

# Web aplikacija za zdravstvene usluge

---

Zec, Hrvoje

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:200:612299>

Rights / Prava: [In copyright](#) / [Zaštićeno autorskim pravom](#).

Download date / Datum preuzimanja: **2024-12-24**

Repository / Repozitorij:

[Faculty of Electrical Engineering, Computer Science and Information Technology Osijek](#)



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

**Sveučilišni studij**

**WEB APLIKACIJA ZA ZDRAVSTVENE USLUGE**

**Završni rad**

**Hrvoje Zec**

**Osijek, 2022.**

**FERIT**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA  
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac ZIP - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 08.09.2022.

Odboru za završne i diplomske ispite

**Prijedlog ocjene završnog rada na  
preddiplomskom sveučilišnom studiju**

<b>Ime i prezime Pristupnika:</b>	Hrvoje Zec
<b>Studij, smjer:</b>	Preddiplomski sveučilišni studij Računarstvo
<b>Mat. br. Pristupnika, godina upisa:</b>	R4301, 26.07.2018.
<b>OIB Pristupnika:</b>	13825042338
<b>Mentor:</b>	Izv. prof. dr. sc. Josip Balen
<b>Sumentor:</b>	,
<b>Sumentor iz tvrtke:</b>	
<b>Naslov završnog rada:</b>	Web aplikacija za zdravstvene usluge
<b>Znanstvena grana rada:</b>	Programsko inženjerstvo (zn. polje računarstvo)
<b>Zadatak završnog rad:</b>	
<b>Prijedlog ocjene završnog rada:</b>	Izvrstan (5)
<b>Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:</b>	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 3 razina
<b>Datum prijedloga ocjene od strane mentora:</b>	08.09.2022.
<b>Datum potvrde ocjene od strane Odbora:</b>	
<b>Potvrda mentora o predaji konačne verzije rada:</b>	<i>Mentor elektronički potpisao predaju konačne verzije.</i>
	Datum:

**FERIT**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA  
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK**IZJAVA O ORIGINALNOSTI RADA**

Osijek, 20.09.2022.

<b>Ime i prezime studenta:</b>	Hrvoje Zec
<b>Studij:</b>	Preddiplomski sveučilišni studij Računarstvo
<b>Mat. br. studenta, godina upisa:</b>	R4301, 26.07.2018.
<b>Turnitin podudaranje [%]:</b>	9

Ovom izjavom izjavljujem da je rad pod nazivom: **Web aplikacija za zdravstvene usluge**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Josip Balen

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.  
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

# SADRŽAJ

<b>1. UVOD</b>	<b>1</b>
<b>1.1. Zadatak završnog rada</b>	<b>2</b>
<b>2. APLIKACIJE PRIMJENJENE ZA REZERVIRANJE TERMINA U RAZLIČITE SVRHE</b>	<b>3</b>
<b>3. ALATI I PROGRAMSKI JEZICI KORIŠTENI U IZRADI RADA</b>	<b>4</b>
<b>3.1. Uređivač izvornog koda Visual Studio Code</b>	<b>4</b>
<b>3.2. Programski alat Postman</b>	<b>5</b>
<b>3.3. Programski jezik JavaScript</b>	<b>6</b>
<b>3.4. Frontend dio aplikacije</b>	<b>6</b>
3.4.1. HTML opisni jezik	6
3.4.2. CSS stilski jezik	7
3.4.3. SCSS stilski jezik	7
3.4.4. Bootstrap razvojni web okvir	7
3.4.5. Mantine	8
3.4.6. Postavljanje Mantinea	8
3.4.7. React	10
3.4.8. Kreiranje React aplikacije	11
<b>3.5. Backend dio aplikacije</b>	<b>11</b>
3.5.1. Node.js	11
3.5.2. Express.js	12
3.5.3. Postavljanje express.js na server	12
<b>3.6. Baza podataka</b>	<b>14</b>
3.6.1. Autentifikacija	14
3.6.2. Baza podataka u stvarnom vremenu	14
3.6.3. Postavljanje baze podataka	15
<b>4. IZRADA APLIKACIJE ZA REZERVACIJU TERMINA KOD DOKTORA SPECIJALISTA</b>	<b>18</b>
<b>5. IZGLED APLIKACIJE</b>	<b>33</b>
<b>6. ZAKLJUČAK</b>	<b>40</b>
<b>LITERATURA</b>	<b>41</b>

<b>POPIS UPOTREBLJENIH KRATICA</b>	<b>42</b>
<b>SAŽETAK</b>	<b>43</b>
<b>ABSTRACT</b>	<b>44</b>
<b>ŽIVOTOPIS</b>	<b>45</b>
<b>PRILOZI</b>	<b>46</b>
<b>P.1. Izvorni kod aplikacije</b>	<b>46</b>

# 1. UVOD

U kineskom gradu Wuhanu krajem prosinca 2019. godine u provinciji Hubei, po prvi put su zabilježeni slučajevi infekcije SARS-CoV-2 virusom koji uzrokuje bolest COVID-19 [1] poznatiji pod nazivom korona virus. U samo nekoliko mjeseci virus je zarazio cijeli svijet. Prvi slučaj u Hrvatskoj zabilježen je 25. veljače 2020. godine. Nepoznavanje novog virusa uzrokovalo je naglo širenje po cijelome svijetu što je dovelo do panike među ljudima. U svrhu zaštite populacije svaka država morala je uvesti restriktivne mjere koje su između ostalog uključivale uvođenjem lockdowna čiji je cilj bio smanjit kontakt među ljudima.

Shodno tome, da izbjegnemo što veći kontakt između ljudi, izrada same aplikacije omogućuje korisniku odnosno pacijentu da se što manje izlaže okolini, odnosno sama funkcija aplikacije je da nudi mogućnost za tri različita korisnika. Jedan od njih je doktor opće prakse kojem se nudi mogućnost da napravi digitalnim putem uputnicu za pacijenta. Pacijent može rezervirati termin kod željenog doktora specijalista za kojeg je dana uputnica, dok doktor specijalist kao korisniku se nudi da vidi sve tražene termine od pacijenta i mogućnost odbijanja ili prihvaćanja ovisno o potrebama korisnika. Korisnik odlaskom kod doktora opće prakse dobiva uputnicu za doktora specijalista kako bi se detaljnije riješio problem pacijenta. Aplikacija nudi rješenje da pacijent s tom uputnicom ne mora dodatno ići u bolnicu i rezervirati termin kod željenog specijalista, već bi mogao otići kući i preko aplikacije rezervirati termin kod specijalista koji korisnik sam odabire. Time ljudi ne moraju odlaziti u bolnicu i čekati u redu kako bi rezervirali termin te gubiti vrijeme, nego bi jednostavnijim rješenjem preko aplikacije sami od kuće rezervirali termin kod željenog specijalista. Specijalist bi dobio ponudu da ima traženi termin od pacijenta te odlučuje odgovara li mu termin pacijenta. Pacijent bi dobio povratnu informaciju od specijalista dali je termin uspješno rezerviran ili da traži neki drugi termin. Uz to sami pacijent ima mogućnost odjaviti termin. U konačnici sama aplikacija bi nudila bržu rezervaciju termina kod doktora, te lakšu interakciju između doktora i pacijenta.

Sami rad sastavljen je od 5 poglavlja, od kojih prva dva opisuju problem koji završni rad rješava prednosti koje nudi, te sami zadatak završnog rada. 3. poglavlje predstavlja alate i programske jezike koje su nam potrebne i koje ćemo koristiti u izradi same web aplikacije te objašnjavanje istih. Nadalje tome u 4.-om poglavlju opisati ćemo sami izgled, rad aplikacije, programski kod te, mogućnosti koje nam nudi web aplikacija. Na kraju imamo zaključak koji predstavlja zadnje, odnosno peto podglavlje u kojem ćemo sažeti rad.

## **1.1. Zadatak završnog rada**

Zadatak koji se želi postići web aplikacijom je konstruiranje sustava čiji bi cilj bio olakšavanje pacijentu rezervaciju termina kod doktora specijalista, gdje bi rezultat bio brža rezervacija kod doktora, smanjeno kretanje pacijenta i ušteda vremena.



## **2. APLIKACIJE PRIMJENJENE ZA REZERVIRANJE TERMINA U RAZLIČITE SVRHE**

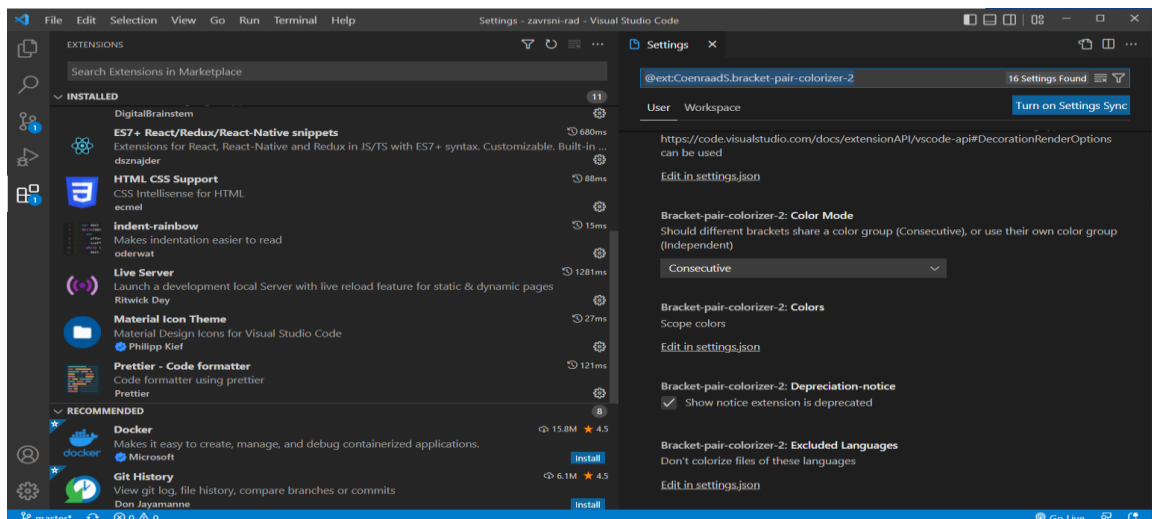
Razvojem interneta i tehnologije olakšale su se mnoge aktivnosti u svakodnevnom životu. Sve više i više koristimo i primjenjujemo tehnologiju kako bi pojednostavili način života. Razvile su se aplikacije primijenjene za rezerviranje termina u različite svrhe, što je ubrzalo proces razvoja dobara. Same aplikacije kao što su Booking [2] ili crno jaje [3], omogućile su korisnicima da im različita mjesta budu što pristupačnija, te ubrzali razvoj turizma. Poveznicu možemo povući u zdravstveni sustav, kako bi se razvojem tehnologije i aplikacija ubrzalo i olakšao način života ljudi. Danas u svijetu postoje puno aplikacija koje su poboljšale način rada u zdravstvenom sustavu. Jedne od aplikacija koje nude slična rješenja web aplikaciji razvijenoj u radu su Practo ili Sminq [4], koja omogućuju rezervaciju termina kod određenog doktora specijalista, te isto tako mogu dobit obavijest dali im je termin prihvaćen ili nije, što je idealno za pacijente kojima je vrijeme kritičan faktor. Većinom se takve aplikacije sastoje od popunjavanja korisničkih osobnih podataka, odabir razloga zakazivanja termina kod doktora do toga da im se nudi mogućnost rezerviranja termina u željeno vrijeme, te isto tako praćenje dali je termin prihvaćen ili odbijen, odnosno nudi obostranu komunikaciju i povezanost između pacijenta i doktora.

### 3. ALATI I PROGRAMSKI JEZICI KORIŠTENI U IZRADI RADA

Za razvoj web aplikacije potrebno je razvojno okruženje u kojem smo kreirali programski kod te samim time i kreirali web aplikaciju. Jedan od alata za razvojno okruženje koristit ćemo VS code u kojem ćemo kreirati našu web aplikaciju. Dodatni alat za lakše testiranje API-ja aplicirat ćemo uz pomoć platforme Postman. Jedan od temelja ove aplikacije je JavaScript, koji je uostalom i temeljni okvir (engl. *Framework*) uz pomoć kojih će se kreirati aplikacija. Za stvaranje grafičkog korisničkog sučelja web stranice (engl. *Front-end*) upotrijebit ćemo JavaScript biblioteku React. Gradivni dio Reacta odnosno korisničkog sučelja (eng. *User interface*) izgradit ćemo pomoću HTML-a, dok stilski dio prednjeg djela aplikacije (engl. *Front-end*) izvodit ćemo uz pomoć SCSS-a kako bi nam bilo lakše za implementiranje CSS-a. Dio aplikacije koji korisnik ne vidi (engl. *Back-end*) upravljati ćemo pomoću okvira (engl. *Framework*) Express.js, koji se pokreće preko web servisa Node.js. Kako bi mogli sačuvati podatke sa Backenda, koje korisnik unosi s korisničkog sučelja treba nam neka baza podataka gdje ćemo spremit te podatke od korisnika, za to će nam pomoći Firebase.

#### 3.1. Uređivač izvornog koda Visual Studio Code

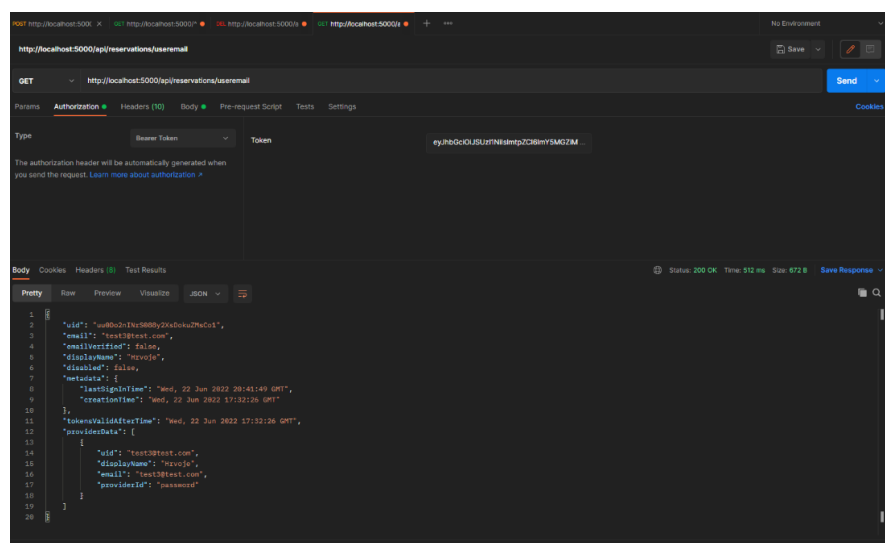
U anketi “Overflow 2021 Developer Survey” [5] Visual Studio Code je rangiran kao najpopularniji alat za razvojno okruženje, a 70% od 80.000 ispitanika izjavilo je da ga koriste. VS Code je lagan, ali prije svega moćan uređivač izvornog koda koji radi na vašoj radnoj površini (engl. *Desktop*) i dostupan je za Windows, macOS i Linux. Dolazi sa već ugrađenom podrškom za podržavanjem JavaScripta, TypeScripta i Node.js, te ima bogat sustav proširenja za druge jezike (kao što su c++, c#, Java, Python, PHP). Osim toga nudi jednostavno korištenje za uklanjanje bilo kakih pogrešaka (engl. *Debugging*), pametno dovršavanje koda, isječaka, te restrukturiranje koda. Nadalje, u sebi ima ugrađen Git sustav radi lakšeg raščlanjivanja koda na GitHub platformu. Kako bi nam lakše bilo pisati kod i raditi u VS Codeu, nudi nam bezbroj dodatnih vanjski proširenja (engl. *Extensins*) prikazano na slici 3.1, koje možemo s lakoćom skinuti i dodati u naš VS Code. Jedna od njih su ES7+React koji nam nudi kratice poput “*rfec*”, čija je uloga automatsko kreiranje “import-export” birane datoteke, te *Prettier* koji nam služi za ljepše formatiranje koda prilikom sačuvanja koda te mnoge druge ekstenzije.



