3</td>
    </tr>
    <tr>
        <td>Row 3, Cell 1</td>
        <td>Row 3, Cell 2</td>
        <td>Row 3, Cell 3</td>
    </tr>
</table>
```

Kodni isječak 2.2 - Kod bazične HTML tablice

U *Kodnom isječku* 2.3 prikazan je programski kod za dinamičko stvaranje HTML tablice koja je dodatno stilizirana pomoću Bootstrapa.

```
<table class="table table-bordered">
    <thead class="thead-dark">
        <tr>
            <th>Header 1</th>
            <th>Header 2</th>
            <th>Header 3</th>
        </tr>
    </thead>
    <tbody>
        <tr class="table-info">
            <td>Row 1, Cell 1</td>
            <td>Row 1, Cell 2</td>
            <td>Row 1, Cell 3</td>
        </tr>
        <tr class="table-warning">
            <td>Row 2, Cell 1</td>
            <td>Row 2, Cell 2</td>
            <td>Row 2, Cell 3</td>
        </tr>
        <tr class="table-secondary">
            <td>Row 3, Cell 1</td>
            <td>Row 3, Cell 2</td>
            <td>Row 3, Cell 3</td>
        </tr>
    </tbody>
</table>
```

Kodni isječak 2.3 - Kod HTML tablice s Bootstrap stilizacijom

Kao što je vidljivo, razlika između dva prikazana kodna isječka nije značajna. Međutim, prilikom generiranja tablica u internet pregledniku, primjećuje se značajna razlika. Osnovni kod je isti, s identičnim rasporedom podataka u ćelijama. Jedina varijacija je u klasama koje se primjenjuju na određene tablične elemente u *Kodnom isječku* 2.3. Ovo ističe ključnu prednost Bootstrapa da prilikom minimalnih izmjena u kodu, moguće je postići značajne promjene u dizajnu.

2.2 Primjeri tehnologija

Da bi se jasnije razumjela funkcionalnost React knjižnice i Laravel programskog okvira, u narednom tekstu će biti detaljno opisana funkcionalnost obrasca koja implementira navedene tehnologije. Točnije, na koji se način šalju podaci iz obrasca

s klijentske strane prema serverskoj strani. Također, kako se vrši validacija i manipulacija podacima na serverskoj strani. Cijeli slijed izvršavanja će biti potkrijepljen kodnim isječcima za jasnije razumijevanje.

Zahtjev za slanje započinje s klijentske strane, gdje se koristi jednostavan obrazac za prikupljanje podataka. Nakon unosa podataka, korisnik pokreće proces slanja pritiskom na odgovarajući gumb. U procesu slanja se radi grupacija podataka nakon koje se inicira POST zahtjev prema serverskoj strani aplikacije, kao što je prikazano u *Kodnom isječku* 2.4.

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

  const formData = new FormData();
  formData.append('name', name);
  formData.append('email', email);

  try {
    const response = await fetch('http://localhost:8000/api/submit-form', {
      method: 'POST',
      body: formData,
    });

    if (response.ok) {
      const data = await response.json();
      setMessage(data.message);
    } else {
      console.error('Form submission failed');
    }
  } catch (error) {
    console.error('Error submitting form:', error);
  }
};
```

Kodni isječak 2.4 - Funkcija unutar Reacta za slanje podataka prema serverskoj strani

Podaci se u JSON formatu preko HTTP protokola šalju na serversku stranu aplikacije gdje se vrši daljnja validacija podataka. U trenutnom primjeru se očekuje da oba parametra budu ispunjena, odnosno da nisu prazni te da njihova duljina ne prekoračuje 255 znakova, kao što je vidljivo u *Kodnom isječku* 2.5.

```php
public function submitForm(Request $request)
{
    $this->validate($request, [
        'name' => 'required|string|max:255',
        'email' => 'required|email|max:255',
    ]);

    $form = new Form();
    $form->name = $request->input('name');
    $form->email = $request->input('email');

    $form->save();

    return response()->json(['message' => 'Form submitted successfully']);
}
```

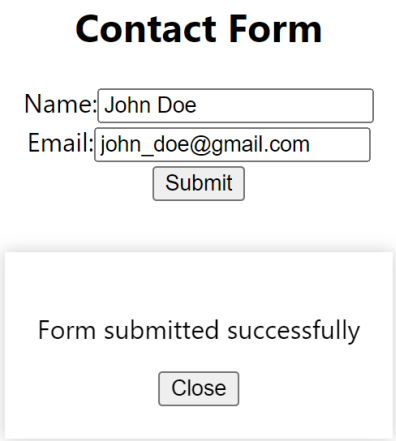
Kodni isječak 2.5 - Funkcija unutar Laravela koja prima i obrađuje podatke

Sljedeći korak je kreiranje instance klase koja će se koristiti za pohranu podataka u bazu. Uvjet za uspješno spremanje podataka je korištenje valjane klase koja se nalazi unutar *Models* mape čiji se parametri poklapaju s onima definiranim u bazi. Primjer spremljenih podataka u bazi je vidlji na *slici* 2.6. Završni korak je slanje poruke prema klijentskoj strani da su podaci uspješno spremljeni.

id	created_at	updated_at	name	email
2	2023-08-12 10:39:23	2023-08-12 10:39:23	John Doe	john_doe@gmail.com

Slika 2.6 Izgled baze podataka nakon izvršenog spremanja

Primitkom odgovora na klijentskoj strani, odrađuje se selekcija podataka među kojima se nalazi poruka o uspješnom spremanju u bazu podataka. U trenutnom primjeru je napravljena implementacija da se poruka prikaže na grafičkom korisničkom sučelju, kao što je i vidljivo na *slici* 2.7.



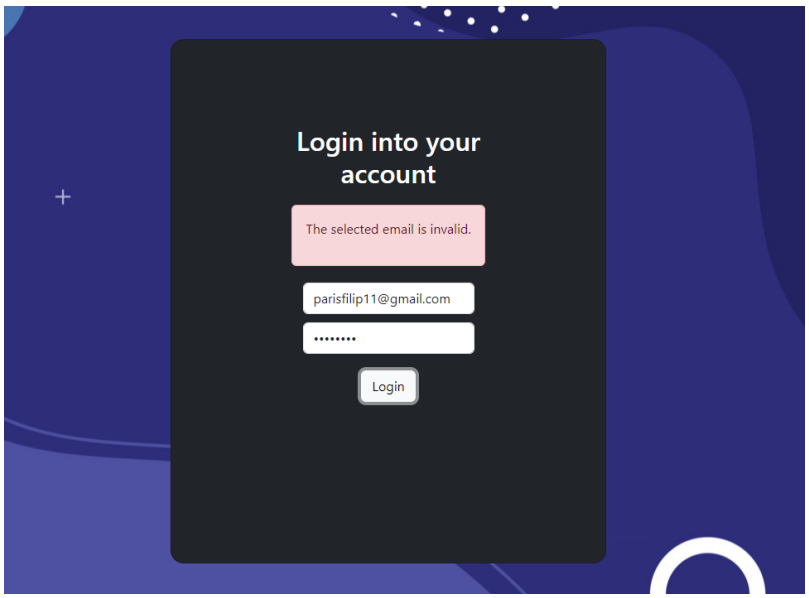
Slika 2.7 Prikaz poruke o uspješnom spremanju podataka

Poglavlje 3

Opis aplikacije

3.1 Prijava u sustav

Proces prijave u sustav odvija se putem korisničkog sučelja koje sadrži obrazac s poljima za unos e-mail adrese i lozinke. U situaciji kada uneseni podaci ne odgovaraju onima pohranjenima u bazi podataka, korisnik će vidjeti obavijest o grešci, kao što se može vidjeti na primjeru na *slici* 3.1.



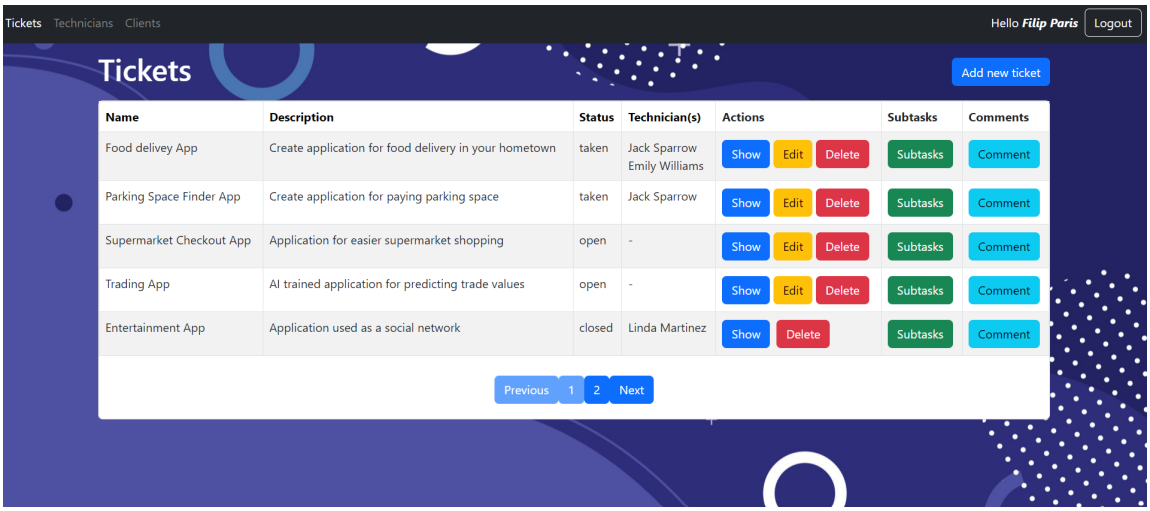
Slika 3.1 Poruka na obrascu za prijavu u slučaju neispravnih podataka

Nakon unosa valjanih podataka, na serverskoj strani generira se token koji se sprema u bazu podataka. Generirani token, zajedno s korisničkim podacima, šalje se kao odgovor natrag na klijentsku stranu. Na klijentskoj strani se token sprema u lokalnu pohranu kako bi se osigurala kontinuirana sesija korisnika i spriječilo slučajno izbacivanje iz sustava, što omogućuje da korisnik ostane prijavljen bez potrebe za ponovnom prijavom.

3.2 Prikaz liste zahtjeva

3.2.1 Prikaz liste zahtjeva s ulogom *admin*

Uspješnom prijavom u sustav, korisnika se preusmjeruje na *Tickets* stranicu. Ukoliko je uloga korisnika *admin*, biti će mu prikazani svi zahtjevi iz baze podataka. Na slici 3.2 prikazana je stranica *Tickets* za korisnike s administratorskom ulogom.



