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://urn.nsk.hr/urn:nbn:hr:200:612299

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-17**

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 Z1P - 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

Ime i prezime Pristupnika:	Hrvoje Zec
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4301, 26.07.2018.
OIB Pristupnika:	13825042338
Mentor:	Izv. prof. dr. sc. Josip Balen
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Web aplikacija za zdravstvene usluge
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	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
Datum prijedloga ocjene od strane mentora:	08.09.2022.
Datum potvrde ocjene od strane Odbora:	

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 20.09.2022.

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

1. UVOD **1**

1.1. Zadatak završnog rada **2**

2. APLIKACIJE PRIMJENJENE ZA REZERVIRANJE TERMINA U RAZLIČITE SVRHE **3**

3. ALATI I PROGRAMSKI JEZICI KORIŠTENI U IZRADI RADA **4**

3.1. Uređivač izvornog koda Visual Studio Code **4**

3.2. Programski alat Postman **5**

3.3. Programski jezik JavaScript **6**

3.4. Frontend dio aplikacije **6**

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

3.5. Backend dio aplikacije **11**

3.5.1. Node.js 11

3.5.2. Express.js 12

3.5.3. Postavljanje express.js na server 12

3.6. Baza podataka **14**

3.6.1. Autentifikacija 14

3.6.2. Baza podataka u stvarnom vremenu 14

3.6.3. Postavljanje baze podataka 15

4. IZRADA APLIKACIJE ZA REZERVACIJU TERMINA KOD DOKTORA SPECIJALISTA **18**

5. IZGLED APLIKACIJE **33**

6. ZAKLJUČAK **40**

LITERATURA **41**

POPIS UPOTREBLJENIH KRATICA **42**

SAŽETAK **43**

ABSTRACT **44**

ŽIVOTOPIS **45**

PRILOZI **46**

P.1. Izvorni kod aplikacije **46**

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 odlazit u bolnicu i čekati u redu kako bi rezervirali termin te gubiti vrijeme, nego bi jednostavni 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

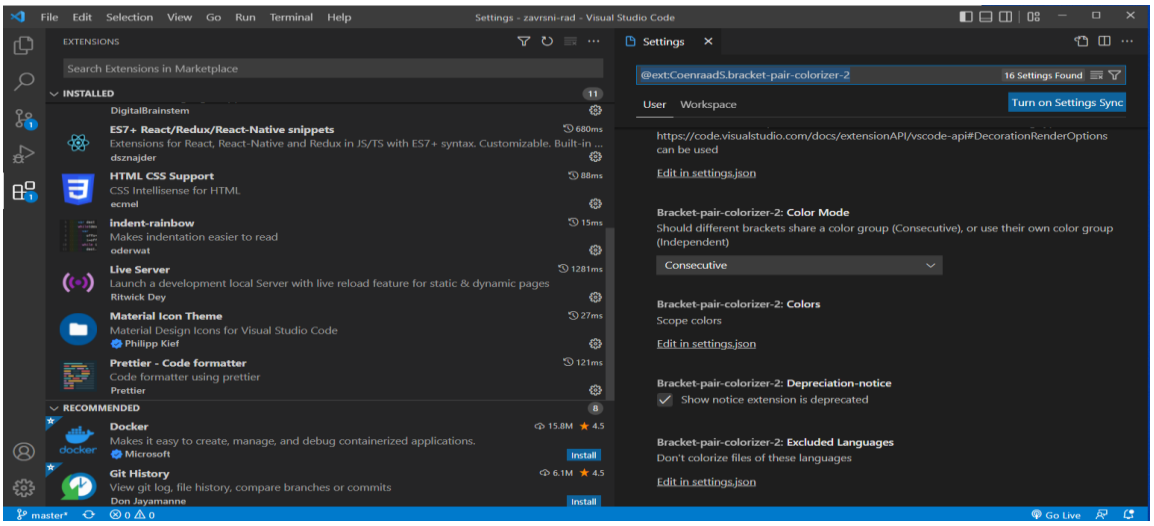
Razvojem interneta i tehnologije olakšale su se mnoge aktivnosti u svakodnevnom životu. Sve više i više koristimo i primjenjujemo tehnologiju kako bi pojednostavili način života. Razvile su se aplikacije primijenjene za rezerviranje termina u različite svrhe, što je ubrzalo proces razvoja dobara. Same 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