Slika 3.1. Prikaz ekstenzija u razvojnom okruženju VS Code

## 3.2. Programski alat Postman

Postman je API platforma koja pomaže pri izgradnji i korištenju API-ja. Prema [6]. On pojednostavljuje svaki korak životnog ciklusa API-ja i olakšava suradnju tako da možemo kreirati bolje i brže API-je. Rezultat toga je da programerima olakšava stvaranje, dijeljenje, testiranje i dokumentiranje API-ja. To se postiže tako što se korisnicima omogućuje stvaranje i spremanje jednostavnih i složenih HTTP/s zahtjeva kao i čitanje njihovih odgovora. U našem radu Postman ćemo koristiti za testiranje i kreiranje API zahtjeva prilikom dodavanja klijentovih zahtjeva sa korisničkog djela (engl. *Front-end*), odnosno pri dodavanju uputnica te dohvaćanje istih sa Firebasea. Metode koje ćemo koristiti su GET za dohvaćanje prikazano na slici 3.2, POST za dodavanje, PATCH za ažuriranje, te DELETE za brisanje podataka.



Slika 3.2. Prikaz GET metode za dohvat klijenta po mail-u

Kao što je vidljivo na slici 3.2. preko Postmana smo poslali GET zahtjev na rutu <http://localhost:5000/api/reservations/useremail> te smo tako dohvatili podatke o korisniku preko njegovog maila koji se nalazi u našem Firebasau. Ono što nam je Postman omogućio je da bez pokretanja prednjeg djela aplikacije (engl. *Fron-end*) uspijemo dohvatiti podatke o željenom korisniku, te smo tim putem puno brži i možemo provesti više različitih testiranja bez da ovisimo o prednjem djelu aplikacije.

### **3.3. Programski jezik JavaScript**

JavaScript ili “JS” je programski jezik koji se najčešće koristi za dinamičku i interaktivnu web stranicu na strani klijenta, ali će često isto koristi i na strani servera, koristeći “*runtime*” kao što je Node.js. JavaScript se ne smije miješati sa programskim jezikom Java. Napravljen je da bude što bliže Java-i, ali nije objektno orijentirana kao što je Java. JavaScript se prvenstveno koristi u pregledniku, omogućujući programerima da manipuliraju sadržajem web stranice putem DOM-a, manipuliraju podacima pomoću AJAX-a i IndexedDBa, te komunicira s uređajem koji pokreće preglednik kroz različite API-je. Shodno tome JavaScript je jedan od najčešće korištenih jezika u svijetu, zahvaljujući nedavnom rastu i poboljšanju performansi API-ja dostupnih u pregledniku.

### **3.4. Frontend dio aplikacije**

Grafičko korisničko sučelje web aplikacije, odnosno poznatije pod nazivom (engl. *Front-end*), uz pomoć HTML-a, CSS-a, Reacta, te ostali alata i programskih jezika, stvara se web preglednik koji čini prednji dio stranice, čime omogućuje korisniku da mogu pregledati i komunicirati s određenom web stranicom.

#### **3.4.1. HTML opisni jezik**

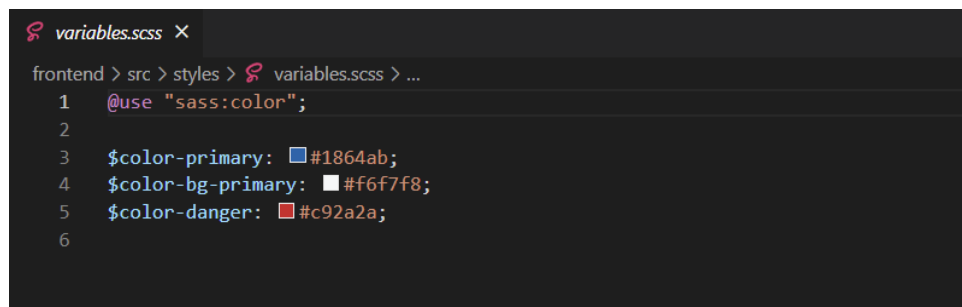
HTML ili punim imenom “Hypertext Markup Language”, je opisni jezik za web koji definira strukturu web stranice. To je jedan od najosnovnijih građevnih blokova svake web stranice, stoga je ključno naučiti ako želimo imati karijeru u web razvoju. Ona omogućuje web korisnicima stvaranje i strukturiranje odjeljaka, odlomka i poveznica pomoću elemenata, oznaka i atributa [7]. Međutim treba napomenuti da se HTML ne smatra programskim jezikom jer ne može stvoriti dinamičku funkcionalnost stranice. Da bi web stranica izgledala dobro i bila interaktivna za korisnika potrebno je koristiti potpomognute tehnologije kao što su CSS i JavaScript kako bi HTML učinili što ljepšim i interaktivnim.

### 3.4.2. CSS stilski jezik

Cascading Style Sheets, iliti skraćeno CSS, jednostavan je jezik dizajna koji je namijenjen pojednostavljenju procesa izrade web stranica da budu što atraktivan. Između ostalog CSS obrađuje izgled i dojam dijela web stranice. Koristeći CSS možemo kontrolirati boju teksta, stil fontova, razmake između paragrafa, veličinu i raspored stupaca, diktiranje pozadinskih slika i boja, dizajn vanjskog izgleda, varijacije u prikazu različitih uređaja i veličine zaslona kao i niz drugih učinaka. Jedna od prednosti CSS-a je ta da možemo napisati jednom CSS, te ga upotrebljavati na više HTML stranica. Nadalje možemo definirati stil za svaki HTML element i primijeniti ga na onoliko web stranica koliko želimo.

### 3.4.3. SCSS stilski jezik

Syntactically Awesome Style Sheet [8] je nad skup, te naprednija verzija CSS-a. Dizajnirana od Hampton Catlin, a razvili su ga Chris Eppstein i Natalie Weizenbaum. Zbog svojih naprednih značajki često se naziva Sassy CSS. SCSS ima ekstenziju datoteka, te sadrži sve značajke koje nisu prisutne u CSS-u, što ga čini dobrim izborom za razvojne programere da ga koriste. Nadalje on nudi varijable (slika 3.3.) uz pomoć kojih možemo skratiti svoj kod, što predstavlja veliku prednost u odnosu na konvencionalni CSS.



```
variables.scss ×
frontend > src > styles > variables.scss > ...
1  @use "sass:color";
2
3  $color-primary: #1864ab;
4  $color-bg-primary: #f6f7f8;
5  $color-danger: #c92a2a;
6
```

*Slika 3.3. prikaz varijabli u scss datoteci*

### 3.4.4. Bootstrap razvojni web okvir

Bootstrap [9] je besplatni i “open-source” razvojni web okvir (engl. *Framework*). Dizajniran je kako bi olakšao proces web razvoja responzivnih web-mjesta usmjerenih na mobilne uređaje pružajući zbirku sintakse za dizajn predložaka. Drugim riječima Bootstrap pomaže web programerima da brže grade web stranice jer ne moraju brinuti o osnovnim naredbama i funkcijama. Sastoji se od HTML-a, CSS-a i JS-a za različite funkcije i komponente vezane uz

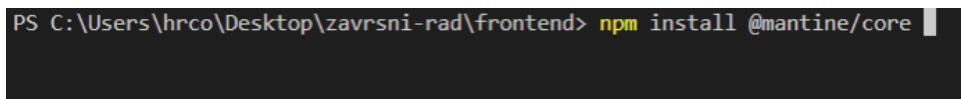
web dizajn. Primarni cilj Bootstrapa je kreiranje responzivnih web-mjesta za mobilne uređaje. Osigurava da svi elementi sučelja web stranice rade optimalno na svim veličinama zaslona. Između ostalog on je dostupan u dvije varijante: unaprijed kompilirano i na temelju verzije izvornog koda. Također se može instalirati sa paketom koji nudi upravljanje i ažuriranje okvira (engl. *Framework*) i biblioteka.

### 3.4.5. Mantine

Mantine [10] je biblioteka React Komponenti usmjerena na pružanje izvrsnog korisničkog i razvojnog iskustva. Razvoj Mantinea započeo je u siječnju 2021., a verzija 1.0 objavljena je 3. svibnja 2021. Svaka Mantine komponenta podržava nadjačavanje stilova za svaki unutarnji element i klasama ili ugrađenim stilovima. Ova značajka uz druge mogućnosti prilagodbe omogućuje implementaciju bilo kakvih vizualnih izmjena komponenti i njihovu prilagodbu kako bi odgovarale gotovo svim zahtjevima dizajna.

### 3.4.6. Postavljanje Mantinea

Najbolji način za korištenje Mantine komponenti je putem *npm* paketa koji može instalirati pomoću naredbe prikazano slikom 3.4.



```
PS C:\Users\hrco\Desktop\završni-rad\frontend> npm install @mantine/core
```

*Slika 3.4. Instalacija Mantine/core naredbom npm*

Navedena biblioteka nakon instalacije prikazano slikom 3.4. trebala bi se pojaviti u našem *package.json*, što možemo vidjeti na slici 3.5. Nakon uspješne instalacije možemo je koristiti na način da dohvatimo biblioteku naredbom u liniji 1 prikazano slikom 3.6.

```
package.json M X
frontend > package.json > {} dependencies > firebase
1  {
2    "name": "frontend",
3    "version": "0.1.0",
4    "private": true,
5    "scripts": {
6      "start": "react-scripts start",
7      "build": "react-scripts build",
8      "test": "react-scripts test",
9      "lint": "eslint src",
10     "lint:fix": "eslint --fix src",
11     "eject": "react-scripts eject"
12   },
13   "dependencies": {
14     "@coreui/icons": "^2.1.0",
15     "@coreui/react": "^4.3.0",
16     "@mantine/core": "^5.0.2",
17     "@mantine/dates": "^5.0.2",
18     "@mantine/form": "^5.0.2",
19     "@mantine/hooks": "^5.0.2",
20     "@mantine/modals": "^5.0.2",
21     "@mantine/notifications": "^5.0.2",
22     "bootstrap": "^5.1.3",
23     "clsx": "^1.1.1",
24     "dayjs": "^1.11.3",
25     "firebase": "^9.8.2",
26     "joi": "^17.6.0",
27     "react": "^18.1.0",
28     "react-dom": "^18.1.0",
29     "react-router-dom": "^6.3.0",
30     "tabler-icons-react": "^1.52.0"
31   },

```

*Slika 3.4 prikaz package.json*

```
ConfirmAcceptModal.jsx U
frontend > src > components > shared > Modal > ConfirmAcceptModal.jsx > ConfirmAcceptModal
1  import { Button, Group, LoadingOverlay, Modal, Text } from "@mantine/core";
2  import React, { useState } from "react";

```

*Slika 3.5. naredba za dohvaćanje komponenti iz mantine/core biblioteke*

Sada možemo koristiti komponente na način prikazan na slici 3.6.

```

export function ConfirmAcceptModal({ opened, title, description, confirmLabel, onConfirm, onClose }) {
  const [loading, setLoading] = useState(false);

  return (
    <Modal title={title} centered opened={opened} onClose={onClose}>
      <LoadingOverlay visible={loading} />
      <Text size="sm" my="xl">
        {description}
      </Text>
      <Group position="right">
        <Button variant="subtle" onClick={onClose}>
          Odustani
        </Button>
        <Button
          color="green"
          onClick={() => {
            setLoading(true);
            onConfirm().finally(() => setLoading(false));
          }}
        >
          {confirmLabel ?? "Potvrdi"}
        </Button>
      </Group>
    </Modal>
  );
}

```

**Slika 3.6.** Prikaz programskog koda korištenjem Mantine komponenti

### 3.4.7. React

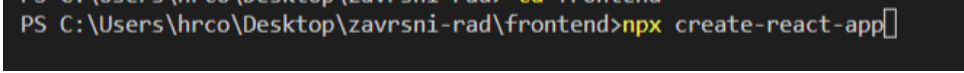
React je deklarativan, učinkovita i fleksibilna JavaScript biblioteka za izgradnju korisničkog sučelja. Omogućuje nam sastavljanje složenih korisničkih sučelja (UI-ja) od malih i izoliranih dijelova koda zvanih “komponente”. *React.js* je objavio softverski inženjer koji radi za Facebook – Jordane Walke 2011. Koristi se za rukovanje slojem prikaza i može se koristiti za web i mobilne aplikacije. Glavni cilj Reacta je biti opsežan, brz, deklarativan, fleksibilan i jednostavan. React nije okvir (engl. *Framework*), već posebna biblioteka. Objašnjenje za to je da se React bavi samo renderiranjem korisničkih sučelja i zadržava mnoge stvari prema nahođenju pojedinačnih projekata. Standardni skup alata za kreiranje aplikacije pomoću React.js-a često se naziva stogom (engl. *Stack*). Nadalje React nudi neke izvanredne značajke koje ga čine najšire prihvaćenom bibliotekom za razvoj Frontend aplikacije. Jedna od njih je JSX koji predstavlja JavaScript sintaktičko proširenje. To je izraz koji se koristi u Reactu da opiše kako bi korisničko sučelje trebalo izgledati. Može pisati HTML strukture u istoj datoteci kao JavaScript kod koristeći XML. React se bavi kontroliranjem stanja i prikazivanjem isto tog u DOM-u. Virtualni DOM (VDOM) je programski koncept u kojem se idealna, ili virtualna reprezentacija korisničkog sučelja čuva u memoriji i sinkronizira s pravim DOM-om pomoću biblioteke kao što je ReactDOM. Ovaj pristup omogućuje deklarativni API Reacta, tako da kaže Reactu u kojem stanju želi da korisničko sučelje bude, a on osigurava da DOM odgovara tom stanju. U React svijetu pojava “virtualni DOM” obično se povezuje s React elementima budući da su oni objekti koji predstavljaju korisničko sučelje. React, međutim, također koristi



interne objekte zvane “*fibers*” za držanje dodatnih informacija o stablu. Oni se također mogu smatrati dijelom implementacije “Virtualnog DOM-a” u Reactu.