Slika 3.2 Prikaz liste zahtjeva za *admin* korisnika

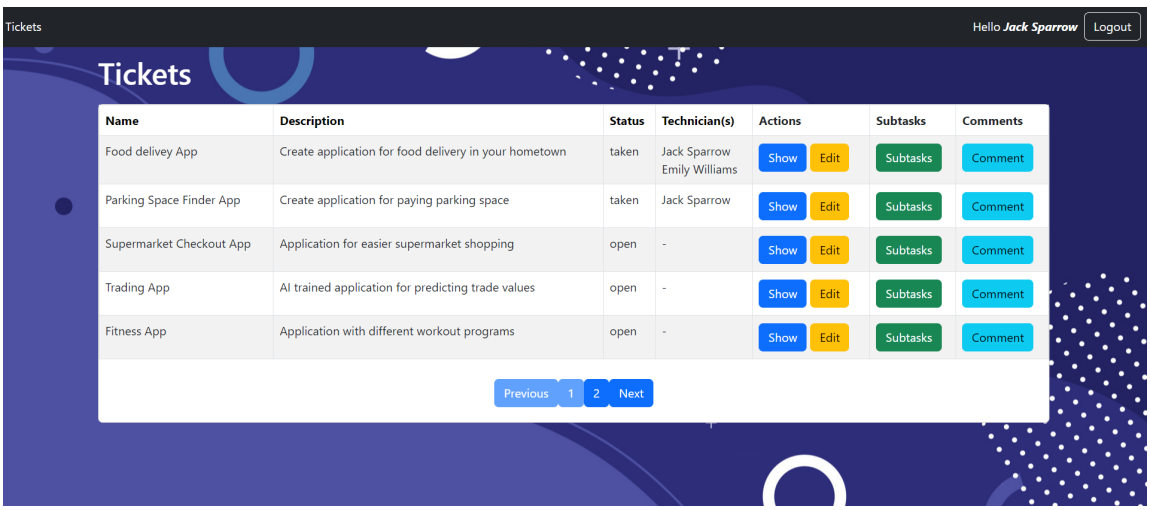
Svaki zahtjev se nalazi u zasebnom retku tablice u kojem su prikazani njegovi podaci, prateće akcije i dodatne funkcionalnosti. Podaci uključuju naziv zahtjeva, opis, trenutni status te tehničare koji su odgovorni za njega. Nadalje, dostupne su opcije za detaljan prikaz, uređivanje i brisanje svakog pojedinačnog zahtjeva.

Također je moguće dodati komentare na svaki zahtjev kada je potrebno, kao i stvoriti podzahtjeve kako bi se zahtjev razložio na specifične faze ili korake potrebne za rješavanje. Naposljetku, postoji mogućnost kreiranja novog zahtjeva.

3.2.2 Prikaz liste zahtjeva s ulogom *technician*

U slučaju da je uloga korisnika *technician*, imati će pristup zahtjevima koji su mu dodijeljeni i onima koji su još uvijek otvoreni i nisu dodijeljeni nikome, kako je prikazano na *slici* 3.3. Uz to, tehničaru nije dopušteno brisanje zahtjeva.

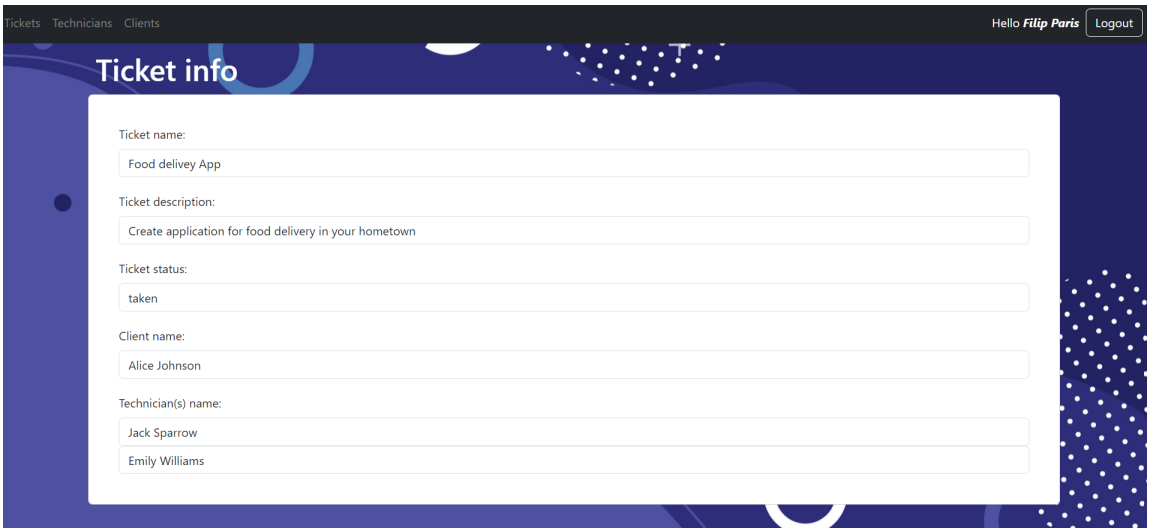
Na navigacijskoj traci navedene slike ne postoje poveznice na stranice gdje se nalaze lista tehničara i klijenata. Razlog navedenog ograničenja ovlasti proizlazi iz pokušaja usmjeravanja tehničara isključivo na rješavanje problema zbog kojeg je zahtjev kreiran. Sve ostale zadatke, uključujući upravljanje tehničarima i klijentima, prepušta se korisnicima s *admin* ovlastima.



Slika 3.3 Prikaz liste zahtjeva za *technician* korisnika

3.3 Prikaz pojedinačnog zahtjeva

Kao što se može primijetiti na *slikama* 3.2 i 3.3, svaki zahtjev ima pridruženi gumb *Show* smješten u stupcu *Actions*. Klikom na navedeni gumb korisnika se preusmjeruje na novu stranicu na kojoj su detaljnije prikazani podaci zahtjeva. Primjer navedene stranice je vidljiv na *slici* 3.4. Svi pojedinačni podaci zahtjeva jasno su organizirani u redovima, što olakšava izdvajanje željenih informacija.



Slika 3.4 Detaljni prikaz pojedinačnog zahtjeva

Uspoređujući detaljni prikaz pojedinačnog zahtjeva s prikazom svih zahtjeva na *slici* 3.2, vidljivo je da detaljni prikaz obuhvaća sve informacije koje se odnose na pojedini zahtjev. To se ponajviše odnosi na prikaz klijenta za kojeg je zahtjev kreiran, koji nije vidljiv u tablici na *slici* 3.2. Razlog za ovakvu implementaciju je izbjegavanje stvaranja prevelike tablice za prikaz svih zahtjeva, čime se nastoji očuvati estetika i preglednost sučelja.

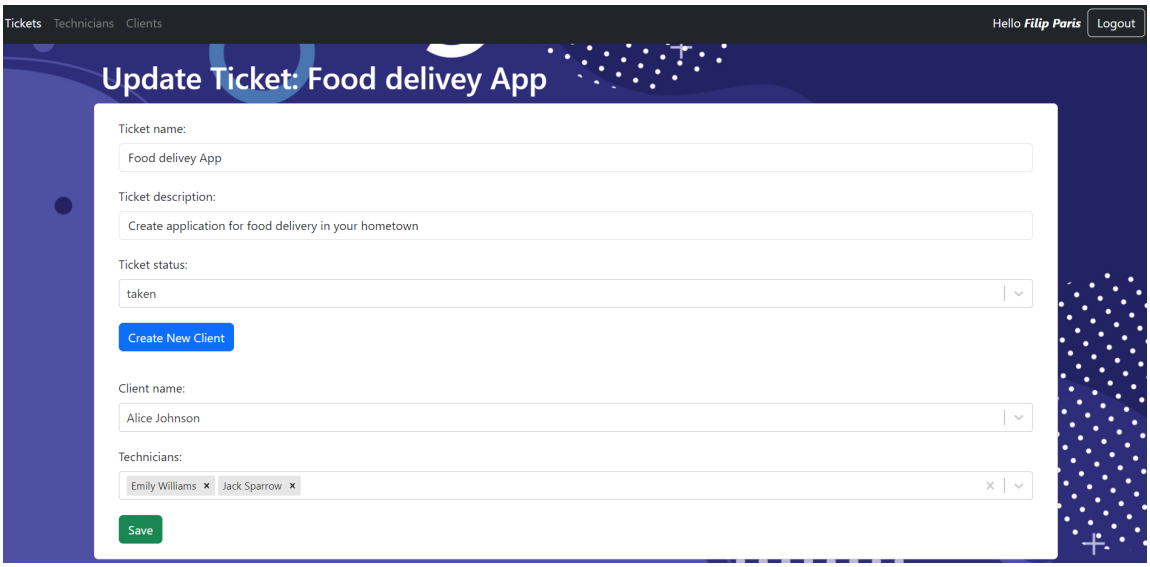
3.4 Izmjena pojedinačnog zahtjeva

Nadovezujući se na odjeljak za prikaz liste zahtjeva, na *slici* 3.2, pod stupcem *Actions* se također nalazi gumb naziva *Edit*. Klikom na navedeni gumb, izvršava se

preusmjeravanje na stranicu za izmjenu podataka zahtjeva.

3.4.1 Izmjena pojedinačnog zahtjeva s ulogom *admin*

Admin korisnik ima ovlasti za uređivanje svih pojedinačnih informacija vezanih uz zahtjev. To uključuje promjenu naziva zahtjeva, opisa, statusa, klijenta na kojeg je zahtjev povezan i listu tehničara odgovornih za njegovo rješavanje. Polje za odabir statusa, klijenta i tehničara koristi isti element sučelja, s time da je samo polje za odabir tehničara konfigurirano kao *multi-select*, omogućavajući odabir više vrijednosti unutar padajućeg izbornika. Kao rezultat toga, navedena implementacija omogućuje dodjeljivanje više od jednog tehničara na svaki zahtjev, ukoliko je to potrebno. Prikaz stranice za izmjenu podataka zahtjeva, ukoliko je uloga korisnika *admin*, vidljiv je na *slici* 3.5.



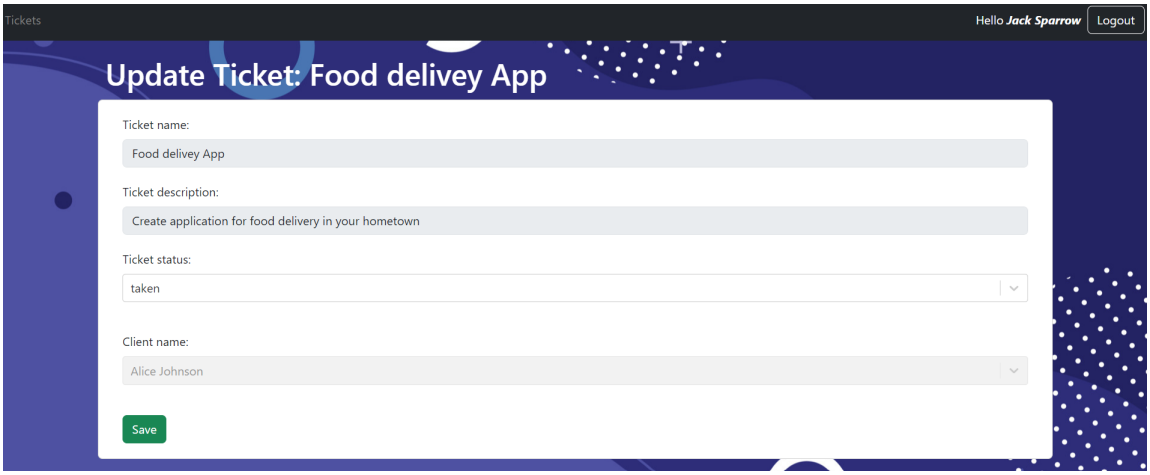
Slika 3.5 Izmjena zahtjeva s *admin* ulogom

Ukoliko se status zahtjeva postavi na *open*, do tada odabrani tehničari će se obrisati te će se prema serverskoj strani poslati podaci o zahtjevu s praznim dijelom za tehničare. Nadalje, postavljanjem statusa na *closed*, na stranici za prikaz svih zahtjeva će se za zahtjev s navedenim statusom ukloniti opcija za izmjenu te će mu

se onemogućiti prikaz podzahtjeva klikom na gumb *Subtasks*.