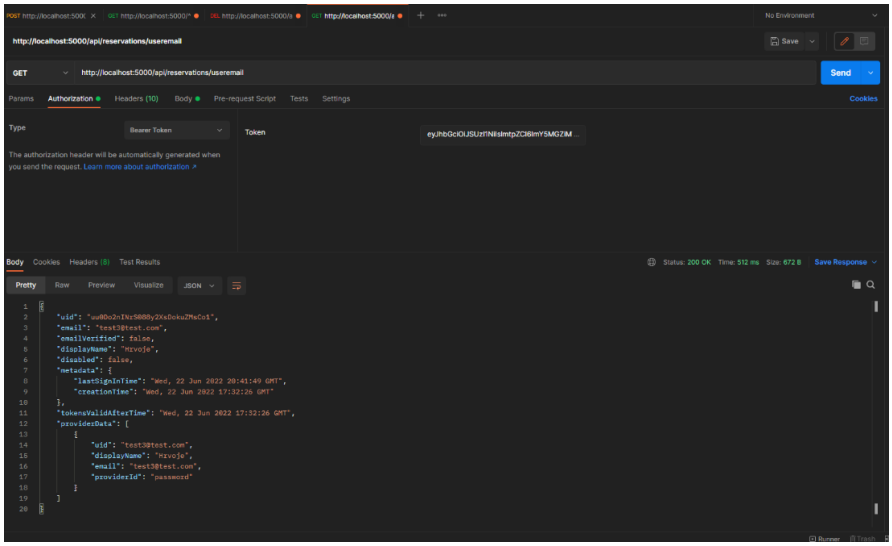
U anketi "Overflow 2021 Developer Survey" [5] Visual Studio Code je rangiran kao najpopularniji alat za razvojno okruženje, a 70% od 80.000 ispitanika izjavilo je da ga koriste. VS Code je lagan, ali prije svega moćan uređivač izvornog koda koji radi na vašoj radnoj površini (engl. *Desktop*) i dostupan je za Windows, macOS i Linux. Dolazi sa već ugrađenom podrškom za podržavanjem JavaScipta, 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 kaki 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 Postaman 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 DELET 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 dohvatit 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.

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 respozivnih 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.