### 3.4.8. Kreiranje React aplikacije

“Create React App” [11] ugodno je okruženje za učenje Reacta i najbolji je način za početak izrade nove aplikacije. Ono ne rukuje pozadinskom logikom ili bazama podataka, već samo stvara putanju za izradu Frontenda, tako da ga možemo koristiti s bilo kojim pozadinskim dijelom koji želimo. Skripta generira sve potrebne datoteke i mape za pokretanje React aplikacije. Naredbom prikazanom slikom 3.7. u Frontend datoteci kreirat ćemo sve potrebne alate i datoteke za korištenje i pokretanje React aplikacije.



```
PS C:\Users\hrco\Desktop\završni-rad\frontend>npx create-react-app
```

*Slika 3.7 naredba za kreiranje React aplikacije*

## 3.5. Backend dio aplikacije

Stražnja strana aplikacije odnosno poznatija pod imenom “Backend” odnosi se na dijelove računalne aplikacije ili koda koji mu omogućuje rad i kojim korisnik ne može pristupiti. Većina podataka operativne sintakse pohranjuju se i pristupaju im u stražnjem dijelu aplikacije iliti računalnog sustava. Kod se obično sastoji od jednog ili više programskih jezika. Za izradu naše aplikacije koristit ćemo *Express.js* okvir (engl. *Framework*) temeljen na *Node.js* okruženju (engl. *Environment*)

### 3.5.1. Node.js

Node.js [12] je “open-source”, “single-threaded” i “cross-platform runtime” okruženje za izvršavanje JavaScript koda izvan preglednika. Moramo imati na umu da *Node.js* nije okvir i nije programski kod. On se prvenstveno koristi za poslužitelje koji ne blokiraju, a upravljaju događajima, zbog svoje prirode s jednom niti (engl. *Single-threaded*). Kada *Node.js* izvrši operaciju, poput čitanja s mreže, pristupa bazi podataka ili datotečnom sustavu, umjesto da blokira nit i troši CPU cikluse na čekanje, *Node.js* će nastaviti s operacijama kada se odgovor vrati. Zbog toga *Node.js* može s lakoćom obrađivati više istovremenih zahtjeva klijenta. Često koristimo *Node.js* za izgradnju pozadinskih usluga poput API-ja kao što su web aplikacije ili

mobilne aplikacije. U Proizvodnji ga koriste velike tvrtke kao što su Paypal, Uber, Netflix, Walmart, itd.

### 3.5.2. Express.js

*Express.js* je okvir (engl. *Framework*) koji se nalazi na vrhu funkcionalnosti web poslužitelja Node.js kako bi pojednostavio svoje API-je i dodao korisne nove značajke. Nadalje on je “unopinionated framework”, što znači da programerima omogućuje totalnu slobodu strukturiranja koda kako žele, umjesto da forsiraju određenu strukturu koda. To možemo primijeniti pri korištenju među programa (engl. *Middleware*). Među program omogućuje izvođenje operacije na zahtjev i odgovorima koji se kreću kroz rute i uvelike se koriste u Express aplikacijama. On se može primijeniti i na razini aplikacije i na razini rute, kao i biti povezan zajedno. Što nam omogućuje lakše upravljanjem kodom.

### 3.5.3. Postavljanje express.js na server

Kako bi dodali *express.js* na naš server moramo prvo imat instaliran *Node.js* kako bi mogli pokrenuti uopće *Express.js*. Nakon toga jednostavnom naredbom u terminalu (slika 3.8) dodajemo *express.js* u našu datoteku [13].

```
PS C:\Users\hrco\Desktop\zavrzni-rad\backend> npm install express
```

*Slika 3.8 instalacija express.js u backend datoteku*

Nakon toga da bi nam bilo lakše pokretanje koda koristit ćemo vanjski alat *nodemon*, kojeg možemo dodati narednom (slika 3.9).

```
PS C:\Users\hrco\Desktop\zavrzni-rad\backend> npm install -g nodemon
```

*Slika 3.9. instalacija nodemon alata*

Nadalje, nakon što smo instalirali *nodemon* u *package.jsonu* u odjeljku “scripts” dodajemo skriptu oznaku *dev* (slika 3.10) sa komandom *nodemon* i dodajemo koju datoteku želimo pokrenuti pod tom komandom, u ovom slučaju *server.js*.

```

package.json x
backend > package.json > {} scripts > formatwrite
1
2 {
3   "name": "backend",
4   "version": "1.0.0",
5   "description": "",
6   "main": "server.js",
7   "scripts": {
8     "test": "echo \\Error: no test specified\\ && exit 1",
9     "format:check": "prettier --check \\.api/**/*\.js\\ \\.config/**/*\.js\\ \\.middleware/**/*\.js\\",
10    "format:write": "prettier --write \\.api/**/*\.js\\ \\.config/**/*\.js\\ \\.middleware/**/*\.js\\",
11    "lint:check": "eslint src",
12    "lint:fix": "eslint --fix src",
13    "dev": "nodemon server"
14  },
15  "keywords": [],
16  "author": "",
17  "license": "ISC",
18  "dependencies": {
19    "cors": "^2.8.5",
20    "dotenv": "^16.0.1",
21    "express": "^4.18.1",
22    "firebase": "^9.8.2",
23    "firebase-admin": "^10.2.0",
24    "joi": "^17.6.0",
25    "node-fetch": "^1.7.3",
26    "uuid": "^8.3.2"
27  },
28  "devDependencies": {
29    "eslint": "^8.18.0",
30    "eslint-config-prettier": "^8.5.0",
31    "nodemon": "^2.0.16",
32    "prettier": "^2.7.1"
33  }
}

```

Slika 3.10 prikaz package.json u backend datoteci

Naposljetku u datoteci *server.js* postavljamo server na način da uz pomoć “require” metode dohvaćamo *express* funkciju te ju dodajemo u novu varijablu *app*. Sada uporabom varijable *app* možemo dohvatiti *express* funkcije koje želimo, Ono što nam treba za postavljanje servera na određeni “port”, je funkcija *listen* (“port”) (slika 3.11).

```

ProtectedRoute.jsx M x RedirectHandler.jsx M server.js M x
backend > server.js > ...
1 const bodyParser = require("body-parser");
2 const cors = require("cors");
3 const express = require("express");
4
5 const doctorsRouter = require("./api/doctors/doctors-router");
6 const referralsRouter = require("./api/referrals/referrals-router");
7 const reservationsRouter = require("./api/reservations/reservations-router");
8 const rolsRouter = require("./api/roles/roles-router");
9 const usersRouter = require("./api/users/users-router");
10 const authenticationMiddleware = require("./middleware/authentication.middleware");
11 const errorHandler = require("./middleware/error-handler");
12
13 const app = express();
14
15 app.use(cors());
16 app.use(bodyParser.json());
17
18 app.use(authenticationMiddleware);
19
20 app.use("/api/roles", rolsRouter);
21 app.use("/api/users", usersRouter);
22 app.use("/api/reservations", reservationsRouter);
23 app.use("/api/referrals", referralsRouter);
24 app.use("/api/doctors", doctorsRouter);
25 app.use(errorHandler);
26
27 app.listen(5000, () => {
28   console.log("Server started on port 5000");
29 });
30

```

Slika 3.11 server.js datoteka

## **3.6. Baza podataka**

U ovome radu kako bi sačuvali negdje svoje podatke koje trebamo koristiti, usporediti, te raditi s njima potreban nam je neki vanjski spremnik koji će zadržati za nas te podatke. U ovoj web aplikaciji koristit ćemo Google Firebase bazu podataka [14]. Google Firebase je softver za razvoj aplikacija kojeg podržava Google i koji razvojnim programerima omogućuje razvoj iOS, Android i web aplikacija. Firebase pruža alate za praćenje analitike, izvješćivanje i popravljavanje padova aplikacija, kreiranje marketinga i eksperimenta s proizvodom. Firebase je kategoriziran kao NoSQL program baze podataka, koji pohranjuje podatke u dokument nalik JSON-u. Firebase nudi različite usluge za korisnika. Za završni rad koristit ćemo Authentication i RealTime Database.

### **3.6.1. Autentifikacija**

U današnje vrijeme svaka aplikacija ima neku vrstu registracije odnosno prijave bilo to preko emaila, Googlea, broja mobitela ili Facebooka. Poznavanje identiteta korisnika omogućuje aplikaciji da sigurno sprema korisničke podatke u nekakvi oblak i pruža isto personalizirano iskustvo na svim uređajima korisnika. Između ostalog Firebase authentication pruža pozadinske usluge, SDK-ove jednostavne za korištenje i gotove biblioteke za provjeru autentičnosti korisnika u našoj aplikaciji, te brojne druge usluge za lakše i sigurnije korištenje aplikacije.

### **3.6.2. Baza podataka u stvarnom vremenu**

Nakon završetka autentifikacije i provjere korisnika, trebamo moći spremiti negdje njegove podatke, za to će nam pomoći NoSQL baza podataka RealTime Database. Podaci se sinkroniziraju na svim klijentima u stvarnom vremenu i ostaju dostupni kada se naša aplikacija isključi. Nadalje podaci se pohranjuju kao JSON i sinkroniziraju u stvarno vremenu sa svakim povezanim klijentom. Nevezano koliko platformi koristimo bilo da je Android ili JavaScript SDK-ovima, svi naši klijenti dijele jednu instancu baze podataka u stvarnom vremenu i automatski primaju ažuriranja s najnovijim podacima.

### 3.6.3. Postavljanje baze podataka

Prvo što moramo napraviti kako bi koristili Firebase je kreirati projekt u Firebaseu za našu aplikaciju. Nakon kreiranja treba dodati Firebase u našu aplikaciju, to možemo postići naredbom u terminalu (slika 3.12).

```
PS C:\Users\hrco\Desktop\zavrnsi-rad\frontend> npm install --save firebase
```

*Slika 3.12 dodavanje firebase-a u frontend datoteku*

Nadalje kreirat ćemo *firebase.js* datoteku u kojem ćemo spojiti našu aplikaciju sa kreiranim Firebase projektom (slika 3.13)

```
frontend > src > services > . firebase.js > ...
1 import { initializeApp } from "firebase/app";
2 import { getAuth } from "firebase/auth";
3 import { getDatabase, ref } from "firebase/database";
4
5 const firebaseConfig = {
6   apiKey: "AIzaSyAjI4ybodxUIjP2Q_NZ0dSjyMB8gLqZzvU",
7   appId: "1:456680698935:web:c35a68c11f739cbe117bb2",
8   projectId: "auth-zavrnsi-rad-35575",
9   authDomain: "auth-zavrnsi-rad-35575.firebaseio.com",
10  storageBucket: "auth-zavrnsi-rad-35575.appspot.com",
11  messagingSenderId: "456680698935",
12  databaseURL: "https://auth-zavrnsi-rad-35575-default-rtdb.firebaseio.com/"
13 };
14
15 const firebaseApp = initializeApp(firebaseConfig);
16
17 export const firebaseAuth = getAuth(firebaseApp);
18 export const firebaseDb = getDatabase(firebaseApp);
19 export const firebaseDbRef = ref(firebaseDb);
20
```

*Slika 3.13 Prikaz firebase.js datoteke*

Nakon što smo omogućili daljnje korištenje varijable *auth* i *app* postavljanjem “export”. Kreirat ćemo još jednu datoteku pod nazivom *AuthContext.js* koja će nam služiti za dohvaćanje svim mogućih funkcija iz Firebase koje će nam trebati za uspješnu radnju naše aplikacije. Iz mape *firebase/auth* (slika 3.14) dohvatit ćemo potrebne funkcije za kreiranje jedinstvenog korisničkog računa.

```
AuthContext.jsx M
frontend > src > stores > AuthContext.jsx > ...
1 import {
2   createUserWithEmailAndPassword,
3   onAuthStateChanged,
4   sendPasswordResetEmail,
5   signInWithEmailAndPassword,
6   updateProfile } from "firebase/auth";
7 import React, { createContext, useContext, useEffect, useMemo, useState } from "react";
8 import { useNavigate } from "react-router-dom";
9 import { appConfig } from "../config/appConfig";
10 import { firebaseAuth } from "../services/firebase";
11 import { Role } from "../utils/constants";
12
13 const AuthContext = createContext();
14
15 export function useAuthData() {
16   const context = useContext(AuthContext);
17   if (context === undefined) {
18     throw new Error("useAuthData must be used within a AuthProvider");
19   }
20   return context;
21 }
22
23 let signUpInProgress = false;
24
25 export function AuthProvider({ children }) {
26   const [user, setUser] = useState(null);
27   const [userRole, setUserRole] = useState(null);
28   const [userLoading, setUserLoading] = useState(true);
29
30   const navigate = useNavigate();
31
32   // Subscribe to authentication method on Firebase
33   useEffect(() => {
34     const unsubscribe = onAuthStateChanged(firebaseAuth, (currentUser) => {
35       if (currentUser) {
36         setUser({
37           uid: currentUser.uid,
38           email: currentUser.email,
39           displayName: currentUser.displayName,
40           photoURL: currentUser.photoURL,
```

Slika 3.14. prikaz *AuthContext.js* datoteke, dohvaćanje funkcija iz *firebase/auth*

Kako bi mogli koristiti funkcije koje ćemo implementirati u datoteci *AuthContext*, pomoći će nam “Hook” *useContext* i *createContext*, tako što ćemo napraviti varijablu *userContext* (slika 3.14, linija 13) kojoj ćemo dodati funkciju *createContext*. Nadalje kreiranjem funkcije *AuthProvider* implementirat ćemo sve potrebne funkcije koje smo dohvatili iz *firebase/auth*. Također ćemo radi ljepšeg izgleda dodati dodatnu varijablu *value* u kojoj ćemo ubaciti sve potrebne funkcije koje želimo koristiti dalje u drugim datotekama. Ubacivanjem *value* varijable u *UserContext.Provider* (slika 3.15) preko *UserAuth* možemo pristupiti svim funkcijama u ovoj datoteci. Za olakšavanje rada s varijablom *value* koristit ćemo „Hook“ metodu *useMemo*, koja se aktivira svaki put kada se ažurira jedna od njegovih ovisnosti.

```
AuthContext.jsx M
frontend > src > stores > AuthContext.jsx > ...
122     Authorization: `Bearer ${newUser.accessToken}`
123   }
124   });
125 };
126
127   const signInUser = (email, password) => {
128     return signInWithEmailAndPassword(firebaseAuth, email, password);
129   };
130
131   const signOut = () => {
132     setUser(null);
133     setUserRole(null);
134     return firebaseAuth.signOut();
135   };
136
137   const resetPassword = (email) => {
138     return sendPasswordResetEmail(firebaseAuth, email);
139   };
140
141   const finishUserSignUp = () => {
142     signUpInProgress = false;
143   };
144
145   const value = useMemo(
146     () => ({
147       user,
148       userRole,
149       userLoading,
150       signUpUser,
151       signInUser,
152       signOut,
153       resetPassword,
154       finishUserSignUp
155     }),
156     [user, userRole, userLoading]
157   );
158
159   console.log("AuthContext - user", user, userRole, userLoading);
160   return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
161 }
```

Slika 3.15 AuthContext.js datoteka, prosljeđivanje funkcija uz pomoć UserAuth varijable

Nakon ovog postavljanja dohvaćanjem naredbe *UserAuth* možemo pristupiti svim funkcijama u *AuthContext.js* datoteci. Kako bi zaštitili odnosno administrirali naš Firebase, kako neko drugi ne bi mogli dirati našu bazu podataka, u našem *backend* mapi kreirat ćemo *config* mapu u koju ćemo ubaciti *firebase-config.js* datoteku (slika 3.16).