3.4.2 Izmjena pojedinačnog zahtjeva s ulogom *technician*

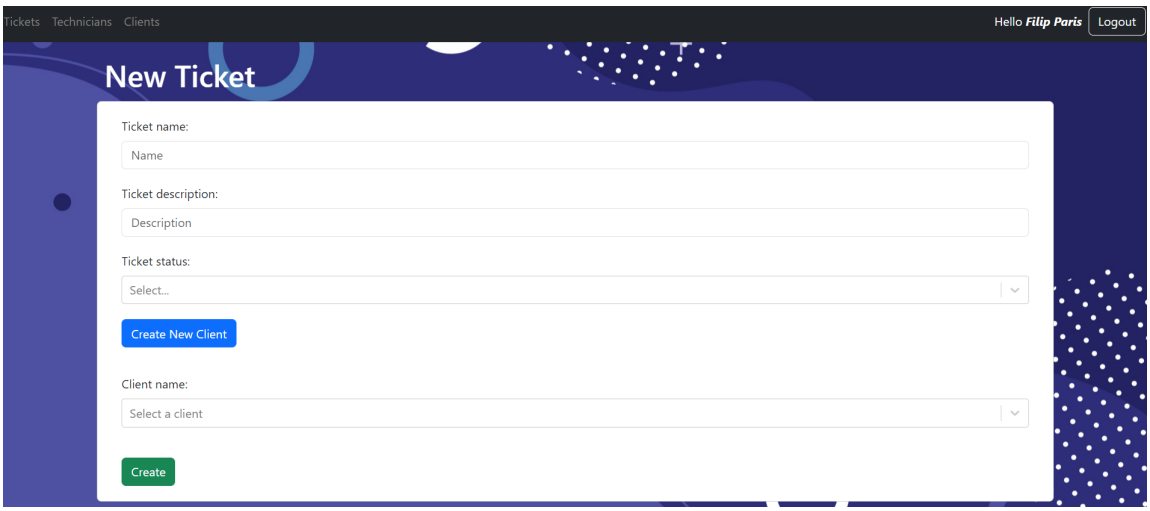
Na *slici* 3.6. jasno su prikazana ograničenja koja tehničar ima prilikom izmjene podataka o zahtjevu. Polja za ime i opis zahtjeva su postavljeni kao *read-only*, odnosno samo za čitanje. Isto je napravljeno i za odabir klijenta. Njemu je omogućena jedino promjena statusa zahtjeva. Ukoliko je trenutni status zahtjeva *open*, tehničar može promijeniti status u *taken* te će se slanjem podataka prema serverskoj strani njegov identifikacijski broj postaviti u polje *technicianId* unutar *ticket* objekta. Na zahtjevu na koji je on zadužen može promijeniti status u *closed*, čime će se zahtjev zatvoriti i daljnje izmjene neće biti moguće.



Slika 3.6 Izmjena zahtjeva s *technician* ulogom

3.5 Kreiranje novog zahtjeva

Klikom gumba *Add new ticket* na stranici za prikaz liste svih zahtjeva vidljivoj na *slici* 3.2, izvršava se preusmjeravanje na stranicu koja sadrži obrazac za kreiranje novog zahtjeva, kako je prikazano na slici 3.7.

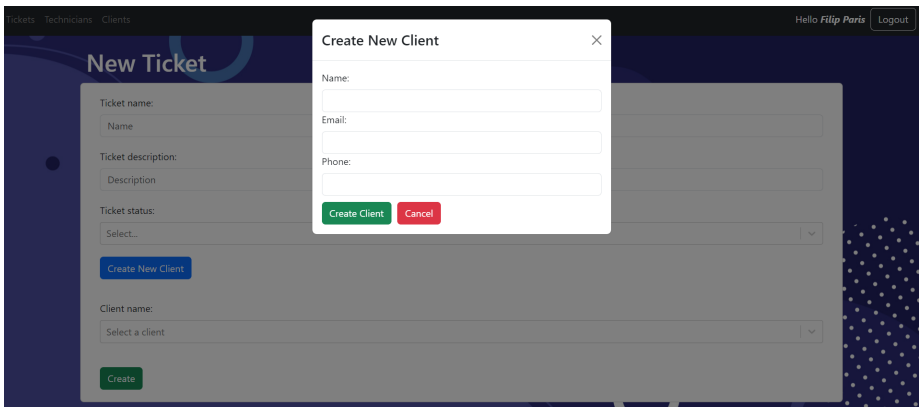


Slika 3.7 Kreiranje novog zahtjeva

Za upis imena i opisa zahtjeva koristi se najobičnije tekstualno polje. Prilikom kreiranja novog zahtjeva, može se postaviti status *open* i *taken*. Ukoliko se odabere status *taken*, u obrascu će se prikazati mogućnost dodavanja tehničara. Odabirom statusa *open*, mogućnost dodavanja tehničara neće biti vidljiva.

Kao što je vidljivo na *slici* 3.7, moguće je unutar obrasca kreirati novog klijenta. Navedena funkcionalnost je dodana iz razloga što se može desiti da se želi kreirati novi zahtjev, a ne postoje podaci o klijentu na kojeg se zahtjev otvara. Da bi se skratilo vrijeme presumjeravanja na prikaz klijenata, kreiranje novog te zatim povratak na kreiranje zahtjeva, implementirana je navedena funkcionalnost.

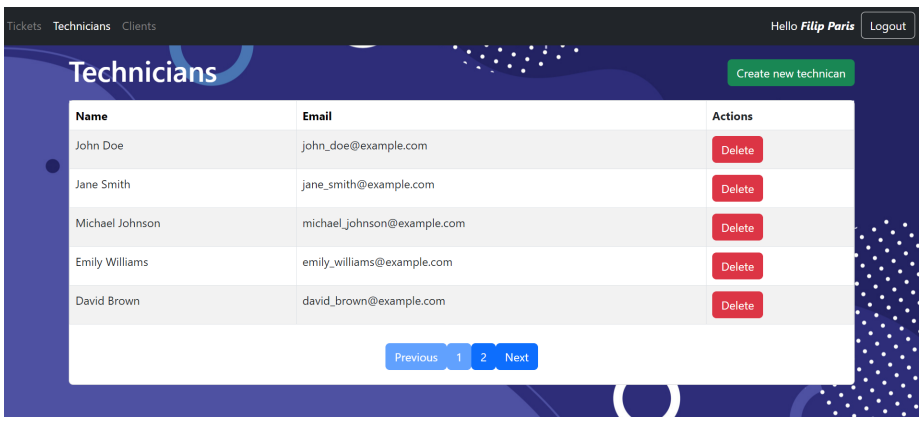
Klikom na gumb *Create New Client* otvara se skočni prozor u kojem se unose podaci o novom klijentu. Nakon uspješnog kreiranja, korisnik ostaje na istoj stranici za kreiranje novog zahtjeva. Primjer skočnog prozora za kreiranje novog klijenta može se vidjeti na *slici* 3.8.



Slika 3.8 Skočni prozor za kreiranje novog klijenta

3.6 Prikaz tehničara

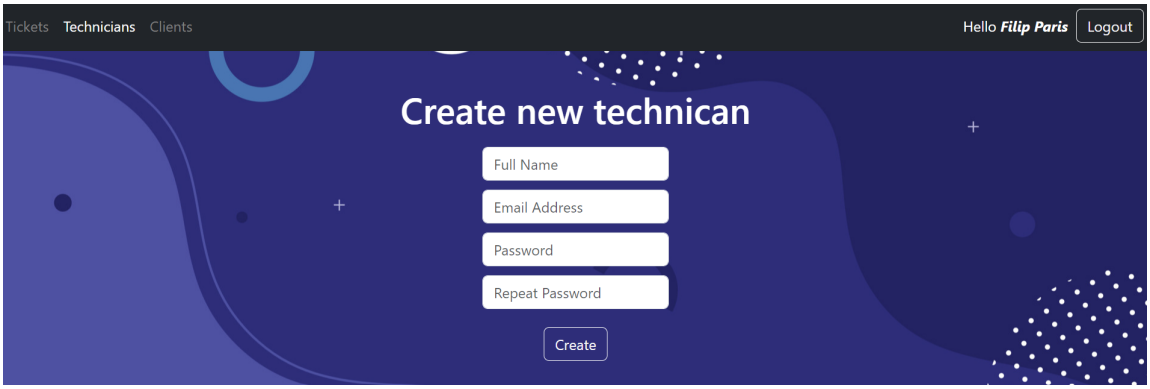
Odabirom poveznice *Technicians* u navigacijskoj traci, odrađuje se preusmjeravanje korisnika na prikaz liste svih tehničara trenutno prisutnih u sustavu. Izgled liste tehničara vidljiv je na *slici* 3.9. Važno je naglasiti da je navedena funkcionalnost dostupna jedino korisnicima s ulogom *admin* iz razloga što nad svakim tehničarom postoji mogućnost brisanja istog.



Slika 3.9 Prikaz svih tehničara

Kao što je navedeno, *admin* korisnik ima mogućnost brisanja svakog tehničara iz sustava, kao i kreiranje novog. Klikom na gumb *Create new technician* korisnika se preusmjerava na stranicu gdje je potrebno upisati podatke o novom tehničaru kojeg se želi dodati u sustav. Izgled navedene stranice prikazan je na *slici* 3.10.

U ovom se projektu ne koristi tradicionalan pristup prilikom kreiranja novih korisnika, što bi uključivalo da stranici za prijavu i registraciju može bilo tko pristupiti. Razlog tome je sprječavanje pristupa osoba koje nemaju odobrenje te je naposljetku moguć ulazak u sustav jedno s postojećim valjanim kredencijama.



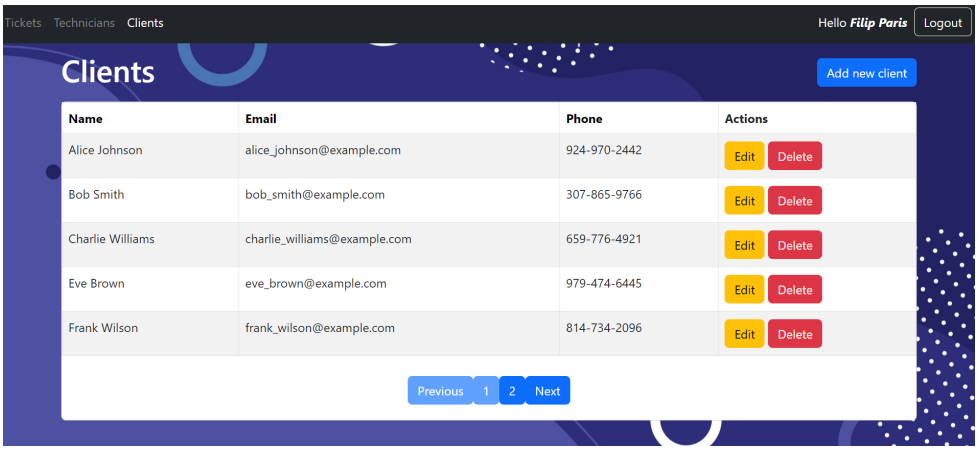
Slika 3.10 Kreiranje novog tehničara

3.7 Prikaz klijenata

Ukoliko je uloga korisnika *admin*, njemu će u navigacijskoj traci biti dostupna i poveznica pod nazivom *Clients*. Klikom na navedenu poveznicu, korisnika se preusmjerava na stranicu za prikaz svih klijenata unutar sustava, prikazanu na *slici* 3.11. Navedena radnja je onemogućena za korisnike s *technician* ulogom iz razloga što postoji mogućnost brisanja pojedinog klijenta koju može odrađivati samo korisnik s najvišim ovlastima, odnosno *admin* ulogom.