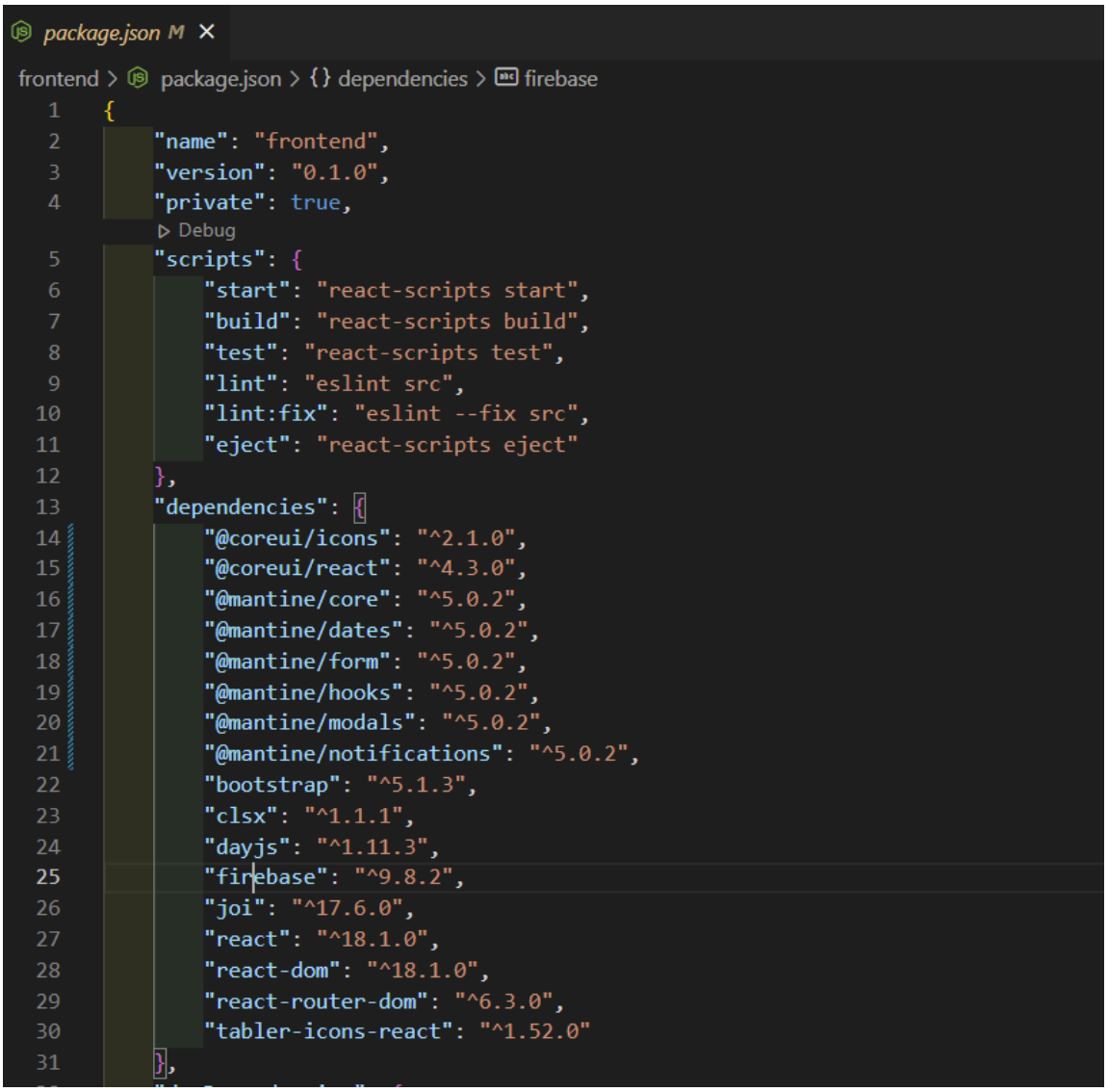
3.4.6. Postavljanje Mantinea

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

```
PS C:\Users\hrco\Desktop\zavrsni-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 pojavit u našem *package.json*, što možemo vidjeti na slici 3.5. Nakon uspješne instalacije možemo je koristit na način da dohvatimo biblioteku naredbom u liniji 1 prikazano slikom 3.6.



Slika 3.4 *prikaz package.json*

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

Sada možemo koristit komponente na način prikazan na slici 3.6.

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 JavaScrpit 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 pojma "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\zavrsni-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. *Enviroment*)

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 "un-opinionated 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\zavrsni-rad\backend> npm install express
```

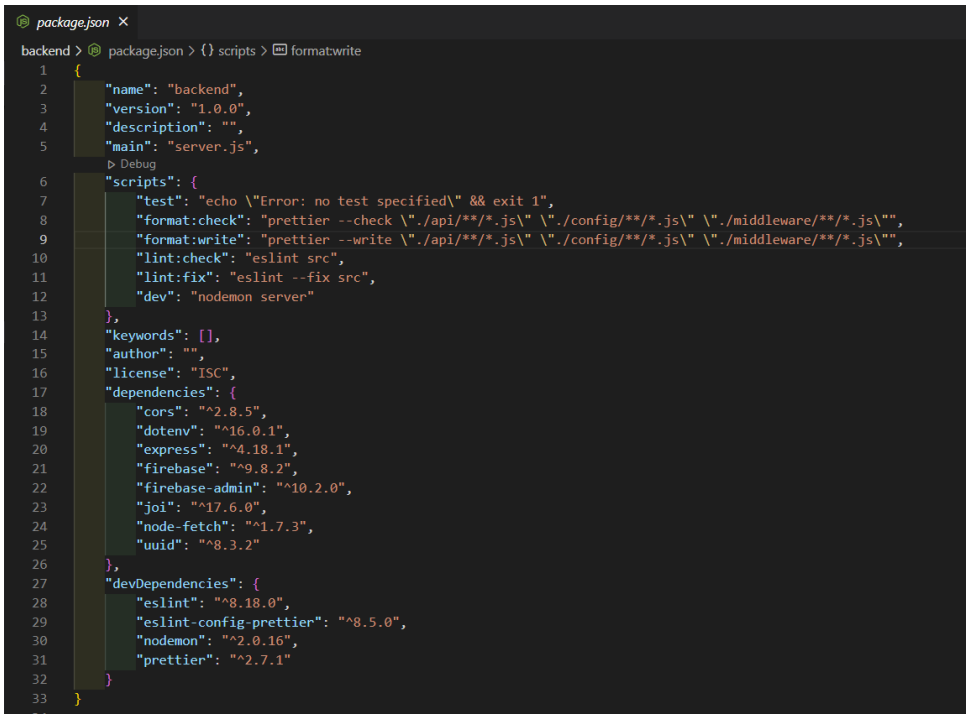
Slika 3.8 *instalacija express.js u backend datoteku*