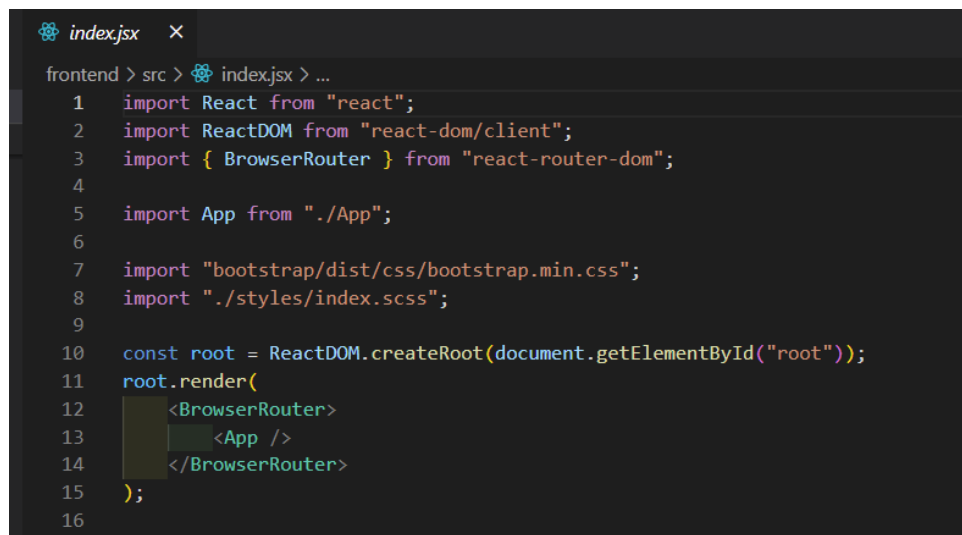
```
firebase-config.js X
backend > config > firebase-config.js > ...
1  const admin = require("firebase-admin");
2  const serviceAccount = require("../serviceAccount.json");
3
4  admin.initializeApp({
5    credential: admin.credential.cert(serviceAccount),
6
7    databaseURL: "https://auth-zavrsni-rad-35575-default-rtdb.firebaseio.com/"
8  });
9
10 module.exports = admin;
11
```

Slika 3.16 datoteka firebase-config.js

## 4. IZRADA APLIKACIJE ZA REZERVACIJU TERMINA KOD DOKTORA SPECIJALISTA

Kako bi olakšali strukturu koda, te bolju organizaciju razdijelit ćemo naš rad u dvije mape *backend* koji će sadržavati *Express.js* i *frontend* koji će sadržavati React dio aplikacije. Nakon postavljanja servera i svih ostalih dodatnih alata objašnjeni u prethodnim poglavljima spremni smo za izradu aplikacije.

Prvi dio aplikacije sastojat će se od tri stranice koje predstavljaju Sign in, Sign up i Forgot Password. Pošto već vidimo da će nam trebati nekakve rute kako bi mogli pristupiti svakoj stranici trebamo našu cijelu aplikaciju zamotati u *BrowserRouter* (slika 4.1).



```
index.jsx ×
frontend > src > index.jsx > ...
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3  import { BrowserRouter } from "react-router-dom";
4
5  import App from "./App";
6
7  import "bootstrap/dist/css/bootstrap.min.css";
8  import "./styles/index.scss";
9
10 const root = ReactDOM.createRoot(document.getElementById("root"));
11 root.render(
12   <BrowserRouter>
13     <App />
14   </BrowserRouter>
15 );
16
```

*Slika 4.1 index.jsx datoteka*

Shodno tome, kreirat ćemo datoteku *app.jsx* koja sadrži pozive svih stranica i njihove rute (slika 4.2). Sada imamo putanju do naših stranica. U našoj *src* mapi kreirat ćemo dodatnu *views* mapu koja će nam sadržavati sve potrebne stranice naše aplikacije. Prvu stranicu koju ćemo napraviti je *SignUpPage.jsx* koja će nam predstavljati registraciju korisnika (slika 4.3).



```

App.jsx M X
frontend > src > App.jsx > ...
1 import { MantineProvider } from "@mantine/core";
2 import { NotificationsProvider } from "@mantine/notifications";
3 import "dayjs/locale/hr";
4 import dayjs from "dayjs";
5 import React from "react";
6 import { Route, Routes } from "react-router-dom";
7 import ReferralPage from "../views/FamilyDoctor/ReferralPage";
8 import { ProtectedRoute } from "../routing/ProtectedRoute";
9 import { RedirectHandler } from "../routing/RedirectHandler";
10 import { AuthProvider } from "../stores/AuthContext";
11 import { ForgotPasswordPage } from "../views/Auth/ForgotPasswordPage";
12 import { SignInPage } from "../views/Auth/SignInPage";
13 import { SignUpPage } from "../views/Auth/SignUpPage";
14 import { DoctorsListPage } from "../views/Doctor/DoctorsListPage";
15 import { PatientAppointmentsPage } from "../views/Patient/PatientAppointmentsPage";
16 import { ProfilePage } from "../views/User/ProfilePage";
17 import RequestsPages from "../views/SpecialistDoctor/RequestsPages";
18 import NewDoctorsReferral from "../views/FamilyDoctor/NewDoctorsReferral";
19
20 // Set dayjs locale globally
21 dayjs.locale("hr");
22
23 export default function App() {
24   return (
25     <MantineProvider
26       theme={{
27         fontFamily: "Inter, sans-serif",
28         primaryShade: 9
29       }}
30       withGlobalStyles
31       withNormalizeCSS
32     >
33       <NotificationsProvider position="top-right">
34         <AuthProvider>
35           <Routes>
36             <Route exact path="/" element={ <RedirectHandler /> } />
37             <Route path="/signin" element={ <SignInPage /> } />
38             <Route path="/signup" element={ <SignUpPage /> } />
39             <Route path="/forgot-password" element={ <ForgotPasswordPage /> } />
40             <Route

```

Slika 4.2 app.jsx datoteka

```

SignUpPage.jsx M X
frontend > src > views > Auth > SignUpPage.jsx > SignUpPage
62
63   return (
64     <Container size={600} my={120}>
65       <Center mb="lg">
66         <Brand size="lg" color="#555" />
67       </Center>
68       <Paper withBorder shadow="md" p={30} radius="md">
69         <Title order={2} align="center" mb={32}>
70           Registracija korisnika
71         </Title>
72
73         <error && {
74           <Alert icon={ <AlertCircle size={16} /> } color="red" mb="lg">
75             {error}
76           </Alert>
77         }
78
79         <form onSubmit={form.onSubmit(values) => handleSubmit(values)}>
80           </> <group direction="column" grow />
81           <TextInput {...form.getInputProps("name")} label="Ime" placeholder="Vaše ime" />
82           <TextInput {...form.getInputProps("phone")} label="Brod" mt="md" placeholder="Vaš broj" />
83           <TextInput {...form.getInputProps("email")} required label="Email" mt="md" placeholder="primjer@email.hr" />
84           <PasswordInput {...form.getInputProps("password")} required label="Lozinka (minimalno 8 znakova)" mt="md" placeholder="Vaša lozinka" />
85           <PasswordInput {...form.getInputProps("repeatPassword")} required label="Ponovljena lozinka" mt="md" placeholder="Ponovite vašu lozinku" />
86           </> </group />
87
88           <Button type="submit" fullWidth mt="xl" loading={loading}>
89             Pošalji podatke
90           </Button>
91
92           <Text size="sm" align="center" mt="lg">
93             Već imate korisnički račun? (" ")
94             <Anchor component={Link} to="/signin" size="sm">
95               Prijavi se
96             </Anchor>
97           </Text>
98         </form>
99       </Paper>
100     </Container>

```

Slika 4.3 prikaz koda za kreiranje izgleda sign up stranice

Logika oko rada aplikacije je da prvo uz pomoć "hook-a useForm" koji nam nudi Mantinea kreiramo dva stanja za dohvaćanje i postavljanje stanja odnosno naših komponenti za registraciju (slika 4.4). Nadalje uz pomoć Joi direktorija napraviti ćemo validaciju unosa naših komponenti u registraciji (slika 4.5)

```

SignUpPage.jsx M X
frontend > src > views > Auth > SignUpPage.jsx > SignUpPage
27     }).with("password", "repeatPassword");
28
29     export function SignUpPage() {
30         const { signUpUser, finishUserSignUp } = useAuthData();
31         const navigate = useNavigate();
32
33         const [error, setError] = useState("");
34         const [loading, setLoading] = useState(false);
35
36         const form = useForm({
37             schema: joiResolver(schema),
38             initialValues: {
39                 name: "",
40                 email: "",
41                 password: "",
42                 repeatPassword: "",
43                 mbo: ""
44             }
45         });
46

```

Slika 4.4. logika oko rada signup stranice

```

SignUpPage.jsx M X
frontend > src > views > Auth > SignUpPage.jsx > SignUpPage
1 | import { Alert, Anchor, Button, Center, Checkbox, Container, Group, NumberInput, Paper, PasswordInput, Text, TextInput, Title } from "@mantine/core";
2 | import { joiResolver, useForm } from "@mantine/form";
3 | import Joi from "joi";
4 | import React, { useState } from "react";
5 | import { Link, useNavigate } from "react-router-dom";
6 | import { AlertCircle } from "tabler-icons-react";
7
8 | import { Brand } from "../components/shared/Brand/Brand";
9 | import { useAuthData } from "../stores/AuthContext";
10
11 | const schema = Joi.object({
12 |   name: Joi.string().min(3).message("Ime mora sadržavati minimalno 3 slova"),
13 |   email: Joi.string()
14 |     .required()
15 |     .email({ minDomainSegments: 2, tlds: { allow: false } })
16 |     .message("Email nije ispravan"),
17 |   password: Joi.string().required().min(8).message("Lozinka ne sadrži potreban broj znakova: 8"),
18 |   repeatPassword: Joi.any().valid(Joi.ref("password")).required().messages({
19 |     "any.only": "Lozinke nisu jednake"
20 |   }),
21 |   mbo: Joi.string().custom((value, helper) => {
22 |     if (value.length !== 9) {
23 |       return helper.message("MBO mora imati 9 znakova");
24 |     }
25 |     return true;
26 |   })
27 | }).with("password", "repeatPassword");
28

```

Slika 4.5. Joi validacija unosa iz registracije

U našem *Form* sekciji dodali smo *onSubmit* metodu kojoj smo dodijelili *handleSubmit* funkciju (slika4.3.), kako bi pritiskom na *Button* se izvršile sve naredbe u *handleSubmit* funkciji. Logika oko nje je ta da prvo moramo provjeriti dali je lozinka uopće unesena, nakon toga ulazimo u *try/catch* blok gdje prvo pokušavamo kreirati korisnika uz pomoć naredbe *signUpUser* koji smo dohvatili iz *AuthContext* datoteke preko *UserAuth* varijable. Ako je naredba dobro izvršena, uputit će nas na sljedeću stranicu, u drugom slučaju dohvatit ćemo error te ga ispisati korisniku da registracija nije dobro uspjela. Na sličnom principu radi i *SignInPage.jsx* (slika 4.6), koji umjesto *createUser* funkcije koristimo *signInUser*.

```

@ SigninPage.jsx M X
frontend > src > views > Auth > @ SigninPage.jsx > @ SigninPage > @ handleSubmit
1 import { Alert, Anchor, Box, Button, Center, Container, Group, Paper, PasswordInput, Text, TextInput, Title } from "mantine/core";
2 import { useForm } from "mantine/form";
3 import React, { useState } from "react";
4 import { Link, useNavigate } from "react-router-dom";
5 import { AlertCircle, Heartbeat } from "tabler-icons-react";
6
7 import { Brand } from "../../components/shared/Brand/Brand";
8 import { useAuthData } from "../../stores/AuthContext";
9
10 export function SigninPage() {
11   const { signInUser } = useAuthData();
12   const navigate = useNavigate();
13
14   const [error, setError] = useState("");
15   const [loading, setLoading] = useState(false);
16
17   const form = useForm({
18     initialValues: {
19       email: "",
20       password: ""
21     }
22   });
23
24   const handleSubmit = async ({ email, password }) => {
25     setError("");
26     setLoading(true);
27
28     try {
29       await signInUser(email, password);
30       navigate("/");
31     } catch (error) {
32       setError("Uneseni podaci nisu ispravni");
33     } finally {
34       setLoading(false);
35     }
36   };
37
38   return (
39     <Container size={460} my={120}>
40       <Center mb="xl">
41         <Brand size="lg" color="#555" />
42       </Center>
43       <Paper withBorder shadow="md" p={30} radius="md">
44         <Title order={2} align="center" mb={32}>
45           Prijava korisnika
46         </Title>
47
48         {error && (
49           <Alert icons={AlertCircle} size={16} /> color="red" mb="lg">
50           {error}
51         </Alert>
52       )}
53
54       <form onSubmit={form.onSubmit((values) => handleSubmit(values))>
55         { /* <Group directions="column" grow * / }
56         <TextInput
57           {...form.getInputProps("email")}
58           required
59           label="Email"
60           onChange={(event) => {
61             setError("");
62             form.setFieldValue("email", event.currentTarget.value);
63           }}
64         />
65         <PasswordInput
66           {...form.getInputProps("password")}
67           required
68           mt="md"
69           label="Lozinka"
70           onChange={(event) => {
71             setError("");
72             form.setFieldValue("password", event.currentTarget.value);
73           }}
74         />
75         { /* </Group * / }

```

Slika 4.6. prikaz signin.jsx datoteke

Naposljetku nam ostaje kreirati *ForgotPasswordPage.jsx* stranicu (slika 4.7), kako bi mogli omogućiti našim korisnicima aplikacije da resetiraju lozinku po želji, preko svog emaila. Sama funkcija za resetiranje lozinke koju smo povukli iz Firebasea, nam omogućuje da njenim pozivom u radu omogućimo našim korisnicima putem maila kojeg koriste, da resetiraju lozinku.

```
ForgotPasswordPage.jsx X
frontend > src > views > Auth > ForgotPasswordPage.jsx > ...
46   return (
47     <Container size={480} my={120}>
48       <Center mb="xl">
49         <Brand size="lg" color="#555" />
50       </Center>
51       <Paper withBorder shadow="md" p={30} radius="md" mt="xl">
52         <Title align="center">Zaboravili ste lozinku?</Title>
53         <Container size={320}>
54           <Text color="dimmed" size="sm" align="center" mt="sm" mb="lg">
55             Unesite vaš email kako biste dobili link za resetiranje vaše lozinke.
56           </Text>
57         </Container>
58
59         {errorMessage && (
60           <Alert icon={AlertCircle size={16}} /> color="red" mb="lg">
61             {errorMessage}
62           </Alert>
63         )}
64
65         {infoMessage && (
66           <Alert icon={AlertCircle size={16}} /> mb="lg">
67             {infoMessage}
68           </Alert>
69         )}
70
71         <Form onSubmit={form.onSubmit((values) => handleSubmit(values))>
72           <TextInput {...form.getInputProps("email")} label="Vaš email" placeholder="primjer@email.hr" required />
73
74           <Group position="apart" mt="lg">
75             <Anchor component={Link} to="/signin" color="dimmed" size="sm">
76               <Center inline>
77                 <ArrowLeft size={12} />
78                 <Box ml={5}>Vrati se na prijavu</Box>
79               </Center>
80             </Anchor>
81             <Button type="submit" loading={loading}>
82               Resetiraj lozinku
83             </Button>
84           </Group>
85         </Form>