Ovdje je također implementirano straničenje te sortiranje podataka. Što se tiče straničenja, podaci se sa serverske strane dohvaćaju preko meta podataka u blokovima od pet objekata po stranici. Dalje se na klijentskoj strani navedeni meta

podaci koriste za implementaciju straničenja. Organizacija sortiranja je ostvarena putem postupka u kojem, po pritisku na ime stupca, zahtjev se upućuje serverskoj strani aplikacije sa *sortBy* parametrom, koji odgovara imenu dotičnog stupca. Istovremeno, *sortDirection* parametar se izmjenjuje nakon svakog klika na ime stupca. Logika sortiranja podataka je implementirana na serverskoj strani aplikacije kako bi se omogućilo ispravno funkcioniranje daljnjeg straničenja podataka.



Slika 3.11 Prikaz svih klijenata

3.8 Izmjena klijenata

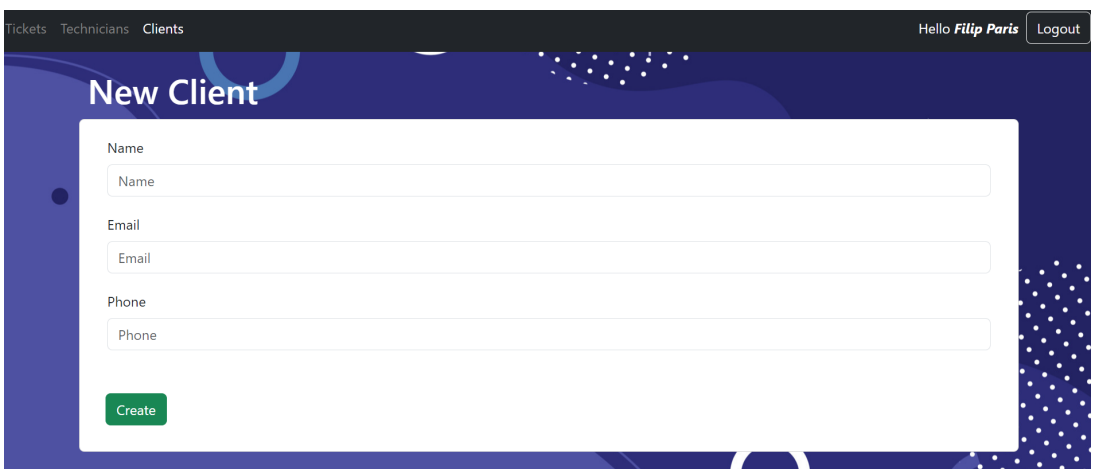
Nadovezujući se na prijašnje poglavlje, na *slici* 3.11 pod stupcem *Actions* se nalazi gumb naziva *Edit*. Klikom na navedeni gumb izvršava se preusmjeravanje korisnika na stranicu za izmjenu podataka klijenta, vidjlivu na *slici* 3.12. Klijentu se mogu izmjeniti ime, email adresa i broj telefona.

Izmjeni podataka klijenata ne mogu pristupiti korisnici s ulogom *technician* iz razloga što pri bilo kakvoj izmjeni podataka se mora izrazito paziti kako se ne bi narušila valjanost podataka. Očekuje se da je korisnik s ulogom *admin* najpouzdaniji za izmjenu tako osjetljivih podataka. Također, nastoji se izbjeći da korisnik s *technician* ulogom pristupi osobnim podacima pojedinog klijenta. Njima je omogućeno samo pregledavanje imena klijenata na pojedinim zahtjevima koji su im dodijeljeni ili koji imaju status *open*.

Slika 3.12 Izmjena podataka klijenata

3.9 Kreiranje klijenata

Na stranicu za kreiranje novog klijenta dolazi se klikom gumba *Add new client* na stranici za prikaz svih klijenata sustava. Izgled stranice za kreiranje klijenta je dostupan na *slici* 3.13. Upisom svih podataka i klikom na gumb *Create*, korisnika se presumjeri nazad na prikaz svih klijenata među kojima se nalazi i novokreirani.



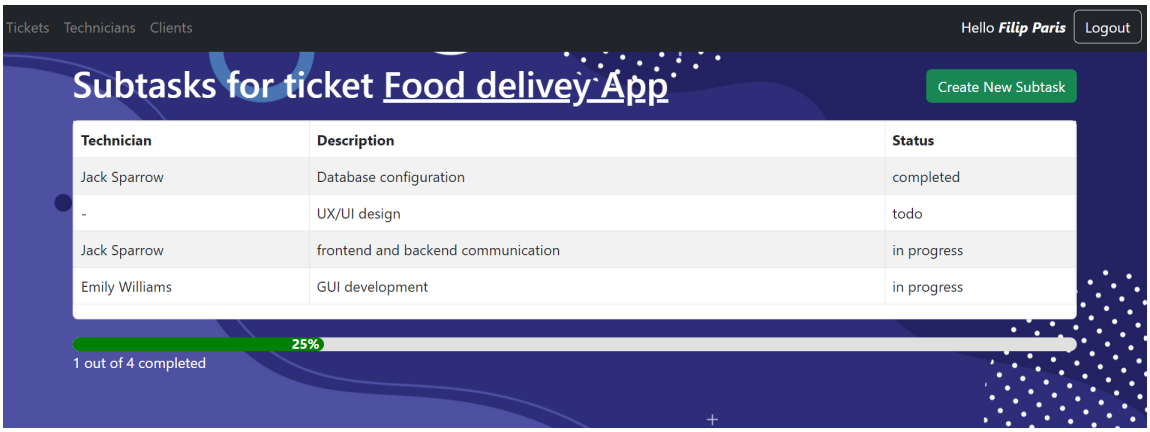
Slika 3.13 Obrazac za kreiranje novog klijenta

3.10 Prikaz podzahtjeva

Stranici za prikaz podzahtjeva pristupa se klikom gumba *Subtasks* koji se nalazi u stupcu *Subtasks* na prikazu svih zahtjeva, što je vidljivo na *slici* 3.2. Za pristup stranici s podzahtjevima, nužan je uvjet da status zahtjeva bude *open* ili *taken*. Ako je status *closed*, gumb za pristup će biti deaktiviran.

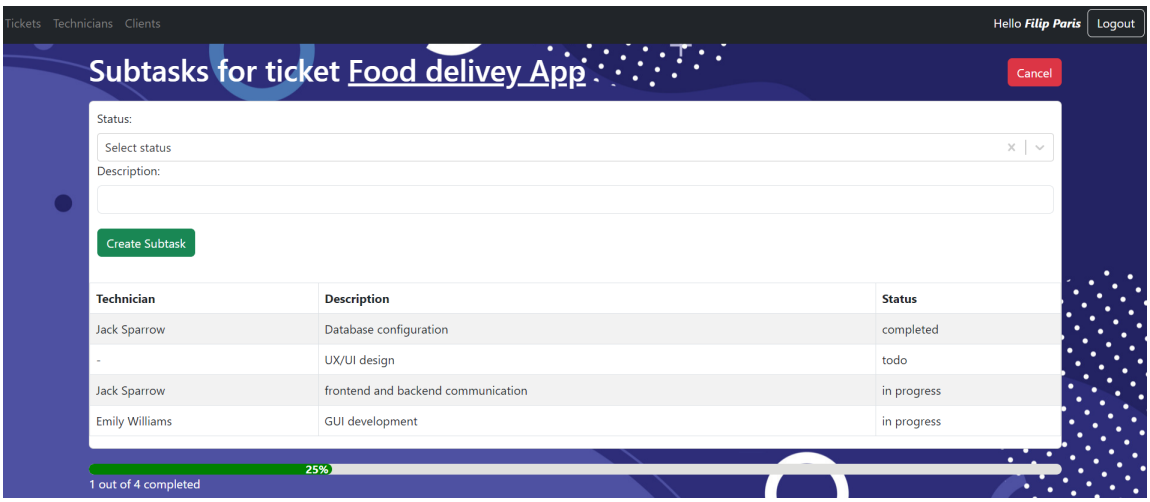
3.10.1 Prikaz podzahtjeva s ulogom *admin*

Prikaz podzahtjeva, vidljiv na *slici* 3.14, sastoji se od naziva zahtjeva radi jasnog identificiranja zahtjeva za koji se stvaraju podzahtjevi, gumba *Create New Subtask* za kreiranje novih podzahtjeva, liste postojećih podzahtjeva te naposljetku trake za napredak kojom se grafički pokazuje postotak izvršenih podzahtjeva.



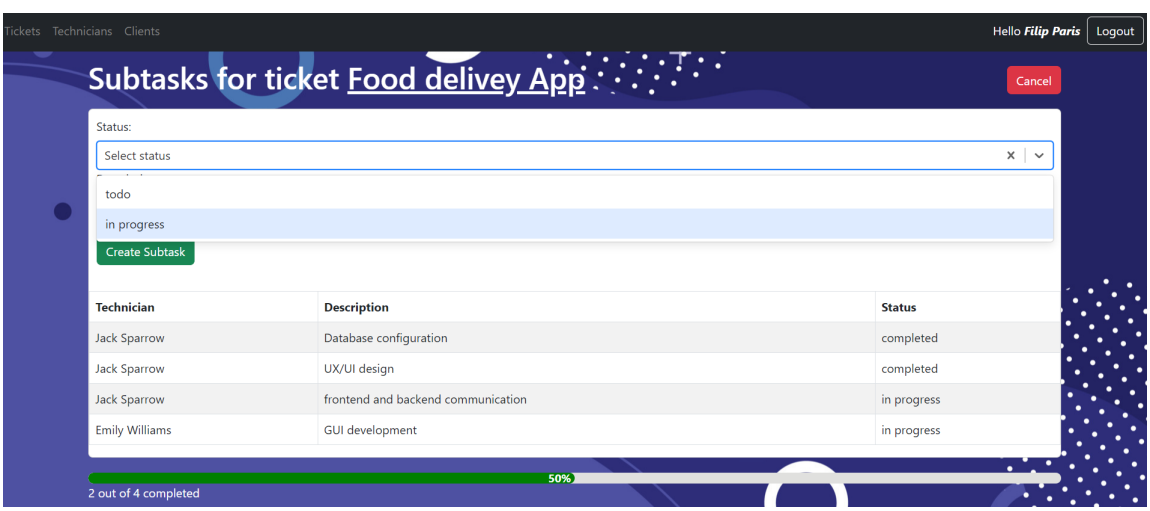
Slika 3.14 Prikaz liste podzahtjeva za korisnika s *admin* ulogom

Trenutna logika aplikacije ne omogućuje korisniku s ulogom *technician* da kreira nove podzahtjeve, već to može napraviti samo *admin* korisnik. Klikom na gumb *Create New Subtask*, otvara se obrazac na trenutnoj stranici s poljima za odabir statusa te opisom podzahtjeva. Izgled stranice s otvorenim obrascem prikazan je na *slici* 3.15.