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\zavrsni-rad\backend> npm install -g nodemon
```

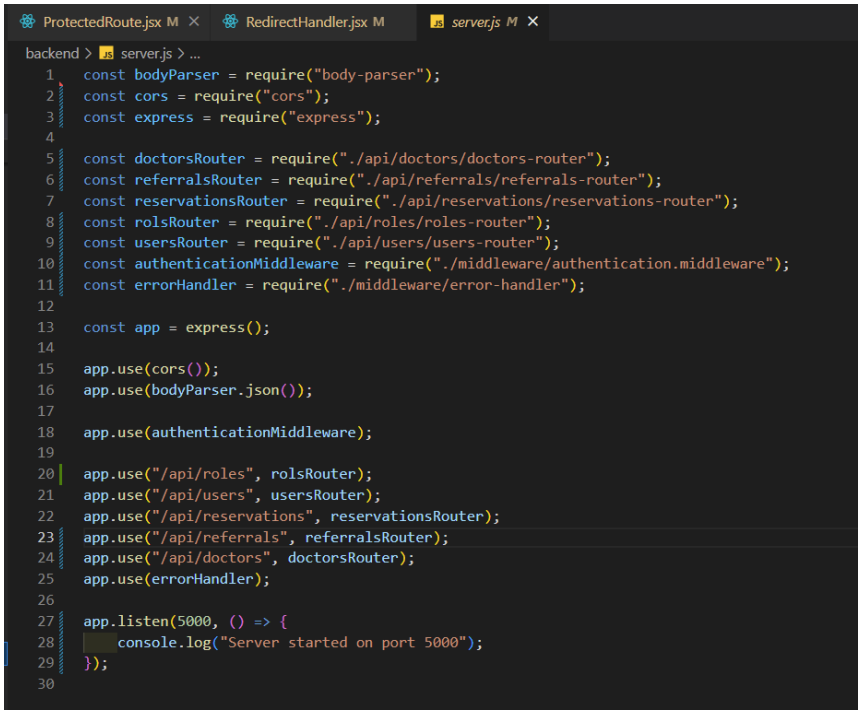
Slika 3.9. *instalacija nodemon alata*

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*.



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).



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 popravljanje 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 napravit 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\zavrsni-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)

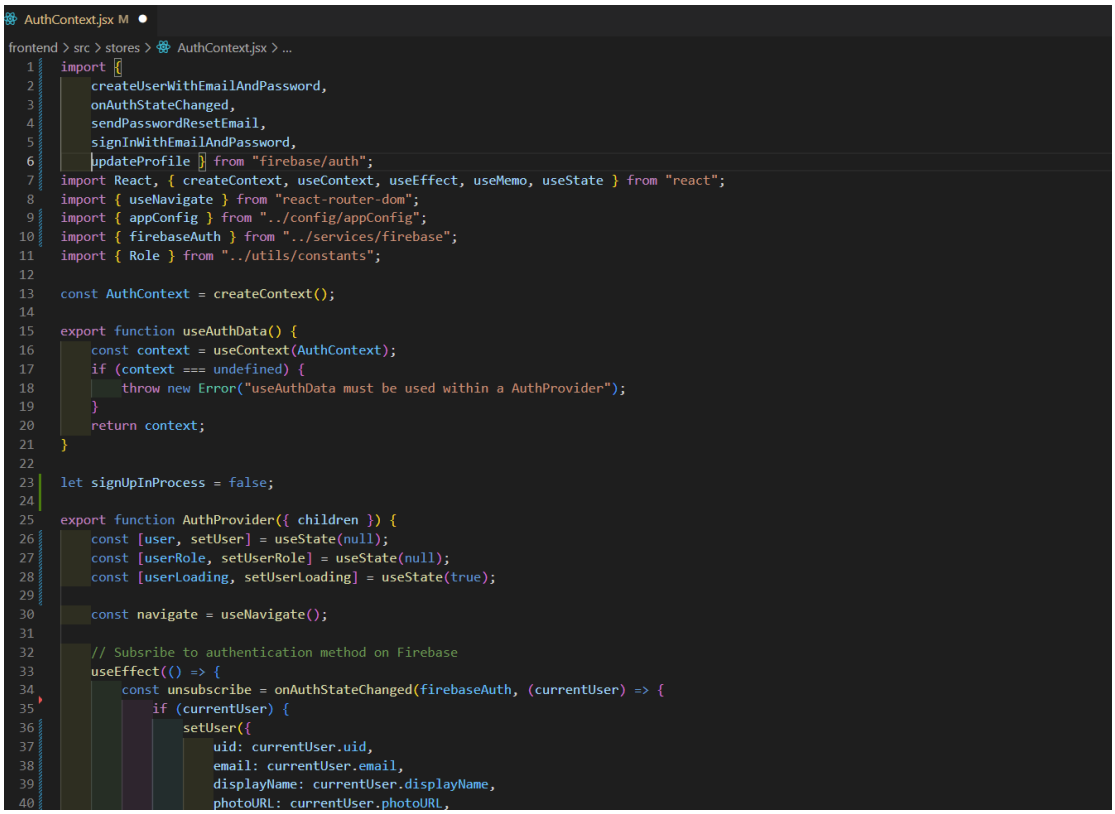
ConfirmAcceptModal.jsx U firebase.js

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_NZOdSjyMB8gLqZzvU",
7      appId: "1:456680698935:web:c35a68c11f739cbe117bb2",
8      projectId: "auth-zavrsni-rad-35575",
9      authDomain: "auth-zavrsni-rad-35575.firebaseapp.com",
10     storageBucket: "auth-zavrsni-rad-35575.appspot.com",
11     messagingSenderId: "456680698935",
12     databaseURL: "https://auth-zavrsni-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
```