```

Slika 4.7. ForgotPassword datoteka

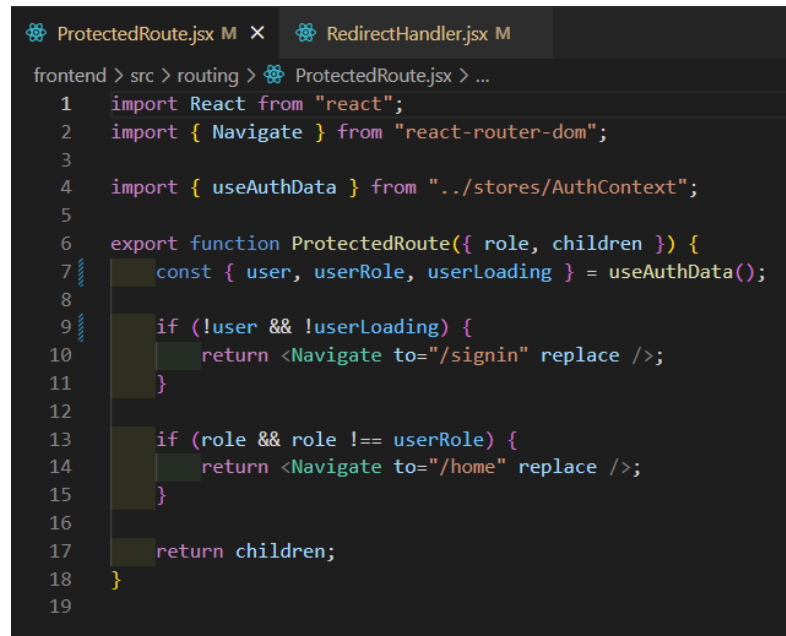
Sama logika kod *ForgotPassword* stranice (slika 4.8) je ta da smo stavili metodu *onSubmit* u *Form* za resetiranje lozinke koja pritiskom na *Button* pokreće funkciju *handleSubmit*. Dohvaćanjem funkcije *resetPassword* iz *AuthContext*, te predajom trenutnog korisničkog email-a, dobili bi u poruci link na email koja bi vodila prema resetiranju lozinke.

```
ForgotPasswordPage.jsx X
frontend > src > views > Auth > ForgotPasswordPage.jsx > ...
6   import { AlertCircle, ArrowLeft } from "tabler-icons-react";
7
8   import { Brand } from "../../components/shared/Brand/Brand";
9   import { useAuthData } from "../../stores/AuthContext";
10
11  const schema = Joi.object({
12    email: Joi.string()
13      .required()
14      .email({ minDomainSegments: 2, tlds: { allow: false } })
15      .message("Email nije ispravan")
16  });
17
18  export function ForgotPasswordPage() {
19    const { resetPassword } = useAuthData();
20
21    const [errorMessage, setErrorMessage] = useState("");
22    const [infoMessage, setInfoMessage] = useState("");
23    const [loading, setLoading] = useState(false);
24
25    const form = useForm({
26      schema: joiResolver(schema),
27      initialValues: {
28        email: ""
29      }
30    });
31
32    const handleSubmit = async ({ email }) => {
33      setErrorMessage("");
34      setLoading(true);
35
36      try {
37        await resetPassword(email);
38        setInfoMessage("Provjerite vaš email za daljnje upute");
39      } catch (error) {
40        setErrorMessage(error.message);
41      } finally {
42        setLoading(false);
43      }
44    };
45

```

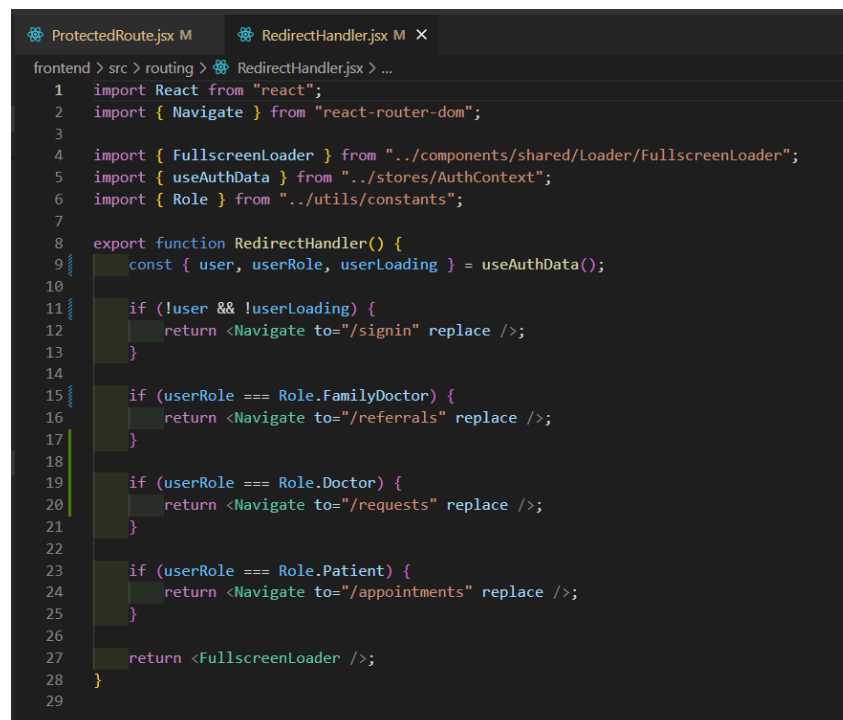
Slika 4.8 logika reset Password stranice

Naime kako ne bi omogućili korisniku da može upravljati rutama kako on hoće te omogućiti da pristupi aplikaciji a da se nije registrirao ili prijavio, moramo zaštititi rute. U tom će nam pomoći *ProtectedRoute.jsx* (slika 4.9) i *RedirectHandler.jsx* (slika 4.10), te ćemo je u *app.jsx* dohvatiti i koristiti tako što ćemo obuhvatiti rute koje želimo zaštititi da im korisnik ne može pristupiti ako prije toga nije obavio prijavu.



```
ProtectedRoute.jsx M × RedirectHandler.jsx M
frontend > src > routing > ProtectedRoute.jsx > ...
1 import React from "react";
2 import { Navigate } from "react-router-dom";
3
4 import { useAuthData } from "../stores/AuthContext";
5
6 export function ProtectedRoute({ role, children }) {
7   const { user, userRole, userLoading } = useAuthData();
8
9   if (!user && !userLoading) {
10    return <Navigate to="/signin" replace />;
11  }
12
13  if (role && role !== userRole) {
14    return <Navigate to="/home" replace />;
15  }
16
17  return children;
18 }
19
```

Slika 4.9 ProtectedRoute.jsx datoteka



```
ProtectedRoute.jsx M RedirectHandler.jsx M ×
frontend > src > routing > RedirectHandler.jsx > ...
1 import React from "react";
2 import { Navigate } from "react-router-dom";
3
4 import { FullscreenLoader } from "../components/shared/Loader/FullscreenLoader";
5 import { useAuthData } from "../stores/AuthContext";
6 import { Role } from "../utils/constants";
7
8 export function RedirectHandler() {
9   const { user, userRole, userLoading } = useAuthData();
10
11  if (user && !userLoading) {
12    return <Navigate to="/signin" replace />;
13  }
14
15  if (userRole === Role.FamilyDoctor) {
16    return <Navigate to="/referrals" replace />;
17  }
18
19  if (userRole === Role.Doctor) {
20    return <Navigate to="/requests" replace />;
21  }
22
23  if (userRole === Role.Patient) {
24    return <Navigate to="/appointments" replace />;
25  }
26
27  return <FullscreenLoader />;
28 }
29
```

Slika 4.10.RedirectHandler.jsx datoteka

Nakon uspješno obavljene prijave pritiskom da gumb bilo to za prijavu ili registraciju vodi nas u drugi dio aplikacije koji ovisi o roli korisnika koji se logira u *app*. Drugi dio aplikacije sastoji se od 5 mapa, *User* mapa sadrži datoteku za Profil stranicu, *SpecialistDoctor* sadrži *Requests* datoteku koja predstavlja zahtjeve od pacijenta, nadalje *Patient* mapa sadrži *PatientAppointemnts* gdje će bit prikazani svi njegove rezervacije, *familiDoctor* ima u sebi dvije datoteke koja jedna predstavlja sve uputnice od pacijenta a druga za kreiranje nove uputnice i *Doctor* datoteka koja predstavlja tablicu svih doktora specijalista gdje pacijent može rezervirati termin. Sama aplikacija radi na principu da klijent odnosno pacijent odlaskom kod doktora opće prakse na pregled dobiva uputnicu na svoj profil koju će ispunit doktor opće prakse. S tom uputnicom filtrirat ćemo doktore specijaliste prema njihovom poslu, ako je uputnica za dermatoverenologa prikazat će se samo ti doktori. Pacijent nakon što doktor opće prakse ispuni uputnicu može poslati zahtjev doktoru specijalistu za željeni termin. Pacijent čeka odgovor specijalista dali mu traženi termin odgovara, te potvrdom termina specijalista rezervira se termin za danog pacijenta. Pacijent u svojim rezervacijama može vidjeti dali je termin potvrđen ili nije te mu se nudi odjava termina ako nije u mogućnosti da pristupi traženom vremenu. Kako bi ovo sve omogućili te razlikovali klijenta dali je doktor opće prakse ili specijalist ili obični pacijent, te kako bi zaštitili rute u našoj aplikaciji trebamo svakom korisniku dodati rolu bilo pacijenta ili doktora, te prema tome dopustiti ograničenja svakom korisniku. Nadalje svaka uputnica će imati 6 stanja (slika 4.11), kako bi omogućili korisniku da zna u kakvom je stanju uputnica.

```
js constants.js M X
frontend > src > utils > js constants.js > ReservationStatus > Expired
1  export const Role = {
2    Doctor: "doctor",
3    FamilyDoctor: "familyDoctor",
4    Patient: "patient",
5    Admin: "admin"
6  };
7
8  export const ReservationStatus = {
9    Waiting: "waiting",
10   Approved: "approved",
11   Completed: "completed",
12   Canceled: "canceled",
13   Denied: "denied",
14   Expired: "expired"
15 };
16
```

*Slika 4.11 prikaz rola i stanja uputnica*

Ako se trenutni korisnik prijavi kao pacijent otvorit će mu se *Appointments* stranica gdje će mu biti vidljive njegove rezervacije, te mu se nudi da otkáže termin pomoću funkcije *cancelReservation* (slika 4.12), ili promijeniti vrijeme termina uz pomoć funkcije *handleBookingUpdate* (slika 4.13).

```
const cancelReservation = async (reservationId) => {
  return fetch(`${appConfig.apiUrl}/api/reservations/${reservationId}`, {
    method: "DELETE",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${user.accessToken}`
    }
  })
  .then((res) => {
    if (res.ok) {
      loadReservations();
      showSuccessNotification({ message: "Uspješno ste otkazali pregled" });
    } else {
      showErrorNotification({ message: "Dogodila se neočekivana pogreška" });
    }
  })
  .catch((error) => {
    showErrorNotification({ message: error.message });
  })
  .finally(() => {
    setModalOpened(false);
    setReservationToCancel(undefined);
  });
};
```

*Slika 4.12 funkcija cancelReservation*

```
53 const handleBookingUpdate = async ({ date, time }, doctorId) => {
54   console.log(reservationToUpdate.doctorId);
55   const reservationData = {
56     time: time,
57     date: date,
58     doctorId: reservationToUpdate.doctorId,
59     uid: user.uid,
60     reservationId: reservationToUpdate.id
61   };
62
63   try {
64     const response = await fetch(`${appConfig.apiUrl}/api/reservations/updateAppointment`, {
65       method: "PATCH",
66       body: JSON.stringify(reservationData),
67       headers: {
68         "Content-Type": "application/json",
69         Authorization: `Bearer ${user.accessToken}`
70       }
71     });
72
73     if (response.ok) {
74       showSuccessNotification({ message: "Uspješno ste rezervirali termin" });
75     } else {
76       showErrorNotification({ message: "Neočekivana greška prilikom rezervacije termina" });
77     }
78   } finally {
79     setUpdateModalOpened(false);
80     setReservationToUpdate(undefined);
81   }
82 }
```

*Slika 4.13 funkcija handleBookingUpdate*

Kao što možemo vidjeti na slikama pozivom funkcije šalјemo podatke na Backend da on odradi svoj dio posla, koji će ili obrisati termin, ili promijeniti datum termina u našoj bazi (slika 4.14) (slika 4.15).

```

113 router.patch("/updateAppointment", authorization(Role.Patient), async (req, res, next) => {
114   const { date, time, uid, doctorId, reservationId } = req.body;
115
116   const reservationData = {
117     date,
118     time,
119     doctorId
120   };
121   console.log(reservationData);
122   console.log(uid);
123   try {
124     await updateTimeAndDate(reservationData, uid, reservationId);
125     return res.status(200).send();
126   } catch (err) {
127     next(err);
128   }
129 });
130
131 const updateTimeAndDate = async (reservationData, userId, reservationId) => {
132   const existingReservationsOfAllUsers = await getExistingReservations();
133   const existingReservationsForUser = await getExistingReservationsForUser(userId);
134
135   validateReservationData(existingReservationsOfAllUsers, reservationData, existingReservationsForUser);
136
137   const reservationToUpdate = existingReservationsForUser.find((it) => it.id === reservationId);
138   if (reservationToUpdate) {
139     reservationToUpdate.date = reservationData.date;
140     reservationToUpdate.time = reservationData.time;
141   }
142
143   await db.ref("reservations").child(userId).set(existingReservationsForUser);
144 };

```

#### 4.14. promjena termina na backendu

```

131 router.delete("/:reservationId", authorization(Role.Patient), async (req, res, next) => {
132   try {
133     await cancelReservation(req.params.reservationId, req.user.uid);
134     return res.status(200).send();
135   } catch (err) {
136     next(err);
137   }
138 });
139
140 const cancelReservation = async (reservationId, userId) => {
141   const existingReservationsForUser = await getExistingReservationsForUser(userId);
142   const reservationToCancel = existingReservationsForUser.find((it) => it.id === reservationId);
143   if (reservationToCancel) {
144     reservationToCancel.status = ReservationStatus.Canceled;
145   }
146
147   await db.ref("reservations").child(userId).set(existingReservationsForUser);
148 };

```

#### Slika 4.15 otkazivanje termina na backendu

Nakon rezervacije termina pacijent čeka potvrdu od doktora specijalista dali termin odgovara ili ne. U *SpecialistDoctor* mapi dodat ćemo datoteku *requests* koja će se baviti upravo time. Doktor će u toj datoteci vidjeti sve zahtjeve od pacijenta koje se odnose na njega te će mu se nuditi da otkáže termin ili potvrdi termin uz pomoć funkcija *cancelReservation* (slika 4.16) i *confirmReservation* (slika 4.17). Ako je doktor potvrdio rezervaciju nakon pacijentovog dolaska nudi mu se gumb koji pokreće funkciju *completeReservation* kako bi u potpunosti završili taj termin, te bi na taj način znali da je termin dovršen.



```

82 const cancelReservation = async (reservationId, userId) => {
83   return fetch(`${appConfig.apiUrl}/api/reservations/doctors-cancel/${reservationId}`, {
84     method: "DELETE",
85     body: JSON.stringify({
86       patientId: userId
87     }),
88     headers: {
89       "Content-Type": "application/json",
90       Authorization: `Bearer ${user.accessToken}`
91     }
92   })
93   .then((res) => {
94     if (res.ok) {
95       loadReservations();
96       showSuccessNotification({ message: "Uspješno ste otkazali pregled" });
97     } else {
98       showErrorNotification({ message: "Dogodila se neočekivana pogreška" });
99     }
100   })
101   .catch((error) => {
102     showErrorNotification({ message: error.message });
103   })
104   .finally(() => {
105     setModalOpened(false);
106     setReservationToCancel(undefined);
107   });
108 };