Slika 3.15 Lista podzahtjeva s obrascem za kreiranje

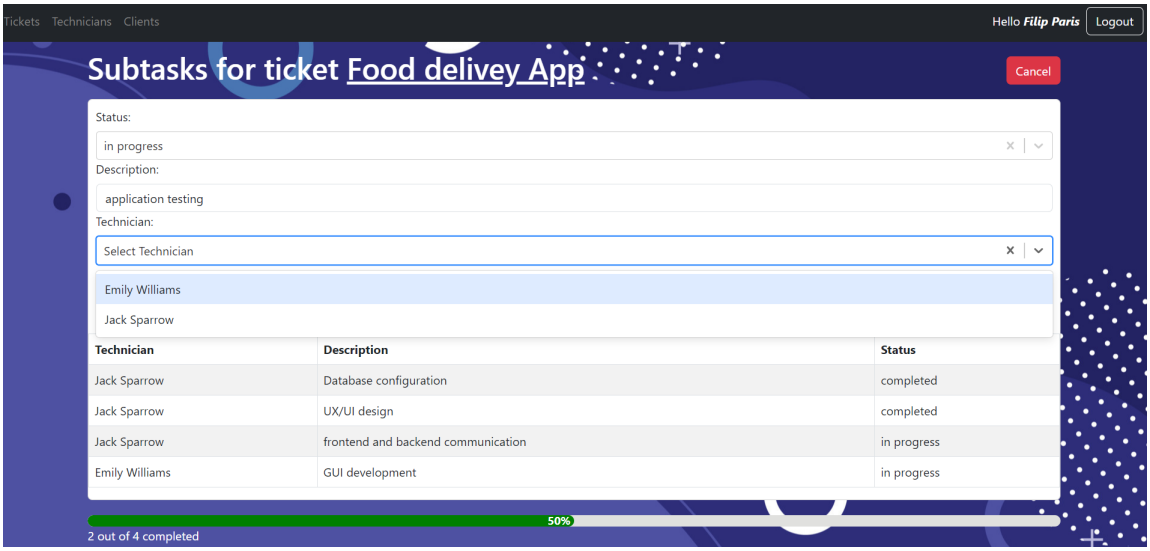
Za svaki podzahtjev je moguće odabrati dva statusa: *todo* i *in progress*, kao što je vidljivo na *slici* 3.16.



Slika 3.16 Lista podzahtjeva s obrascem za kreiranje i njegovim otvorenim sučeljem za odabir statusa

Ukoliko se kao status odabere *todo*, podatak u koji se upisuje tehničar će ostati prazan. Odabirom statusa *in progress*, potrebno je odabrati tehničara kako bi se

podaci uspješno spremili u bazu te kasnije prikazali na stranici. Također, važno je istaknuti da se lista tehničara izvlači iz elemenata *technicianId* zahtjeva za koji se stvaraju podzahtjevi. Ako je za rješavanje zahtjeva angažirano četiri tehničara, obrazac za stvaranje podzahtjeva će sadržavati izbornik s četiri tehničara. Primjer obrasca u kojem je potrebno odabrati tehničara je vidljiv na *slici* 3.17.



Slika 3.17 Lista podzahtjeva s obrascem za kreiranje i njegovim otvorenim sučeljem za odabir tehničara

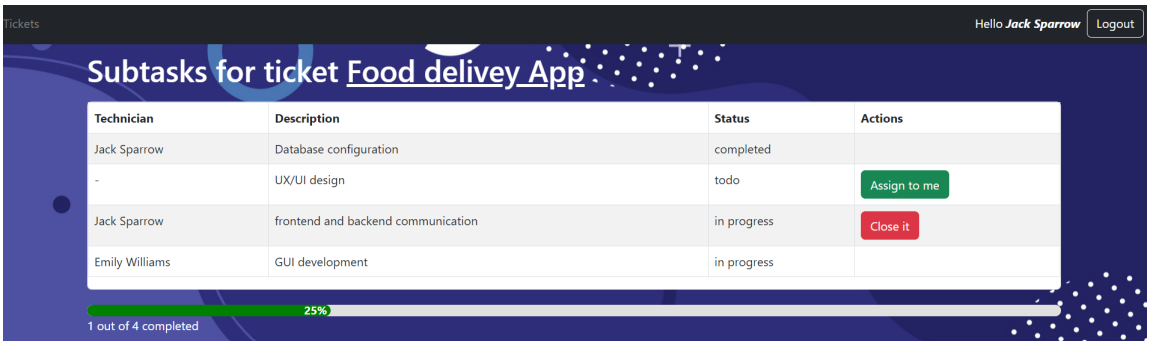
3.10.2 Prikaz podzahtjeva s ulogom *technician*

Technician korisnik na prikazu podzahtjeva nema ponuđeno mogućnost kreiranja novog, ali ima nadodani stupac naziva *Actions*, prikazano na *slici* 3.18.

Ukoliko je status podzahtjeva *todo*, tehničar može predodijeliti izvršavanje istog na sebe klikom na gumb *Assign to me*. Na taj način će se navedenom podzahtjevu izmjeniti status na *in progress* te će se u stupcu *Technicians* prikazati ime tehničara.

U slučaju da je status zahtjeva *in progress*, tehničar može kliknuti gumb *Close it* kojim se status podzahtjeva prebacuje na *closed* te se povećava vrijednost na traci za napredak. Preduvjet mogućnosti zatvaranja podzahtjeva je taj što prijavljeni

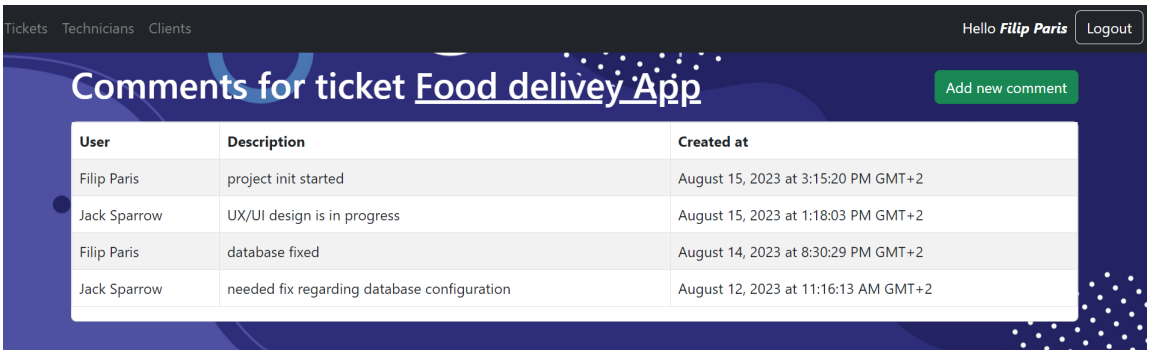
korisnik mora biti jednak onome na kojeg je podzahtjev dodijeljen.



Slika 3.18 Lista podzahtjeva s *technician* ulogom

3.11 Prikaz komentara

Na stranici za prikaz svih zahtjeva postoji gumb *Comment* kojim se preusmjerava na stranicu za prikaz komentara. Navedena stranica, prikazana na *slici* 3.19, sadrži sve komentare koji su napravljeni za određeni zahtjev, kao i mogućnost kreiranja novog komentara.

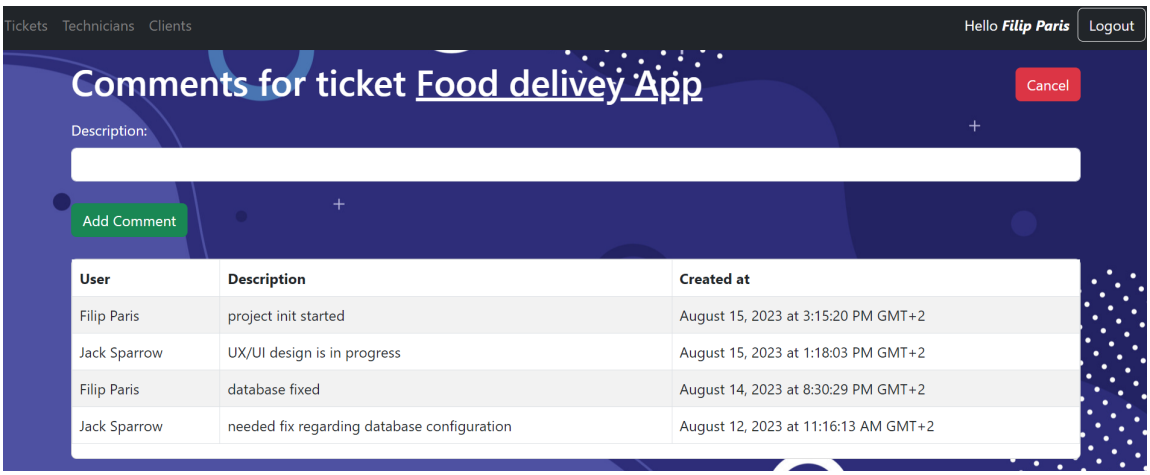


Slika 3.19 Prikaz komentara zahtjeva

Ova funkcionalnost je omogućena za sve korisnike sustava iz razloga što su komentari i tehničara i admina od iznimne važnosti kako bi se precizno pratila faza rješavanja zahtjeva te identificirali eventualni problemi.

Svi komentari se dohvaćaju sa serverske strane aplikacije i sortiraju prema datumu stvaranja, kako bi se osiguralo da najnoviji komentar bude prvi prikazan na listi. Time se omogućava da najrelevantniji komentar za zahtjev bude istaknut na početku.

Klikom na gumb *Add new comment*, na stranici se otvara forma koja sadrži polje za upis komentara. Slanjem ispunjenog obrasca, na stranici će se prikazati novonastali komentar. Prikaz stranice s otvorenim obrascem za kreiranje komentara nalazi se na *slici* 3.20.



Slika 3.20 Prikaz komentara s obrascem za kreiranje novog

Poglavlje 4

Opis funkcionalnosti aplikacije na razini koda

4.1 Kreiranje zahtjeva

4.1.1 Grupiranje podataka

Iniciranje procesa stvaranja novog zahtjeva započinje popunjavanjem podataka u poljima koja se nalaze unutar obrasca namijenjenog za njegovo stvaranje, kako je prikazano na *slici* 3.7. Početni segment obrasca očekuje unos naziva zahtjeva te njegove pripadajuće opisne informacije. Njihova implementacija prikazana je u *Kodnom isječku* 4.1.