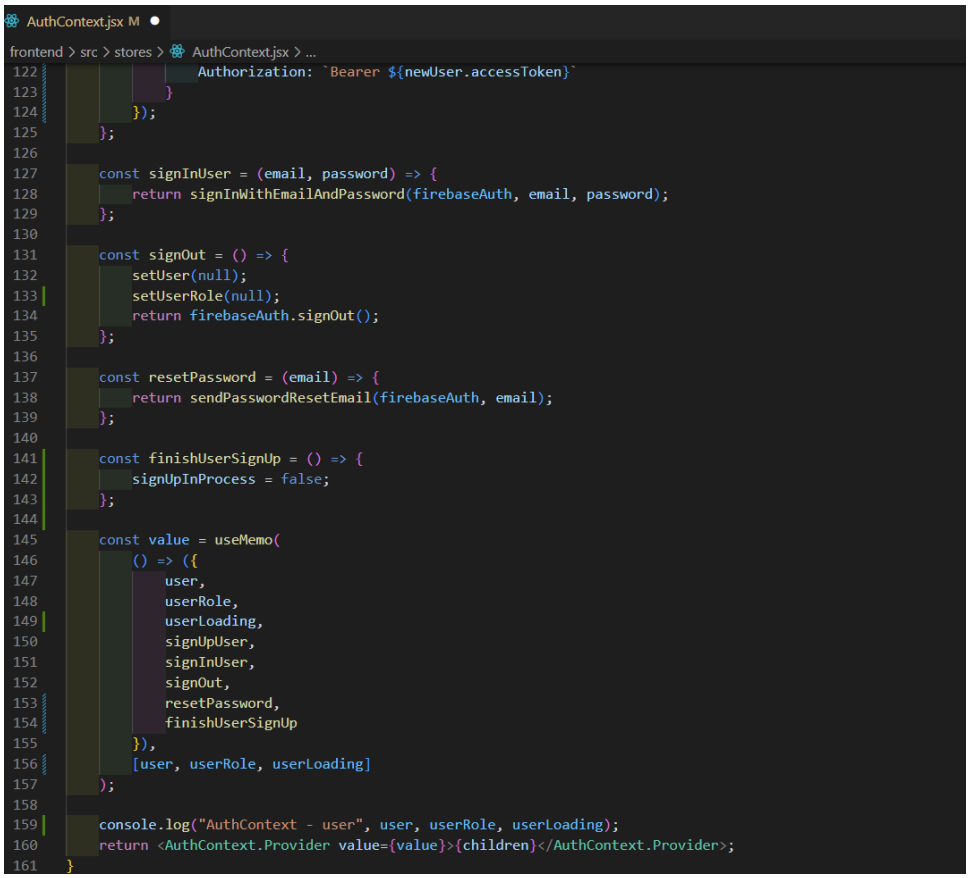
Slika 3.13 *Prikaz firebase.js datoteke*

Nakon što smo omogućili daljnje korištenje varijable *auth* i *app* postavljanjem "export". Kreirat ćemo još jednu datoteku pod nazivom AuthContext.js koja će nam služit 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.



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 napravit varijablu userContext (slika 3.14, linija 13) kojoj ćemo dodati funkciju *createContext*. Nadalje kreiranjem funkcije *AuthContextProvider* implementirat ćemo sve potrebne funkcije koje smo dohvatili iz *firebase/auth*. Također ćemo radi ljepšeg izgleda dodati dodatnu varijablu *value* u kojoj ćemo ubacit sve potrebne funkcije koje želimo koristit dalje u drugim datotekama. Ubacivanjem *value* varijable u *UserContext.Prvoider* (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 njegovi ovisnosti.