```

*Slika 4.16. cancelReservation u requests datoteci*

```

54 const completeReservation = async (reservationId, userId) => {
55   return fetch(`${appConfig.apiUrl}/api/reservations/doctor-complete/${reservationId}`, {
56     method: "PATCH",
57     body: JSON.stringify({
58       patientId: userId
59     }),
60     headers: {
61       "Content-Type": "application/json",
62       Authorization: `Bearer ${user.accessToken}`
63     }
64   })
65   .then((res) => {
66     if (res.ok) {
67       loadReservations();
68       showSuccessNotification({ message: "Uspješno ste potvrdili termin" });
69     } else {
70       showErrorNotification({ message: "Dogodila se neočekivana pogreška" });
71     }
72   })
73   .catch((error) => {
74     showErrorNotification({ message: error.message });
75   })
76   .finally(() => {
77     setCompleteModalOpened(false);
78     setReservationToComplete(undefined);
79   });
80 };

```

*Slika 4.17 completeResrvation u requests datoteci*

```

27 const confirmReservation = async (reservationId, userId) => {
28   return fetch(`${appConfig.apiUrl}/api/reservations/doctor-confirm/${reservationId}`, {
29     method: "PATCH",
30     body: JSON.stringify({
31       patientId: userId
32     }),
33     headers: {
34       "Content-Type": "application/json",
35       Authorization: `Bearer ${user.accessToken}`
36     }
37   })
38   .then((res) => {
39     if (res.ok) {
40       loadReservations();
41       showSuccessNotification({ message: "Uspješno ste potvrdili termin" });
42     } else {
43       showErrorNotification({ message: "Dogodila se neočekivana pogreška" });
44     }
45   })
46   .catch((error) => {
47     showErrorNotification({ message: error.message });
48   })
49   .finally(() => {
50     setUpdateModalOpened(false);
51     setReservationToUpdate(undefined);
52   });
53 };

```

*Slika 4.18 confirmReservation u requests datoteci*

Kao što i ovdje vidimo poziv je na Backend te će on odraditi glavni dio posla (slika 4.19).

```
140 router.delete("/doctors-cancel/:reservationId", authorization(Role.Doctor), async (req, res, next) => {
141   try {
142     await doctorDeniedReservation(req.params.reservationId, req.body.patientId);
143     return res.status(200).send();
144   } catch (err) {
145     next(err);
146   }
147 });
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Slika 4.19 prikaz za odbijanje, potvrdu i dovršavanje narudžbe preko backend dijela

Da bi pacijent mogao rezervirati termin kod doktora specijalista mora mu doktor opće prakse napraviti uputnicu za danu vrstu pregleda. U mapi *familyDoctor* imamo dvije datoteke, jedna koja predstavlja sve uputnice pacijenata (slika 4.20), a druga za kreiranje nove uputnice.

```

Referral.jsx U X
frontend > src > views > FamilyDoctor > Referral.jsx > Referral
0
7 function Referral() {
8   const { data: referrals } = useAllReferralsData();
9
10  const rows = referrals.map((patientReferral) => {
11    return (
12      <tr key={` ${patientReferral.id}`}>
13        <td>{patientReferral.userMbo}</td>
14        <td>{patientReferral.userEmail}</td>
15        <td>{patientReferral.dateCreated}</td>
16        <td>{patientReferral.speciality}</td>
17        <td>{patientReferral.description}</td>
18      </tr>
19    );
20  });
21  return (
22    <Container>
23      <Title order={2} mt="lg">
24        Uputnice od pacijenata
25        <Button className={classes.button} position="right">
26          <Link to={"/newReferrals"}>kreiraj novu uputnicu</Link>
27        </Button>
28      </Title>
29      <Paper shadow="xs" p="md" mt="lg">
30        <Table sx={{ minWidth: 800 }} verticalSpacing="xs">
31          <thead>
32            <tr>
33              <th>MBO</th>
34              <th>Email</th>
35              <th>Vrijeme kreiranja</th>
36              <th>Vrsta pregleda</th>
37              <th>Opis uputnice</th>
38            </tr>
39          </thead>
40          <tbody>{rows}</tbody>
41        </Table>
42      </Paper>
43    </Container>
44  );
45

```

*Slika 4.20 Prikaz Referrals stranice*

Kreiranje nove uputnice radi na principu da doktor opće prakse preko MBO broja pronalazi pacijenta iz baze podataka (slika 4.21) , te shodno tome automatski se popunjavanju podaci o tom pacijentu izvučeni iz Firebasea. Doktoru ostaje da dovršiti uputnicu tako da će birati za čega se uputnica upućuje ,te koja je uputna dijagnoza pacijenta (slika 4.21).

```

28
29 const handleSearch = async (e) => {
30   e.stopPropagation();
31   console.log(searchValue);
32   try {
33     const mboUser = await fetch(`${appConfig.apiUrl}/api/referrals/mbo?mbo=${searchValue}`, {
34       headers: {
35         "Content-Type": "application/json",
36         Authorization: "Bearer " + token
37       }
38     });
39     .then((res) => res.json())
40     .then((data) => {
41       setName(data.name);
42       setOib(data.oib);
43       setDatum(data.dateOfBirth);
44       setAddress(data.address);
45       setEmail(data.email);
46       setTelephone(data.phoneNumber);
47       setId(data.uid);
48     });
49
50     if (!mboUser) {
51       setSearchValue('Pacijent s unesenim MBO brojem: "${searchValue}" nije pronaden');
52     }
53     setEditing(true);
54     setFoundUser(mboUser);
55   } catch (error) {
56     console.log(error);
57   }
58 };
59

```

*Slika 4.21 dohvaćanje pacijenta preko MBO broja*

```
const submitHandler = async (e) => {
  e.preventDefault();
  const current = new Date();

  const date = `${current.getFullYear()}-${current.getMonth() + 1}-${current.getDate()}`;
  console.log(id, date, speciality, description);
  const referralData = {
    uid: id,
    speciality: speciality,
    description: description,
    dateCreated: date
  };
  console.log(referralData);
  console.log(token);

  const response = await fetch(`${appConfig.apiUrl}/api/referrals/data`, {
    method: "POST",
    body: JSON.stringify(referralData),
    headers: {
      "Content-Type": "application/json",
      Authorization: "Bearer " + token
    }
  });
  if (response.ok) {
    showSuccessNotification({ message: "Uspješno ste rezervirali termin" });
  } else {
    showErrorNotification({ message: "Neočekivana greška prilikom rezervacije termina" });
  }
  setEditing(false);
};
```

*Slika 4.21 submitHandler funkcija za dovršetak uputnice*

Također kako bi kroz cijelu aplikaciju imali *header* i *footer* stavit ćemo ih u datoteku *layout* (slika 4.22). Tako samo mijenjamo unutarnji dio aplikacije a gori i donji dio ostaje cijelo vrijeme isto. Pošto imamo više rola razlikovat će nam se *header* ovisno o tome dali je pacijent, doktor opće prakse ili doktor specijalist.

```
UpdateBookingModal.jsx U  Layout.jsx ×  PatientAppointments.jsx M
frontend > src > components > layout > Layout.jsx > ...
1  import React from "react";
2
3  import { Footer } from "./Footer";
4  import { Header } from "./Header";
5
6  export function Layout(props) {
7    return (
8      <div className="layout">
9        <Header />
10       <main>{props.children}</main>
11       <Footer />
12     </div>
13   );
14 }
15
```

*Slika 4.22 prikaz layout datoteke*

Shodno tome u *headeru* ćemo dohvatit role te po tome prilagodit izgled aplikacije roli korisnika koji je ulogiran (slika 4.23)

```

14 export function Header() {
15   const { userRole } = useAuthData();
16   const { data: reservations } = useReservationsData();
17   const { data: referrals } = useAllReferralsData();
18   const isRolePatient = userRole === Role.Patient;
19   const isRoleDoctor = userRole === Role.Doctor;
20   const isRoleFamilyDoctor = userRole === Role.FamilyDoctor;
21
22   const activeReservations = reservations.filter(r => r.status === ReservationStatus.Approved || r.status === ReservationStatus.Waiting);
23   const waitingReservations = reservations.filter(r => r.status === ReservationStatus.Waiting);
24
25   return (
26     <header className={classes.header}>
27       <Brand />
28       <group spacing={64}>
29         <nav>
30           <ul>
31             <li>
32               {isRoleFamilyDoctor && (
33                 <li>
34                   <Indicator>
35                     label={
36                       <Text size="xs" weight="bold">
37                         {referrals.length}
38                       </Text>
39                     }
40                     color="orange"
41                     size={20}
42                     offset={8}
43                   >
44                     <Button size="md" component={Link} to="/referrals" className={classes.navLink}>
45                       Uputnice
46                     </Button>
47                   </Indicator>
48                 </li>
49               )}
50               {isRoleDoctor && (
51                 <li>
52                   <Indicator>
53                     label={
54                       <Text size="xs" weight="bold">
55                         {waitingReservations.length}
56                       </Text>
57                     }
58                     color="orange"
59                     size={20}
60                     offset={8}
61                   >
62                     <Button size="md" component={Link} to="/requests" className={classes.navLink}>
63                       Zahtjevi za pregled
64                     </Button>
65                   </Indicator>
66                 </li>
67               )}
68               {isRolePatient && (
69                 <li>
70                   <Button size="md" component={Link} to="/doctors" className={classes.navLink}>
71                     Doktori
72                   </Button>
73                 </li>
74               )}
75               {isRolePatient && (
76                 <li>
77                   <Indicator>
78                     label={
79                       <Text size="xs" weight="bold">
80                         {activeReservations.length}
81                       </Text>
82                     }
83                     color="orange"
84                     size={20}
85                     offset={8}
86                   >
87                     <Button size="md" component={Link} to="/appointments" className={classes.navLink}>
88                       Pregledi
89                     </Button>
90                   </Indicator>
91                 </li>
92               )}
93             </ul>
94           </nav>
95         </group>
96       </header>
97     )
98   }

```

Slika 4.23 Prikaz header datoteke

Naposljetku korisniku se nudi mogućnost da na svom profilu popuni do kraja registraciju popunjavanjem danih polja (slika 4.24), te uz pomoć *updateProfile* funkcije ažuriramo svoj profil, jedino polje za MBO i email ne može mijenjati jer je jedinstveno i prema dome razlikujemo korisnike. U profilu imamo i provjeru unosa kako ne bi dobili prazno polje od korisnika (slika 4.25).

```
export function Profile() {
  const { data: person, updateProfile } = usePersonData();

  const [editing, setEditing] = useState(false);
  const [error, setError] = useState();
  const [loading, setLoading] = useState(false);

  const form = useForm({
    schema: joiResolver(schema),
    initialValues: {
      ...person,
      dateOfBirth: dayjs(person.dateOfBirth).toDate()
    }
  });

  const handleSubmit = async (values) => {
    setError("");
    setLoading(true);

    const profileUpdateData = {
      ...values,
      dateOfBirth: dayjs(values.dateOfBirth).format("YYYY-MM-DD")
    };