Također, važno je naglasiti da se za postizanje funkcionalnosti stvaranja i uređivanja zahtjeva koristi isti obrazac, pri čemu se ovisno o odabranoj funkcionalnosti određene komponente ne prikazuju ili im se ne može pristupiti. Glavni razlog za ovakav pristup je sprječavanje nepotrebnog dupliciranja komponenata, što bi se dogodilo da su ovi obrasci razvijani u odvojenim datotekama. Međutim, kroz upotrebu zajedničkog obrasca za implementaciju funkcionalnosti stvaranja i uređivanja zahtjeva, postiže se značajno efikasnija optimizacija koda.

```
<Form onSubmit={onSubmit}>
  <Row>
    <Col>
      <Form.Group>
        <Form.Label>Ticket name:</Form.Label>
        <Form.Control
          value={ticket.name}
          onChange={ev => setTicket({ ...ticket, name: ev.target.value })}
          placeholder="Name"
          disabled={user.role === "tech"}
        />
      </Form.Group>
    </Col>
  </Row>
  <Row>
    <Col>
      <Form.Group controlId="ticketDescription">
        <Form.Label>Ticket description:</Form.Label>
        <Form.Control
          value={ticket.description}
          onChange={ev => setTicket({ ...ticket, description: ev.target.value })}
          placeholder="Description"
          disabled={user.role === "tech"}
        />
      </Form.Group>
    </Col>
  </Row>
```

Kodni isječak 4.1 - Implementacija polja za ime i opis zahtjeva

U prvoj liniji koda se prilikom klika gumba za slanje podataka poziva funkcija *onSubmit()*. Navedena funkcija će biti detaljnije objašnjena u sljedećem odjeljku. Nadalje, u kodu se nalazi polje koje služi za unos naziva zahtjeva. Vrijednost tog polja se čita iz elementa *name* sadržanom u objektu *ticket*. Svaka promjena u tom polju rezultira ažuriranjem vrijednosti elementa *name* u objektu *ticket*, koja se kasnije koristi pri slanju podataka serverskoj strani aplikacije. Ukoliko je element *name* u *ticket* objektu prazan, u polju će biti prikazan tekst *Name* za čiji je prikaz zaslužan atribut *placeholder*.

Polje za unos opisa zahtjeva slijedi iste principe kao i za unos njegovog naziva. Vrijednost navedenog polja se dohvaća iz elementa *description* unutar objekta *ticket*. Svaka promjena u tom polju rezultira ažuriranjem vrijednosti *description* elementa unutar objekta *ticket*.

Kao što je ranije objašnjeno, isti obrazac se koristi za stvaranje i izmjenu zahtjeva. Zbog ove zajedničke karakteristike, korisnici s ulogom *technician* nemaju mogućnost

uređivanja ovih dvaju polja. Ove stavke su im dostupne isključivo u svrhu pregleda prilikom izmjene zahtjeva.

Sljedeće polje je namijenjeno za odabir statusa i prikazano je u *Kodnom isječku* 4.2.

```
<Row>
  <Col>
    <Form.Group controlId="ticketStatus">
      <Form.Label>Ticket status:</Form.Label>
      <Select
        value={statusOptions.find(option => option.value === ticket.status)}
        options={statusOptions}
        onChange={selectedOption => {
          const selectedStatus = selectedOption.value;
          if (selectedStatus === 'open') {
            setTicket({ ...ticket, status: selectedStatus, technician_id: ['-'] });
          } else if (selectedStatus === 'taken') {
            if (user.role === 'admin') {
              setTicket({ ...ticket, status: selectedStatus, technician_id: '' });
            } else {
              setTicket({ ...ticket, status: selectedStatus, technician_id: [user.id] });
            }
          } else {
            setTicket({ ...ticket, status: selectedStatus });
          }
        }}
      />
    </Form.Group>
  </Col>
</Row>
```

Kodni isječak 4.2 - Implementacija polja za status

Funkcija *find()* se koristi za bilježenje vrijednosti polja, provjeravajući niz *statusOptions* koji sadrži riječi *open*, *taken* i *closed*. Ako se jedna od navedenih riječi nalazi u *status* elementu objekta *ticket*, taj će se podatak prikazati na korisničkom sučelju aplikacije. Također će se u padajućem izborniku prikazati sve ostale opcije za navedeno polje. Prilikom bilo kakve izmjene statusa, radi se dodatna provjera odabrane vrijednosti zbog postavljanja elementa *technicianId* unutar *ticket* objekta.

Ukoliko je status *open*, *technicianId* će se postaviti na "-" čime se simbolizira da niti jedan tehničar nije zadužen na zahtjevu. Ukoliko je status *taken*, radi se dodatna provjera o ulozi trenutno prijavljenog korisnika. Ako je korisnik *admin*, *technicianId* će se postaviti na prazan niz znakova iz razloga što *admin* korisnik može opredijeliti više tehničara na zahtjev. U slučaju da je prijavljeni korisnik *technician*, njegov identifikator će se automatski postaviti u element *technicianId* kako bi se znalo da

je on opredijeljen na izvršavanje navedenog zahtjeva. Važno je napomenuti da ovo pravilo vrijedi isključivo prilikom izmjene postojećeg zahtjeva. Posljednja mogućnost je status *closed*, nakon kojeg se samo izmjenjuje status *ticket* objekta. Kao što je već ranije objašnjeno, budući da se isti obrazac koristi za stvaranje i uređivanje zahtjeva, svi statusi su implementirani na jednom zajedničkom mjestu. U konkretnom slučaju, implementacija statusa *closed* je u kodu prisutna zajedno s druga dva statusa, ali se prikazuje na korisničkom sučelju samo prilikom uređivanja postojećeg zahtjeva, a ne prilikom kreiranja novog. U *Kodnom isječku* 4.3. prikazan je objekt koji sadrži sve raspoložive statusne opcije za zahtjev i iz kojeg se čita prilikom odabira statusa.

```
const statusOptions = [
  ...((!ticket.id || user.role === 'admin') ? [{ value: 'open', label: 'open' }] : []),
  { value: 'taken', label: 'taken' },
  ...(ticket.id && (ticket.technician_id.includes(user.id) || user.role === 'admin') ? [{ value: 'closed', label: 'closed' }] : []),
];
```

Kodni isječak 4.3 - Polje statusa zahtjeva

Kod prikaza statusa u padajućem izborniku provodi se dodatna provjera specifičnih parametara. U situaciji gdje postoji *ticket.id*, to ukazuje da se vrši ažuriranje postojećeg zahtjeva, a ne stvaranje novog. Ovo znači da status *open* neće biti vidljiv ako zahtjev već postoji. No, zbog drugog uvjeta, status će i dalje biti prikazan *admin* korisniku, čak i ako zahtjev već postoji. To je zbog mogućnosti navedenog korisnika da ukloni sve tehničare sa zahtjeva i vrati ga u početno stanje, što nije moguće za *technician* korisnika. Status *taken* će se prikazivati u bilo kojem slučaju. Status *closed* će biti prikazan u slučaju da se radi izmjena nad postojećim zahtjevom jer nema smisla da se prilikom kreiranja novog može postaviti status *closed*. Dodatni uvjeti za prikaz su da prijavljeni korisnik ima ulogu *admin* ili ukoliko se radi o *technician* korisniku, njegov identifikator se mora nalaziti unutar *technicianId* elementa *ticket* objekta. Važno je napomenuti da se provjera identiteta tehničara primjenjuje samo prilikom izmjene postojećeg zahtjeva, a ne prilikom kreiranja novog. S obzirom na to da se isti obrazac koristi za oba slučaja, ove provjere su smještene jedna pored druge.

Sljedeća stavka unutar obrasca za kreiranje zahtjeva je polje za odabir klijenta te mogućnost kreiranja novog. U *Kodnom isječku* 4.4 prikazan je način izvedbe polja

za odabir klijenta te poziv obrasca za kreiranje novog.

```
<Row>
  {user.role === "admin" && (
    <Col md={6}>
      <Button type="button" onClick={toggleClientForm} className="mb-3">
        Create New Client
      </Button>
    </Col>
  )}
</Row>
<Row>
  <Col>
    <NewClientForm
      isOpen={isModalOpen}
      onClose={toggleClientForm}
      onCreateClient={handleNewClientCreate}
    />
    <Form.Group controlId="ticketClient">
      <Form.Label>Client name:</Form.Label>
      <Select
        value={clientOptions.find((option) => option.value === ticket.client_id)}
        options={clientOptions}
        onChange={(selectedOption) =>
          setTicket({ ...ticket, client_id: '' + selectedOption.value })
        }
        isDisabled={user.role === 'tech'}
        placeholder={!ticket.client_id ? 'Select a client' : undefined}
      />
    </Form.Group>
  </Col>
</Row>
```

Kodni isječak 4.4 - Polje odabira klijenta te poziv za kreiranje novog

Sposobnost stvaranja novog klijenta unutar obrasca za generiranje novog zahtjeva dostupna je isključivo *admin* korisniku, kao što je i vidljivo iz prikazanog koda. Klikom na gumb *Create New Client*, boolean vrijednost *isModalOpen* se postavlja na *true*, što rezultira otvaranjem modalnog prozora s obrascem za stvaranje novog klijenta. Slanjem ispunjenih podataka ili zatvaranjem modalnog prozora, poziva se funkcija *toggleClientForm()* kojom se mijenja vrijednost *isModalOpen* na suprotnu od one trenutne. U trenutnoj situaciji, vrijednost se postavlja sa *true* na *false* zbog čega dolazi do zatvaranja navedenog modalnog prozora. Također je bitno napomenuti da se prilikom slanja ispunjenih podataka poziva funkcija *handleNewClientCreate()* čija je uloga nanovo napraviti poziv prema serverskoj strani za dohvat svih klijenata kako bi se unutar polja za odabir klijenata nalazio i novonastali.

Prilikom odabira klijenta, vrijednost polja se određuje pretragom svih klijenata sustava te uspoređivanjem njihovog identifikatora s vrijednosti zapisanoj u *clientId* elementu *ticket* objekta. Među ostalim opcijama za klijente nalaze se svi preostali klijenti. Uslijed odabira klijenta odrađuje se spremanje njegovog identifikatora u *clientId* element *ticket* objekta. Ukoliko nije odabran niti jedan klijent, u polju će biti prikazana rečenica *"Select a client"*. Važno je istaknuti da korisnici s *technician* ulogom nemaju pristup ovoj funkcionalnosti, pri čemu se navedeno ograničenje odnosi isključivo na izmjene postojećih zahtjeva.

Posljednje polje obrasca odnosi se na odabir tehničara koji će biti zaduženi za rješavanje zahtjeva. Implementacija navedene funkcionalnosti se nalazi u *Kodnom isječku* 4.5.

```
<Row>
  {ticket.status !== 'open' && ticket.status !== '' && user.role === "admin" && (
    <Col>
      <Form.Group controlId="ticketTechnician">
        <Form.Label>Technicians:</Form.Label>
        <Select
          className="select-container"
          isMulti
          placeholder="Select technician(s)"
          options={technicianOptions}
          value={technicianOptions.filter((technician) =>
            isTechnicianSelected(technician.value)
          )}
          onChange={(selectedOptions) => {
            const selectedTechnicians = selectedOptions.map((option) => option.value);
            setTicket({ ...ticket, technician_id: selectedTechnicians });
          }}
        />
      </Form.Group>
    </Col>
  )}
</Row>
```

Kodni isječak 4.5 - Odabir tehničara

Polje obrasca će biti prikazano samo ako je status zahtjeva definiran te je različit od *open*. Dodatno, za prikaz navedenog polja, nužno je da je uloga prijavljenog korisnika *admin*. Odabir tehničara je definiran atributom *isMulti* koji označuje mogućnost odabira više od jedne opcije. Pod opcijama odabira se nalaze svi tehničari sustava pohranjeni u *technicianOptions* objekt.