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

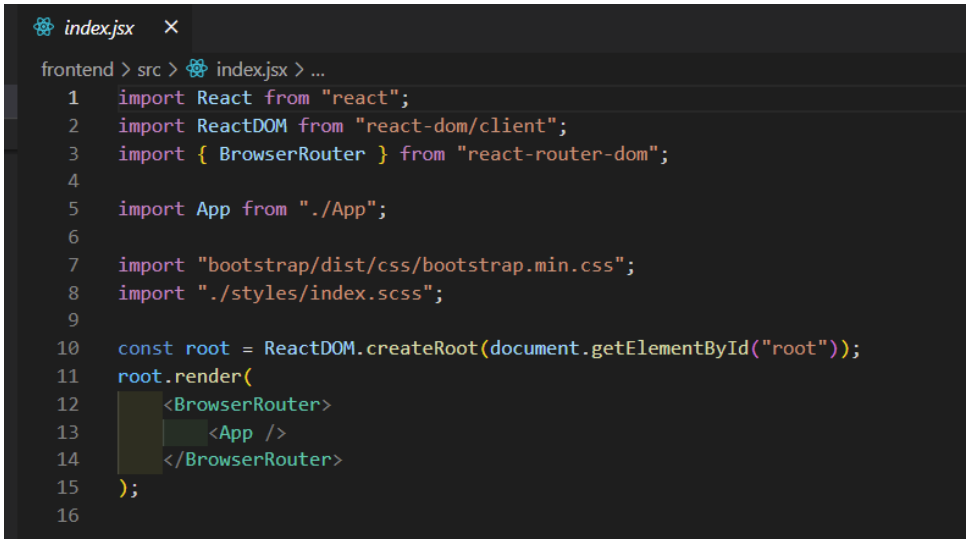
Nakon ovog postavljanja dohvaćanjem naredbe *UserAuth* možemo pristupit svim funkcijama u *AuthContext.js* datoteci. Kako bi zaštitili odnosno administrirali naš Firebase, kako neko drugi ne bi mogli dirati našu bazu podataka, u našem *backend* mapi kreirat ćemo *config* mapu u koju ćemo ubacit *firebase-config.js*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

Slika 3.16 *datoteka firebase-config.js*

4. IZRADA APLIKACIJE ZA REZERVACIJU TERMINA KOD DOKTORA SPECIJALISTA

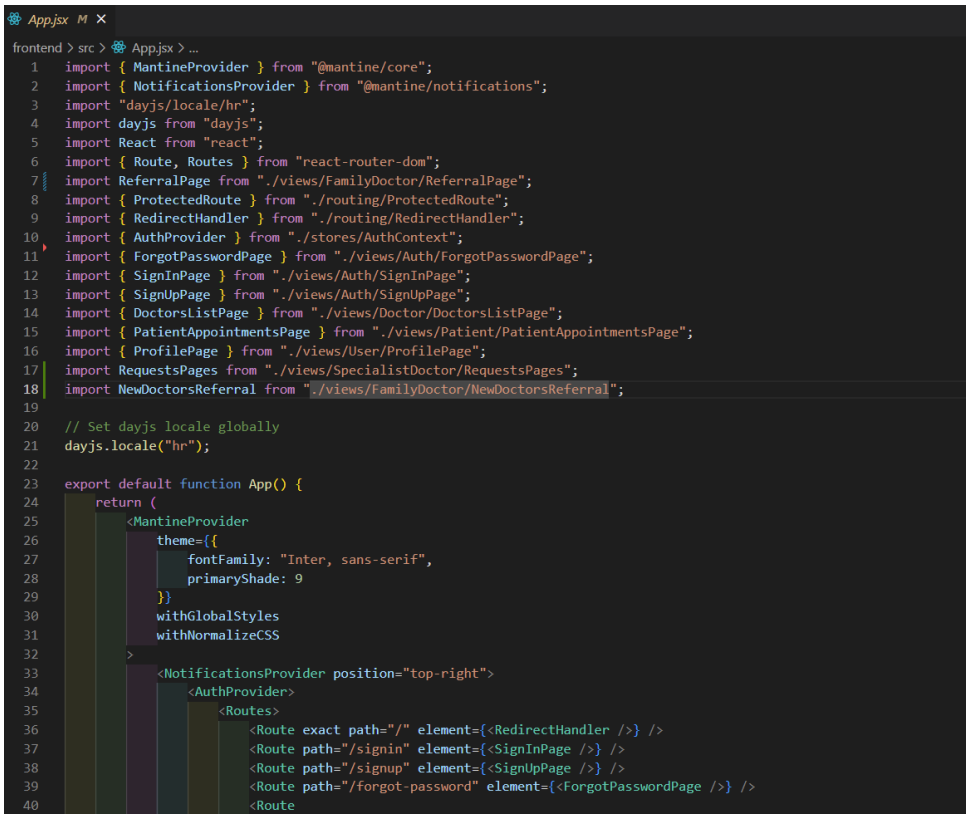
Kako bi olakšali strukturu koda, te bolju organizaciju razdijelit ćemo naš rad u dvije mape *backend* koji će sadržavati *Express.js* i *frontend* koji će sadržavati React dio aplikacije. Nakon postavljanja servera i svih ostalih dodatnih alata objašnjeni u prethodnim poglavljima spremni smo za izradu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