    try {
      await updateProfile(profileUpdateData);
    } catch (error) {
      setError(error.message);
    } finally {
      setLoading(false);
      setEditing(false);
    }
  };
};
```

*Slika 4.24 ažuriranje profila*

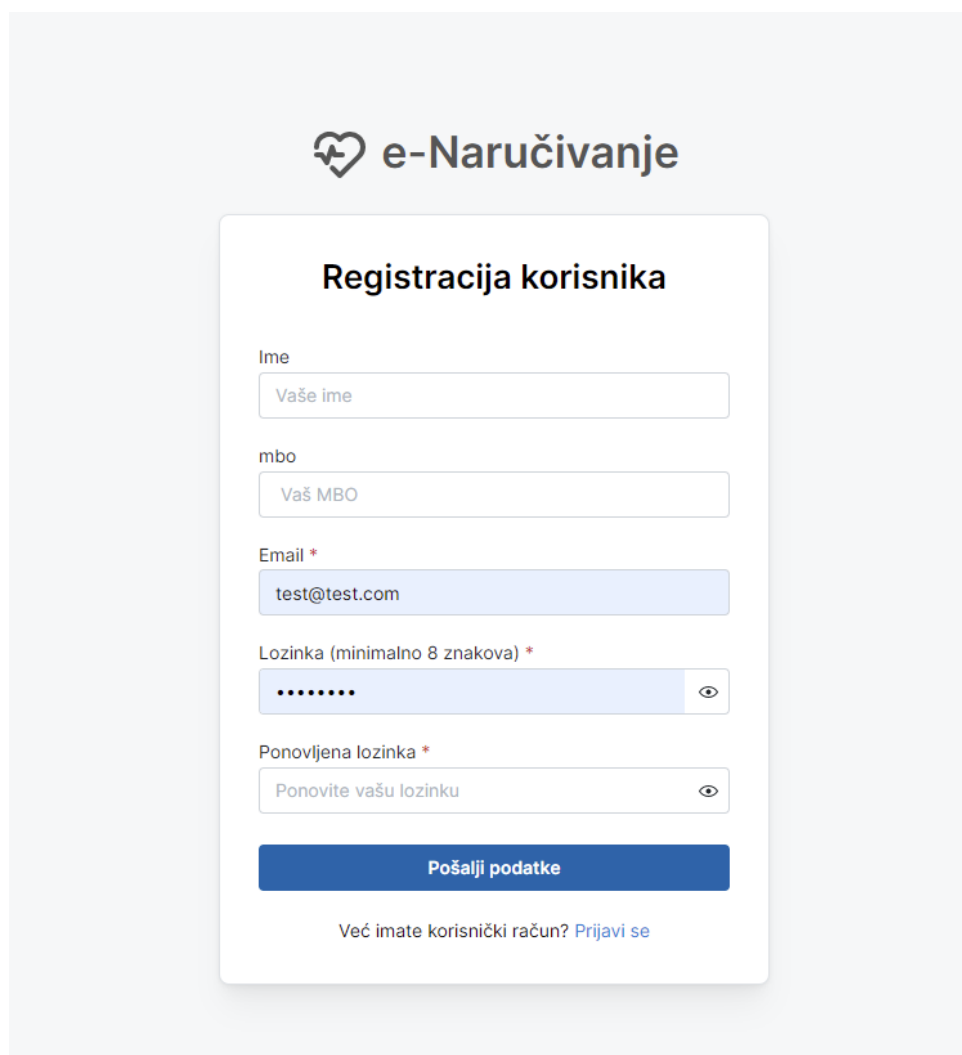
```
UpdateBookingModal.jsx U Profile.jsx M PatientAppointments.jsx M
frontend > src > views > User > Profile.jsx > ...
9 import { usePersonData } from "../../stores/PersonContext";
10 import { isOibValid } from "../../utils/validations";
11
12 const MIN_AGE_YEARS = 18;
13 const maxDatePickerDate = dayjs().startOf("day").subtract(MIN_AGE_YEARS, "years").toDate();
14
15 const schema = Joi.object({
16   name: Joi.string().min(3).message("Ime mora sadržavati minimalno 3 znaka"),
17   oib: Joi.string().custom((value, helper) => {
18     if (value.length !== 11) {
19       return helper.message("OIB mora imati 11 znakova");
20     }
21     return true;
22   }),
23   dateOfBirth: Joi.date().max(maxDatePickerDate).message("Datum rođenja nije ispravan"),
24   address: Joi.string().min(5).message("Adresa mora sadržavati minimalno 5 znakova"),
25   phoneNumber: Joi.string()
26     .min(10)
27     .message("Broj telefona mora imati minimalno 10 znakova")
28     .pattern(/^+[0-9]+$/).message("Broj telefona nije ispravnog formata")
29   }).unknown();
30
31 }
```

*Slika 4.25 validacija unosa u Profile.jsx datoteci*

## 5. IZGLED APLIKACIJE

U ovom poglavlju prikazano je i objašnjen vanjski dio aplikacije, te kako sama aplikacija radi ovisno o tome, dali je korisnik koji se prijavljuje ima rolu admina, pacijenta, doktora opće prakse ili doktora specijalist.


Pri samom otvaranju aplikacije pod nazivom e-naručivanje prikazano nam je prvi dio aplikacije koji predstavlja registraciju (slika 4.26), prijavu (slika 4.27), te resetiranje lozinke (slika 4.28).



The screenshot shows a registration form titled "Registracija korisnika" within the "e-Naručivanje" application. The form includes the following fields and elements:

- Ime:** A text input field with the placeholder "Vaše ime".
- mbo:** A text input field with the placeholder "Vaš MBO".
- Email \*:** A text input field containing "test@test.com".
- Lozinka (minimalno 8 znakova) \*:** A password input field with a masked password "....." and a visibility toggle icon.
- Ponovljena lozinka \*:** A text input field with the placeholder "Ponovite vašu lozinku" and a visibility toggle icon.
- Pošalji podatke:** A blue button to submit the registration data.
- Već imate korisnički račun? Prijavi se:** A link for existing users to log in.


*Slika 4.26 prikaz registracije korisnika*

 e-Naručivanje

### Prijava korisnika

Email \*

Lozinka \*


 

Zaboravili ste lozinku?

[Prijavi se](#)

Još nemate svoj račun? [Registriraj se](#)

*Slika 4.27 prikaz prijave korisnika*

 e-Naručivanje

### Zaboravili ste lozinku?

Unesite vaš email kako biste dobili link za resetiranje vaše lozinke.

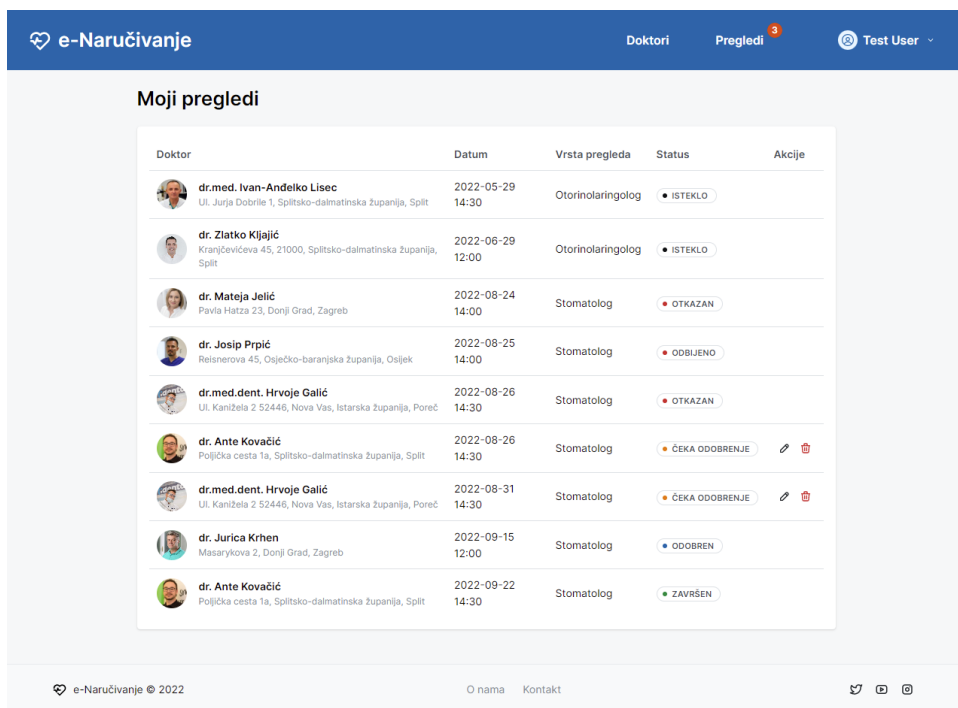
Vaš email \*














[← Vрати se na prijavu](#) [Resetiraj lozinku](#)

*Slika 4.28 prikaz resetiranje lozinke korisnika*



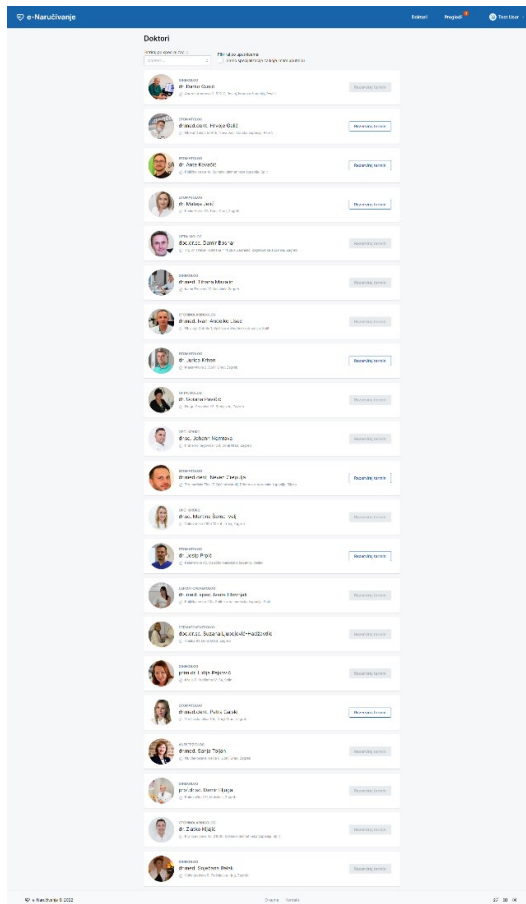
Nakon prijave ovisno o tome koja je rola korisnika ulazi u drugi dio aplikacije, točnije ako je korisnik pacijent ide na stranicu *Appointments* (slika 4.29), gdje će mu biti vidljivi rezervacije termina ako ih ima.



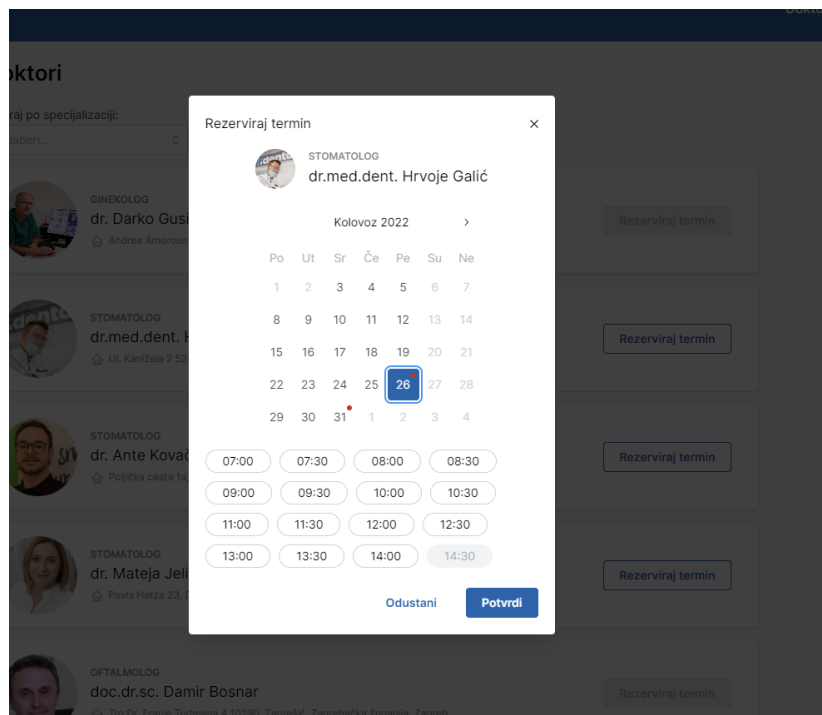
Doktor	Datum	Vrsta pregleda	Status	Akcije
 <b>dr.med. Ivan-Andelko Lisec</b> Ul. Jurja Dobrića 1, Špišsko-dalmatinska županija, Šplit	2022-05-29 14:30	Otorinolaringolog	ISTEKLO	
 <b>dr. Zlatko Kljajić</b> Krajinčevićeva 45, 21000, Špišsko-dalmatinska županija, Šplit	2022-06-29 12:00	Otorinolaringolog	ISTEKLO	
 <b>dr. Mateja Jelić</b> Pavla Horca 23, Donji Grad, Zagreb	2022-08-24 14:00	Stomatolog	OTKAZAN	
 <b>dr. Josip Prpić</b> Reisnerova 45, Osječko-baranjska županija, Osijek	2022-08-25 14:00	Stomatolog	ODBIJENO	
 <b>dr.med.dent. Hrvoje Galić</b> Ul. Kanižela 2 52446, Nova Vas, Istarska županija, Poreč	2022-08-26 14:30	Stomatolog	OTKAZAN	
 <b>dr. Ante Kovačić</b> Pojčička cesta 1a, Špišsko-dalmatinska županija, Šplit	2022-08-26 14:30	Stomatolog	ČEKA ODOBRENJE	 
 <b>dr.med.dent. Hrvoje Galić</b> Ul. Kanižela 2 52446, Nova Vas, Istarska županija, Poreč	2022-08-31 14:30	Stomatolog	ČEKA ODOBRENJE	 
 <b>dr. Jurica Krhen</b> Masarykova 2, Donji Grad, Zagreb	2022-09-15 12:00	Stomatolog	ODOBREN	
 <b>dr. Ante Kovačić</b> Pojčička cesta 1a, Špišsko-dalmatinska županija, Šplit	2022-09-22 14:30	Stomatolog	ZAVRŠEN	

*Slika 4.29 prikaz rezerviranih termina*

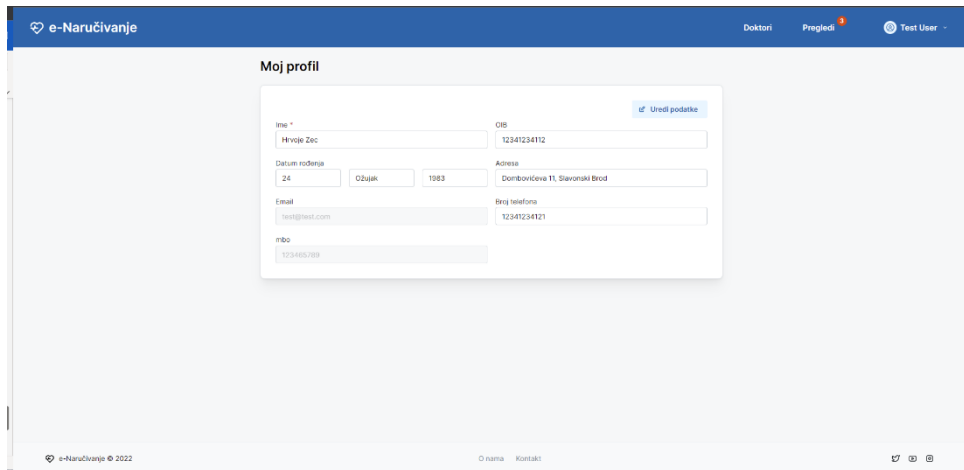
Korisniku se nude mogućnosti da rezervira termin na *doctors* stranici (slika 3.30), te će uz pomoć skočnog prozora (slika 4.31) izabrati željeni termin, nadalje korisniku se nudi opcija *Profile* (slika 4.32) gdje pacijent popunjava polja za unos, kao što su njegov OIB, broj mobitela, datum rođenja, te adresu. Polja kao što su MBO, ime korisnika i email koje je korisnik unijeo tijekom registracije ne mogu se mijenjati, te prema njima raspoznavamo razlike između pacijenta, odnosno ta polja su jedinstvena, time pacijent može dovršiti svoju prijavu, te otići kod doktora na pregled. Kako bi mogao birat kod koje vrste doktora će ići, doktor opće prakse mora ispuniti uputnicu preko *newReferral* (slika 4.33) stranice koju može otvoriti samo korisnik sa rolom *familyDoktor*. Doktoru opće prakse nudi se još i *referral* stranice gdje može vidjeti sve uputnice od pacijenata (slika 4.34)



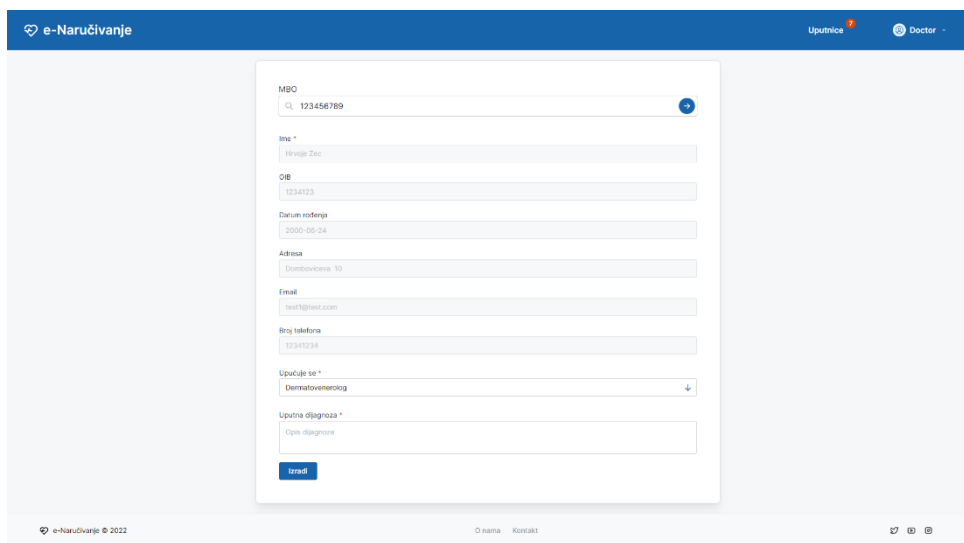
Slika 3.30 prikaz doctors stranice



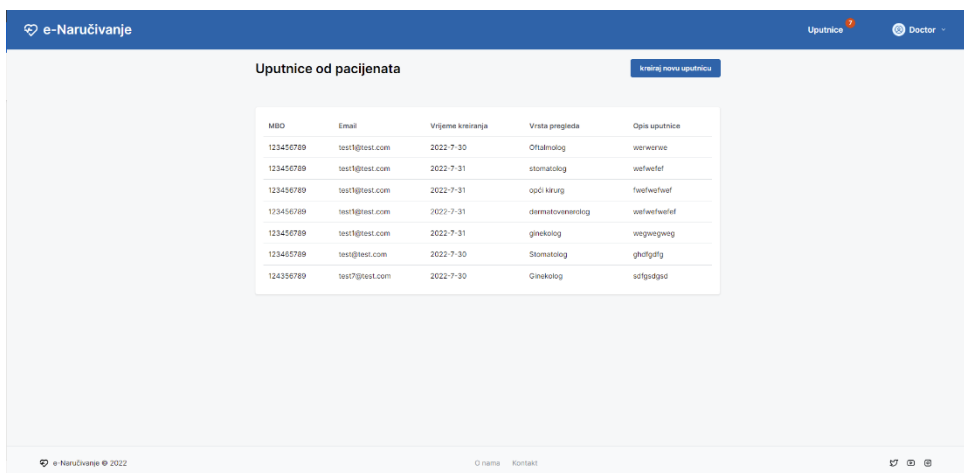
Slika 3.31 skočni prozor za popunjavanje termina



Slika 4.32 prikaz profila korisnika

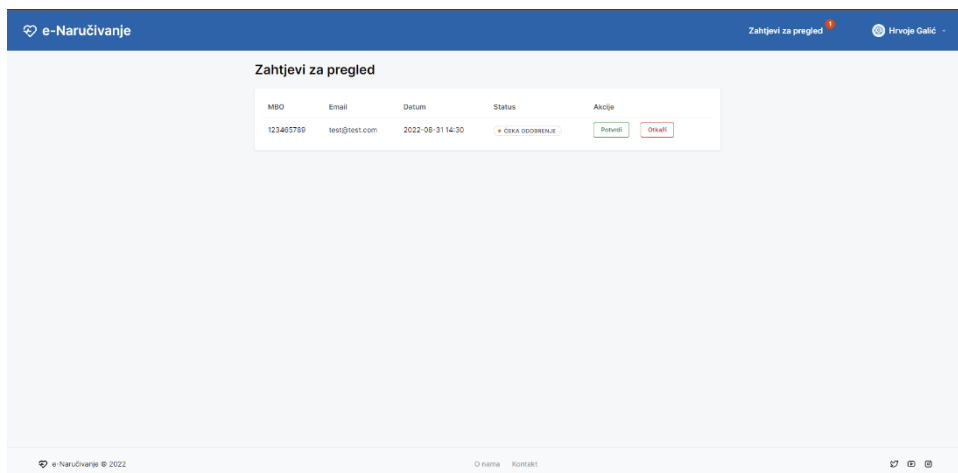


Slika 4.33 prikaz ispunjavanja uputnice



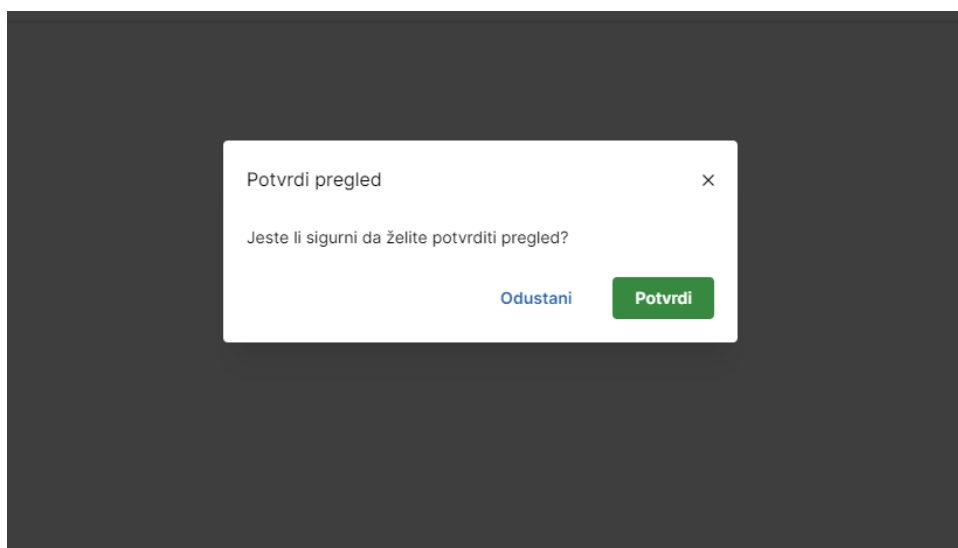
Slika 4.34 prikaz svih uputnica

Nakon što je pacijent rezervirao termin čeka odobrenje doktora specijalista da potvrdi ili odbije traženi termin. Doktoru specijalistu se preko stranice *Requests* nude svi zahtjevi od pacijenta za pregled (slika 4.35).

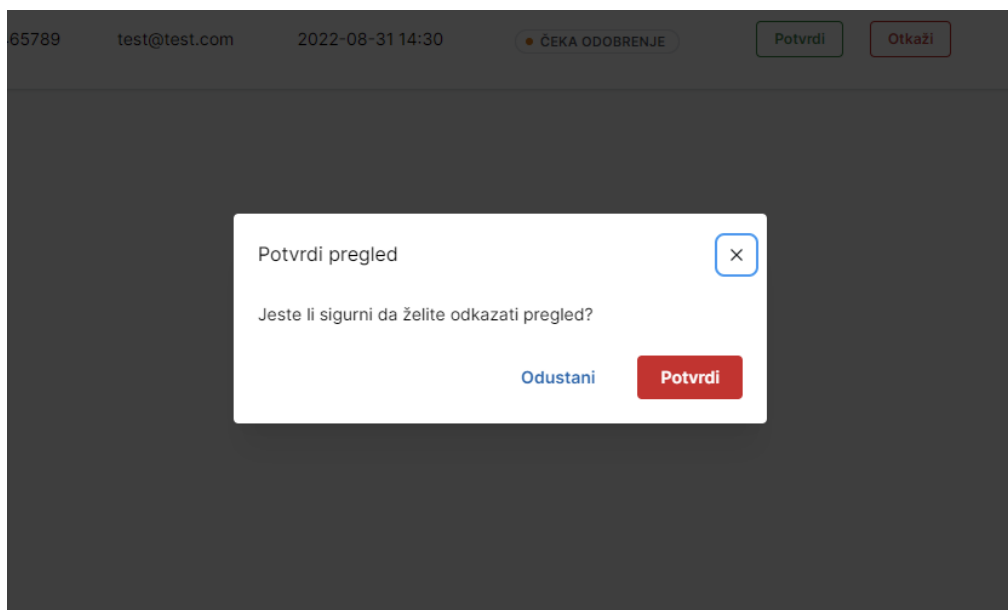


*Slika 4.35 zahtjevi za pregled*

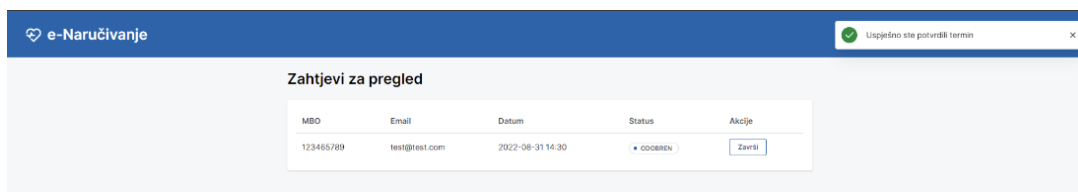
Klikom na gumb potvrdi (slika 4.36) ili otkazi (slika 4.37) otvara se skočni prozor radi potvrde željene akcije. Nadalje, nakon potvrde termina doktoru specijalistu se nudi dovršetak termina (slika 4.38) nakon što se termin završi kako bi znao da je obavio dani pregled.



*Slika 4.36 skočni prozor kod potvrde termina*

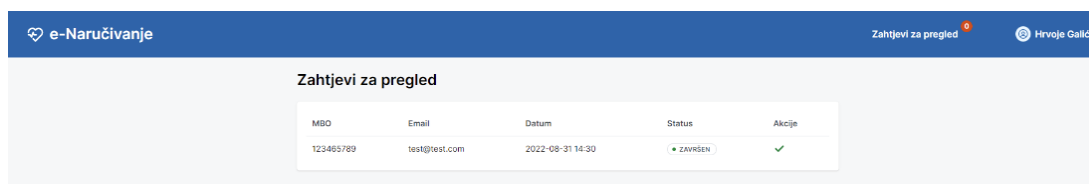


*Slika 4.37 skočni prozor kod otkazivanja termina*



*Slika 4.38 prikaz završetka termina*

Kako bi znali da je završeno promijenit ćemo status narudžbe te dodati kvačicu u stupac za akciju (slika 4.39).



*Slika 4.39 završen termin*