Za provjeru postoje li već odabrani tehničari, koristi se funkcija *isTechnicianSe-*

lected(). Implementacija navedene funkcije se nalazi u *Kodnom isječku* 4.6.

```
const isTechnicianSelected = (technicianId) =>
  ticket.technician_id.includes(technicianId);
```

Kodni isječak 4.6 - Funkcija za provjeru odabranih tehničara

U *Kodnom isječku* 4.5 iterira se kroz tehničare, pri čemu se njihov identifikator šalje kao argument funkciji u *Kodnom isječku* 4.6. U funkciji se provjerava je li taj identifikator prisutan unutar *technicianId* elementa objekta *ticket*. Bitno je napomenuti da je tip varijable *technicianId* polje cijelih brojeva iz razloga što sadrži identifikatore svih tehničara dodijenjenih na zahtjev.

Posljednji atribut unutar polja za odabir tehničara odnosi se na izmjenu vrijednosti *technicianId* elementa *ticket* objekta. Dohvaćaju se identifikatori svih odabranih tehničara te se spremaju u navedeni element u obliku polja cijelih brojeva.

4.1.2 Slanje podataka prema serverskoj strani

Nadovezujući se na prijašnji odjeljak, klikom gumba *Create* na obrascu za kreiranje zahtjeva poziva se funkcija *onSumbit()*, kao što je prikazano u *Kodnom isječku* 4.1 u prvoj liniji. Izgled *ticket* objekta koji se šalje na serversku stranu nalazi se u *Kodnom isječku* 4.7.

```
const ticket: {
    id: string;
    name: string;
    description: string;
    status: string;
    client_id: string;
    technician_id: never[];
}
```

Kodni isječak 4.7 - Definicija *ticket* objekta

Pošto se za kreiranje i izmjenu zahtjeva koristi identičan obrazac, funkcija *onSumbit()* se koristi u oba slučaja, kao što je vidljivo u *Kodnom isječku* 4.8.

```
const onSubmit = ev => {
  ev.preventDefault()
  if (ticket.id) {
    axiosClient.put(`/tickets/${ticket.id}`, ticket)
      .then(() => {
        setNotification('Ticket was successfully updated')
        navigate('/tickets')
      })
      .catch(err => {
        const response = err.response;
        if (response && response.status === 422) {
          setErrors(response.data.errors)
        }
      })
  } else {
    axiosClient.post('/tickets', ticket)
      .then(() => {
        setNotification('Ticket was successfully created')
        navigate('/tickets')
      })
      .catch(err => {
        const response = err.response;
        if (response && response.status === 422) {
          setErrors(response.data.errors)
        }
      })
  }
}
```

Kodni isječak 4.8 - Funkcija za slanje podataka o zahtjevu prema serverskoj strani

Vrijedno je istaknuti da je u upotrebi funkcija *preventDefault()*, koja sprječava standardne radnje internet preglednika koje se izvršavaju uslijed podnošenja zahtjeva za slanjem podataka. To je učinjeno kako bi se omogućilo nesmetano i uspješno izvršavanje ostatka JavaScript koda.

Kao prethodno objašnjeno, funkcija *onSubmit()* se aktivira pri stvaranju novog zahtjeva i prilikom njegove izmjene. Daljnji koraci se utvrđuju na temelju provjere prisutnosti identifikatora zahtjeva. Ako identifikator ne postoji, izvršava se *POST* zahtjev prema serverskoj strani s popunjenim *ticket* objektom. S druge strane, pri-

sutnost identifikatora ukazuje na potrebu izvršavanja *PUT* zahtjeva, gdje se zahtjev identificira prema svom identifikatoru i ažurira s novim podacima.

4.1.3 Dohvat podataka na serverskoj strani

Na strani poslužitelja, prvo se provodi provjera unutar datoteke usmjerivača (eng. router) koja sadrži funkcije za usmjeravanje toka podataka prema odgovarajućim upravljačkim funkcijama. Sadržaj navedene datoteke prikazan je u *Kodnom isječku* 4.9.

```
Route::middleware('auth:sanctum')->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']);
    Route::get('/user', function (Request $request) {
        return $request->user();
    });

    Route::apiResource('/users', UserController::class);
    Route::apiResource('/tickets', TicketController::class);
    Route::apiResource('/clients', ClientController::class);
    Route::apiResource('/comments', CommentController::class);
    Route::apiResource('/technicians', TechnicianController::class);
    Route::apiResource('/subtasks', SubtaskController::class);
});

Route::post('/signup', [AuthController::class, 'signup']);
Route::post('/login', [AuthController::class, 'login']);
```

Kodni isječak 4.9 - *Router* datoteka na serverskoj strani aplikacije

Svaka ruta kojom se vrši manipulacija podataka unutar sustava je zaštićena Laravel Sanctumom. Ovo osigurava da korisnici moraju imati valjani *Bearer* token kako bi pristupili rutama. Samim tim, neautoriziranim korisnicima je onemogućen pristup. Provjera za daljnju propagaciju podataka se obavlja na osnovu putanje definirane u zahtjevu, kao i vrsti zahtjeva. U konkretnom scenariju, kada se šalje *POST* zahtjev na putanju *"/tickets"*, usmjerivač (eng. router) preusmjerava zahtjev funkciji *store()* unutar *TicketController* datoteke. Detaljni prikaz svih ruta vezanih za putanju *"/tickets"* se nalazi na *slici* 4.1.

```
GET|HEAD        api/tickets ..................................... tickets.index › Api\TicketController@index
POST            api/tickets ..................................... tickets.store › Api\TicketController@store
GET|HEAD        api/tickets/{ticket} ............................ tickets.show › Api\TicketController@show
PUT|PATCH       api/tickets/{ticket} ........................... tickets.update › Api\TicketController@update
DELETE          api/tickets/{ticket} ........................... tickets.destroy › Api\TicketController@destroy
```

Slika 4.1 Detaljni prikaz ruta zahtjeva

Podaci se pregledavaju prije nego što budu proslijeđeni funkciji *store()*. Ova provjera se obavlja putem klase *StoreTicketRequest*, gdje se za svaki pojedini podatak ispituje ispunjava li određene uvjete. Implementacija navedene klase prikazana je u *Kodnom isječku* 4.10.

```php
class StoreTicketRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array|string>
     */
    public function rules(): array
    {
        return [
            'name' => 'required|string',
            'description' => 'required|string',
            'status' => 'required|string',
            'technician_id' => 'required|array',
            'client_id' => 'required|string',
        ];
    }
}
```

Kodni isječak 4.10 - Klasa za provjeru valjanosti podataka

U funkciji *authorize()* postavlja se boolean vrijednost koja označava mogućnost korisnika da nastavi koristiti podatke koji su dobiveni s klijentske strane aplikacije. Funkcija *rules()* sadrži uvjete koje svaki pojedini podatak mora zadovoljiti kako bi

bilo dopušteno daljnje korištenje tih podataka. Za svaki podatak zahtijeva se da bude popunjen jer je obavezan (eng. required), te da su svi podaci nizovi znakova (eng. string), osim podatka *technicianId* za koji se očekuje da bude polje (eng. array).

Nakon uspješne provjere valjanosti podataka u klasi *StoreTicketRequest*, slijedi izvršavanje funkcije *store()* unutar klase *TicketController*, koja je prikazana u *Kodnom isječku* 4.11.

```php
public function store(StoreTicketRequest $request)
{
    $technicianIds = implode(',', $request->input('technician_id', []));

    $ticket = new Ticket([
        'name' => $request->input('name'),
        'description' => $request->input('description'),
        'status' => $request->input('status'),
        'technician_id' => $technicianIds,
        'client_id' => $request->input('client_id'),
    ]);

    $ticket->save();

    return new TicketResource($ticket);
}
```

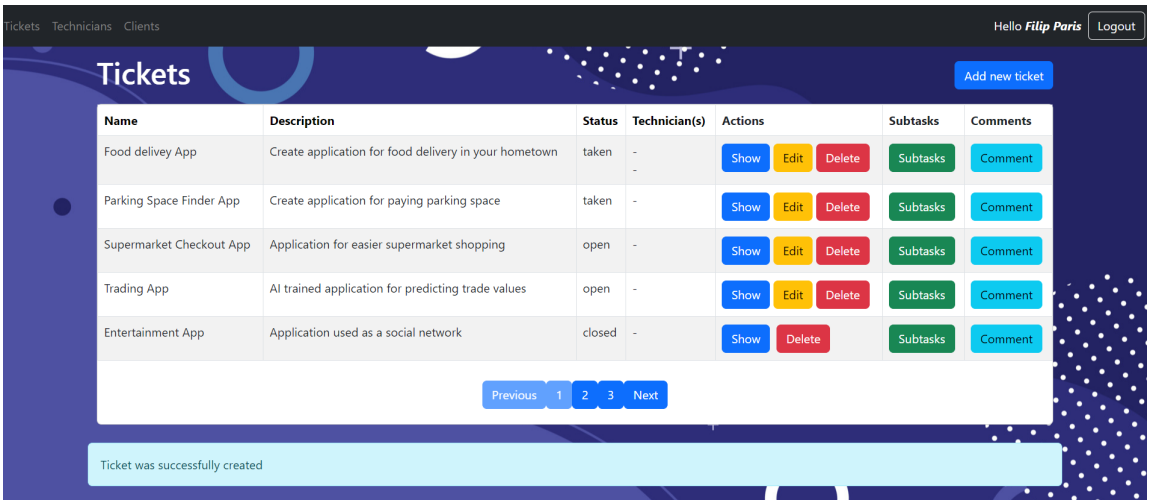
Kodni isječak 4.11 - Spremanje zahtjeva u bazu podataka

U prvoj liniji koda funkcije, izvlači se niz cijelih brojeva koji sadrži identifikatore tehničara dodijeljenih zahtjevu. Dobiveni identifikatori se potom konkateniraju u niz znakova u kojem su međusobno razdvojeni razmacima. Zatim se kreira instanca *Ticket* klase u čije se elemente zapisuju podaci dobiveni s klijentske strane aplikacije. Nadalje, izvršava se spremanje u bazu podataka i naposlijetsku se prema klijentskoj strani aplikacije šalje novonastali zahtjev u svrhu prikaza na grafičkom korisničkom sučelju. Primjer spremljenog zahtjeva u bazu podataka se nalazi na *slici* 4.2.

id	created_at	updated_at	name	description	status	client_id	technician_id
59	2023-08-17 09:23:24	2023-08-17 09:23:24	Calorie counter app	App that counts calories taken throught the day	taken	78	32,40,30

Slika 4.2 Zahtjev spremljen u bazu podataka

Nadovezujući se na *Kodni isječak* 4.8, uslijed uspješnog *POST* zahtjeva prema serverskoj strani aplikacije, postavlja se obavijesna poruka *"Ticket was successfully created"*, nakon čega slijedi preusmjeravanje na stranicu za prikaz svih zahtjeva. Izgled navedene stranice s obavijesti je vidljiv na *slici* 4.3.