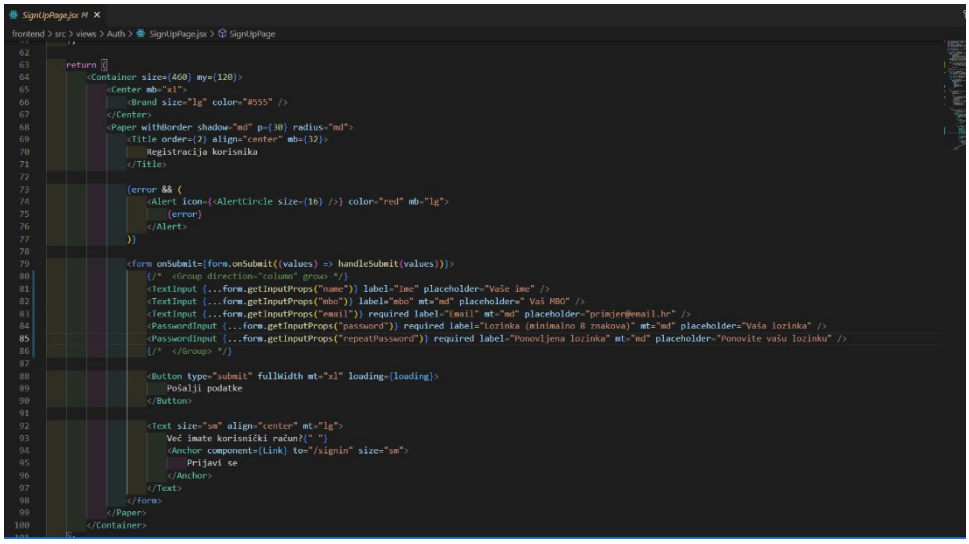


Slika 4.1 *index.jsx datoteka*

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 napravit je *SignUpPage.jsx* koja će nam predstavljati registraciju korisnika (slika 4.3).