## 6. ZAKLJUČAK

U današnje vrijeme, kada nam je svaki trenutak bitan, web aplikacija ima danu funkciju da omogućuje tri vrste korisnika. Doktoru opće prakse se nudi funkcija da napravi digitalnim putem uputnicu za pacijenta odnosno pacijentu lakšu rezervaciju termina kod doktora specijalista, što bi rezultiralo bržom rezervacijom termina kod doktora specijalista, gdje pacijent ne bi trošio vrijeme na čekanje u redu, nego bi uz pomoć aplikacije i par klikova rezervirao termin kod željenog doktora. Nadalje aplikacija nudi obostranu komunikaciju između pacijenta i doktora. Doktoru specijalistu se nudi mogućnost odobravanja ili otkazivanja termina ako mu traženi termin od pacijenta ne odgovara. Pacijent može vidjeti potvrdu doktora te je tako siguran da li je uspješno rezervirao termin ili mu je termin otkazan. Kod doktora opće prakse aplikacija nam nudi mogućnost da sami doktor napravi uputnicu tako da ne mora ispunjavati sva polja unutar uputnice već preko MBO broja pronalazi pacijenta, te mu se automatski popunjavaju polja o pacijentu koja su unesena preko profila samog pacijenta. Što dovodi do toga na doktor opće prakse treba ispunit samo dva polja unutar uputnice za čega se uputnica upućuje, te koja je uputna dijagnoza pacijenta. Doktoru opće prakse ili poznatijem kao obiteljski doktor nudi mu se mogućnost pregleda svih uputnica koje je napravio za svoje pacijente. Ovaj način rada ubrzao bi proces rezerviranja termina kod doktora, te brže i točnije pravljenje uputnica. Nadalje aplikacija nudi veću i bolju preglednost bilo kod pacijenta ili doktora, te veću povezanost između njih dvoje. Naposljetku korisnici bi uštedjeli puno više vremena, te bi imali veću organiziranost u svome životu.

## LITERATURA

- [1] World Health Organization, COVID-19, dostupno na: <https://www.who.int/> [25.06.2022]
- [2] Booking.com, dostupno na: <https://www.booking.com/> [25.06.2022]
- [3] Crno jaje.com, dostupno na: <https://www.crnajaje.hr/> [25.06.2022]
- [4] Practo,Sminq,Doctors Appointments Apps for effective and timely consultation, dostupno na: <https://www.spec-india.com/blog/doctor-appointment-apps> [30.08.2022]
- [5] Stack OverFlow 2021, Developer Survey, dostupno na: <https://insights.stackoverflow.com/> [26.06.2022]
- [6] Postman, What is Postman?, dostupno na: <https://www.postman.com/> [26.06.2022]
- [7] Sebesta, R.W. Programming the World Wide Web (2nd Ed.) Addison-Wesley, Boston, MA, 2004.
- [8] SCSS, What is difference between CSS and SCSS, dostupno na: <https://www.geeksforgeeks.org/> [27.06.2022]
- [9] Bootstrap, Get started with bootstrap, dostupno na: <https://getbootstrap.com/> [27.06.2022]
- [10] Mantine, Fully featured React components library, dostupno na: <https://mantine.dev/> [27.06.2022]
- [11] React, Create-React-App, dostupno na: <https://reactjs.org/docs/create-a-new-react-app> [28.06.2022]
- [12] E. Brown, Web Development with Node and Express, July 2014
- [13] Express.js, Installing express.js, dostupno na: <https://expressjs.com/> [28.06.2022]
- [14] Firebase, Development platform firebase, dostupno na: <https://firebase.google.com/docs/> [28.06.2022]

## **POPIS UPOTREBLJENIH KRATICA**

**SARS-CoV-2** (engl. *Severe acute respiratory syndrome coronavirus 2*)

**COVID-19** (engl. *Coronavirus disease*)

**VS Code** (engl. *Visual Studio Code*)

**HTML** (engl. *HyperText Markup Language*)

**CSS** (engl. *Cascading Style Sheets*)

**PHP** (engl. *Hypertext Preprocessor*)

**API** (engl. *Application Programming Interface*)

**HTTP** (engl. *HyperText Transfer Protocol*)

**DOM** (engl. *Document Object Model*)

**AJAX** (engl. *Asynchronous JavaScript and XML*)

**XML** (engl. *Extensible markup language*)

**JS** (engl. *JavaScript*)

**CDN** (engl. *Content delivery network*)

**CPU** (engl. *Central processing unit*)

**JSON** (engl. *JavaScript Object Notation*)

**SDK** (engl. *Software development kit*)



## SAŽETAK

Rad web aplikacije izrađene u Reactu i Express-u se sastoji od da ima danu funkciju da pacijent prilikom odlaska na pregled kod doktora opće prakse dobiva uputnicu od malo prije navedenog doktora. Aplikacija nudi da ne moramo odlazi u bolnicu i čekati u redu da bi rezervirali termin kod željenog doktora , nego jednostavno od kuće putem aplikacije možemo rezervirati željeni termin, nadalje sve te termine pacijent može vidjeti, te mu nudi mogućnost otkazivanja termina ili promjenu vremena termina, ako mu ne odgovara prije navedeni termin. Pacijent po statusu rezervacije može vidjeti dali mu je traženi termin kod doktora specijalista, isti taj doktor odobrio traženi termin. Doktorima se isto tako nudi lakšu komunikaciju između pacijenta. Doktor opće prakse putem MBO broja pronalazi pacijenta te mu se automatski polja popunjavanju, te tako ne mora svaki put ispisivati cijelu uputnicu nego samo dva dijela za čega je uputnica, te opis dijagnoze pacijenta. Doktor specijalist vidi sve zahtjeve od pacijenta te tako može lakše organizirati si vrijeme i način posla. Doktoru specijalistu se nudi da dokaže traženi termin ili da potvrdi, nakon potvrde specijalist nakon pregleda potvrđuje dolazak pacijenta, te tako zna da je pregled obavljen.

Ključne riječi: doktor opće prakse, doktor specijalist, Express, pacijent, React, rezervacija kod doktora, uputnica

## **ABSTRACT**

The operation of the web application which is made in React and Express consists in the fact that when the patient goes to see a family doctor, he receives a referral from the previously mentioned doctor. The application offers that we do not have to go to the hospital and wait in line to book an appointment with the desired doctor, but simply from home through the application we can book the desired appointment, furthermore the patient can see all these appointments and offers him the possibility of canceling the appointment or changing the appointment time, if the aforementioned appointment does not suit him. According to the reservation status, the patient can see if requested appointment with a specialist doctor has been approved by the same doctor. Doctors are also offered easier patient to patient communication. The general practitioner finds the patient through the MBO number and fills in the fields automatically, so he does not have to write the entire referral every time, but only two parts, what the referral is for and a description of the patient's diagnosis. The specialist doctor sees all the requests from the patient and thus can more easily organize his time and way of working. The specialist doctor is offered to cancel the requested appointments or to confirm, after confirmation the specialist confirms the arrival of the patient after the examination, and thus knows that the examination has been completed.

Keywords: Family doctor, specialist doctor, Express, patient, React, doctors appointment, referral

## **ŽIVOTOPIS**

Hrvoje Zec, rođen 27.04.1999 u Slavonskom Brodu, pohađao osnovnu školu Bogoslav Šulek. Nakon završetka osnovne škole upisao opću Gimnaziju Matija Mesića u Slavonskom Brodu. Fakultet elektrotehnike, računarstva i informacijske tehnologije upisao 2018 godine u Osijeku.

---

Potpis autora

## **PRILOZI**

### **P.1. Izvorni kod aplikacije**

**Dostupno na:** [https://github.com/HrvojeZec/Hrvoje\\_Zavrzni\\_rad.git](https://github.com/HrvojeZec/Hrvoje_Zavrzni_rad.git)