Slika 4.3 Prikaz svih zahtjeva s obavijesti o uspješnom kreiranju

4.2 Kreiranje komentara

4.2.1 Implementacija na klijentskoj strani

Pritiskom na *Comment* gumb za određeni zahtjev na stranici za pregled svih zahtjeva, korisnik će biti preusmjeren na stranicu koja prikazuje sve komentare vezane za taj zahtjev. Na toj stranici također postoji gumb *Add new comment*, kojim se otvara obrazac za unos novog komentara. Programska implementacija obrasca vidljiva je u *Kodnom isječku* 4.12.

```
{showAddComment && (
  <div>
    <Form onSubmit={handleAddComment}>
      <Form.Label className='custom'>Description:</Form.Label>
      <Form.Control type="text" name="description" required />
      <br/>
      <Button type="submit" className="btn-success">Add Comment</Button>
    </Form>
    <br/>
  </div>
)}
```

Kodni isječak 4.12 - Obrazac za kreiranje novog komentara

Za utvrđivanje potrebe prikaza obrasca na korisničkom sučelju ili ne, koristi se boolean parametar naziva *showAddComment*. Klikom gumba *Add new comment* na stranici za prikaz svih komentara, navedeni parametar će se postaviti na vrijednost *true*, što će rezulirati prikazom obrasca. Navedeni obrazac sastoji se isključivo od jednog polja za unos komentara, koje je realizirano kao osnovno tekstualno polje. Nadalje, klikom na gumb *Add Comment*, poziva se funkcija naziva *handleAddComment()*, prikazana u *Kodnom isječku* 4.13.

```
const handleAddComment = async (e) => {
  e.preventDefault();
  const description = e.target.elements.description.value;

  const newComment = {
    user_id: String(user.id),
    ticket_id: ticketId,
    description: description,
  };

  try {
    await axiosClient.post(`/comments/`, newComment);

    e.target.reset();
    setShowAddComment(false);

    fetchData();
  } catch (error) {
    console.error("Error creating comment:", error);
  }
};
```

Kodni isječak 4.13 - Funkcija za slanje komentara prema serverskoj strani

Kao i kod kreiranja zahtjeva, koristi se *preventDefault()* funkcija kojom se sprječava internet pregledniku odrađivanje predefiniranih radnji uslijed pokretanja zahtjeva za slanjem podataka.

U varijablu *description* se zapisuje vrijednost koju korisnik unese u tekstualno polje prilikom kreiranja komentara. Nadalje, stvara se novi objekt koji se nakon ispune šalje prema serverskoj strani. Objekt sadrži identifikator trenutno prijavljenog korisnika u sustav, identifikator zahtjeva za koji se kreira komentar te sam tekst komentara. Konačno, izvršava se POST zahtjev prema serverskoj strani aplikacije.

4.2.2 Prikupljanje i obrada podataka na serverskoj strani

Kao što je ranije spomenuto, svaka ruta kojom se vrši manipulacija podataka unutar sustava je zaštićena Laravel Sanctumom, među kojima se nalazi i ona za kreiranje komentara. Prilikom obrade podataka na serverskoj strani, oni prvo prolaze kroz klasu *StoreCommentRequest*, koja je prikazana u *Kodnom isječku* 4.14.

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreCommentRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array|string>
     */
    public function rules(): array
    {
        return [
            'description' => 'required|string',
            'user_id' => 'required|string',
            'ticket_id' => 'required|string',
        ];
    }
}
```

Kodni isječak 4.14 - Klasa za validaciju primljenih podataka s klijentske strane

U navedenoj klasi vrši se dodatna provjera svakog podatka koji dolazi s klijentske strane aplikacije. Za svaki podatak se očekuje da bude prisutan i da nije prazan. Također, svaki podatak mora biti u obliku niza znakova (eng. string). Ukoliko su svi uvjeti zadovoljeni, korisniku se omogućuje daljnja propagacija podataka.

Nadalje, izvršava se logika zapisana u funkciji *store()* unutar *CommentController* klase, vidljivoj u *Kodnom isječku* 4.15.

```php
public function store(StoreCommentRequest $request)
{
    $comment = new Comment([
        'description' => $request->input('description'),
        'ticket_id' => $request->input('ticket_id'),
        'user_id' => $request->input('user_id'),
    ]);

    $comment->save();

    return response(null, 201);
}
```

Kodni isječak 4.15 - Funkcija za spremanje podataka u bazu

Kreira se istanca objekta *Comment* te se njegovi elementi ispunjuju podacima dohvaćenim s klijente strane. Nadalje, odrađuje se spremanje novog komentara u bazu podataka. Naposljetku, na klijetsku stranu aplikacije se šalje uspješan status kreiranja.

Nadovezujući se na *Kodni isječak* 4.13, primitkom odgovora na klijentskoj strani odrađuje se resetiranje podataka unutar događaja (eng. event) iz kojeg je prvotno bila pročitana vrijednost komentara. Nadalje, boolean vrijednost *showAddComment* postavlja se na *false* kako se obrazac za kreiranje novog komentara više ne bi prikazivao na korisničkom sučelju. Naposljetku, poziva se funkcija *fetchData()* kojom se ažuriraju prikazani komentari.

Poglavlje 5

Zaključak

U ovom radu predstavljena je aplikacija za evidentiranje zahtjeva, odnosno zadataka, uz dodatnu korisnu funkcionalnost praćenja napretka izvršavanja svakog pojedinačnog zadatka. Ova aplikacija također pruža opciju definiranja podzadataka, omogućujući raslojavanje većih problema na manje segmente. Ovim pristupom se unapređuje organizacija i jasnije definiraju koraci koje je potrebno odraditi prilikom rješavanja zadataka. Također, važan element je i jednostavnost uporabe aplikacije za sve vrste korisnika, bilo da imaju potrebno domensko znanje ili ne. Osim toga, aplikacija pruža mogućnost stvaranja komentara koji mogu imati veliku važnost u slučaju bilo kakvih promjena ili problema tijekom procesa rješavanja zadataka. Naposljetku, uloge koje se primjenjuju na korisnike oblikuju aplikaciju na način da se nameću određena ograničenja prema pojedinim funkcionalnostima.

Laravel programski okvir odabran je zbog svoje specijalizacije za izradu *web* aplikacija. Pomoću Artisan komandnog sučelja olakšano je stvaranje različitih vrsta datoteka na jednostavan i efikasan način, značajno ubrzavajući proces razvoja aplikacije. Također, sadrži unaprijed definirane funkcionalnosti za česte CRUD akcije, što je zapravo baza projekta.

React knjižnica je korištena zbog svoje svestranosti i prilagodljivosti. Izmjena stranica je implementirana na poprilično jednostavan način korištenjem *routera*. Nadalje, mogućnost automatskog renderiranja novoizmjenjenih funkcionalnosti rezultira uštedom vremena koje se može utrošiti na dodatni razvoj aplikacije. Naposljetku, ve-

lika popularnost navedene knjižnice olakšava pronalaženje rješenja za razne probleme putem velikog broja foruma.

MySQL je upotrijebljen zbog svoje popularnosti te intuitivnog koncepta. U projektu gdje se provodi osnovna obrada podataka, MySQL je odabran kao optimalno rješenje. Dinamičan prikaz stranica, kao i njihova skalabilnost, odrađen je pomoću Bootstrap tehnologije. Korištenjem ove tehnologije, brze i značajne promjene na dizajnu moguće su u kratkom vremenu. Radi toga se pokazala kao najbolji izbor za posljednje prilagodbe na dizajnu stranica.

Prilikom prijave korisnika u sustav, bilo bi potrebno implementirati dodatnu autorizaciju korisnika. Opcije za to uključuju korištenje aplikacija poput *Microsoft Authenticatora* ili *VIP Accessa*, ili razvoj vlastitog sustava dodatne korisničke provjere.

Nadalje, važno je osigurati korištenje valjanih e-mail adresa ukoliko bi aplikacija bila upotrebljavana u stvarnom okruženju. To implicira da bi prilikom stvaranja novog korisničkog računa trebalo uvesti provjeru ispravnosti e-mail adresa. Nadalje, važno je dodatno implementirati mehanizam slanja obavijesti kad god se dogode promjene na pojedinim zahtjevima. *Admin* korisnik bi primao e-mail obavijesti uslijed bilo kakve promjene u sustavu, dok *technician* korisnici bi bili obaviješteni ukoliko se dogodi izmjena na zahtjevima na kojima su oni zaduženi.

S obzirom na trenutnu dostupnost isključivo verzije aplikacije za stolna računala i laptope, postoji mogućnost razvoja mobilne aplikacije u budućnosti. Budući da su klijentska i serverska strana aplikacije razdvojeni, ne bi bilo potrebno raditi nikakve preinake na aplikaciji. Serverska strana bi ostala nepromijenjena, što znači da bi bilo potrebno trajno je pokrenuti na određenoj internet lokaciji, omogućujući pristup putem klijentske strane aplikacije. Klijentska strana bi bila realizirana kroz mobilnu aplikaciju sadržavajući sve funkcionalnosti prisutne u verziji za stolna računala i laptope.

Ukoliko bi sustav koristio veliki broj ljudi, bilo bi potrebno napraviti više uloga. Svaka uloga bi imala neka svoja ograničenja ovisno o namjeni i veličini aplikacije. Također bi mogla postojati grupacija zahtjeva ovisno o namjeni unutar kojih bi samo određeni korisnici vidjeli sadržaj. Primarna uloga, *admin*, bi i dalje omogućavala potpuni pristup svim funkcionalnostima aplikacije.

Bibliografija

[1] Bootstrap. , s Interneta, https://getbootstrap.com/ , kolovoz 2023.

[2] CSS. , s Interneta, https://www.w3schools.com/css/ , kolovoz 2023.

[3] HTML. , s Interneta, https://html.com/ , kolovoz 2023.

[4] React. , s Interneta, https://legacy.reactjs.org/ , kolovoz 2023.

[5] PHP. , s Interneta, http://www.php.net

[6] Laravel. , s Interneta, https://laravel.com/ , kolovoz 2023.

[7] MySQL. , s Interneta, https://www.mysql.com/ , kolovoz 2023.

[8] Node.JS. , s Interneta, https://nodejs.org/en , kolovoz 2023.

[9] Composer. , s Interneta, https://getcomposer.org/ , kolovoz 2023.

[10] Artisan. , s Interneta, https://laravel.com/docs/10.x/artisan , kolovoz 2023.

Sažetak

U ovom radu predstavljena je RESTful web aplikacija za upravljanje zahtjevima koja omogućuje praćenje faza izvršavanja svakog zahtjeva. Glavna značajka aplikacije je upotreba uloga koje omogućuju kontrolu pristupa različitim funkcionalnostima. Također, aplikacija podržava komentiranje i stvaranje podzahtjeva za svaki zatraženi zahtjev. Za razvoj korisničkog sučelja korišten je React zbog njegove jednostavne implementacije, dok je za izradu serverske strane aplikacije odabran Laravel zbog svoje prikladnosti za razvoj *web* aplikacija. Krajnja stilizacija korisničkog sučelja je realizirana Boostrap tehnologijom, a za pohranu podataka korišten je MySQL sustav.

Ključne riječi **— zahtjev, status, React, Laravel**

Abstract

This paper presents a RESTful web application for managing requests, allowing tracking of the execution phases of each request. The main feature of the application is the use of roles that enable access control to different functionalities. Additionally, the application supports commenting and creating sub-requests for each requested task. React was used for developing the user interface due to its straightforward implementation, while Laravel was chosen for building the server-side application because of its suitability for web application development. The final styling of the user interface was achieved using Bootstrap technology, and MySQL system was used for data storage.

Keywords **— ticket, status, React, Laravel**