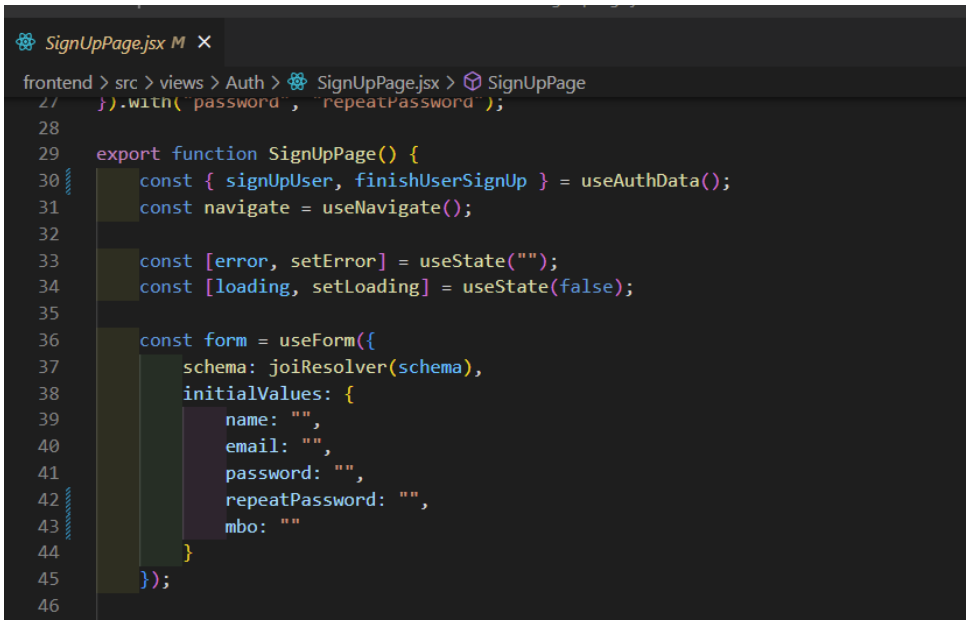


Slika 4.2 *app.jsx datoteka*



Slika 4.3 *prikaz koda za kreiranje izgleda sign up stranice*

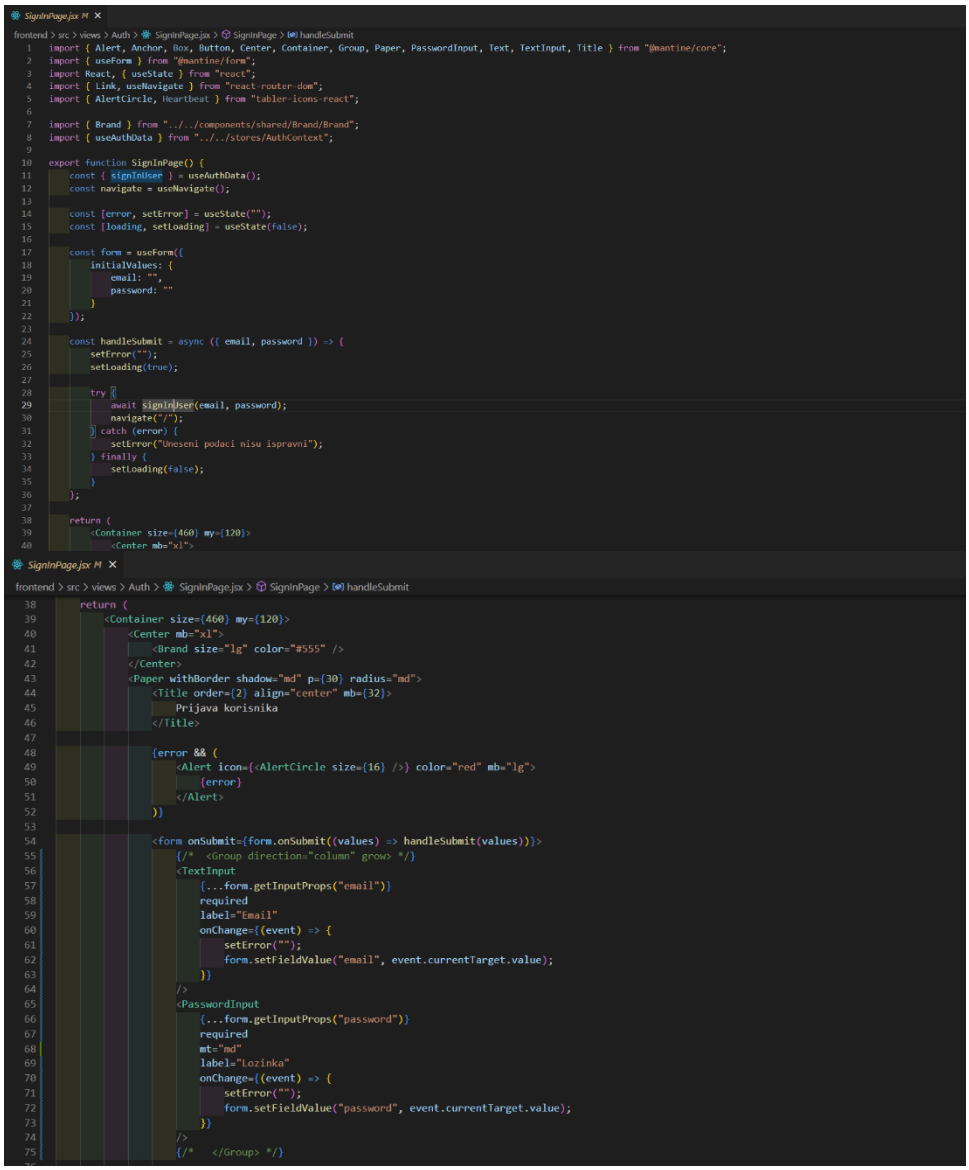
Logika oko rada aplikacije je da prvo uz pomoć *"hook-a useForm"* koji nam nudi Mantinea kreiramo dva stanja za dohvaćanje i postavljanje stanja odnosno naših komponenti za registraciju (slika 4.4). Nadalje uz pomoć Joi direktorija napravit ćemo validaciju unosa naših komponenti u registraciji (slika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



Slika 4.4. *logika oko rada signup stranice*

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 provjerit 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 S*ignInPage.jsx* (slika 4.6), koji umjesto *createUser* funkcije koristimo *signInUser*.

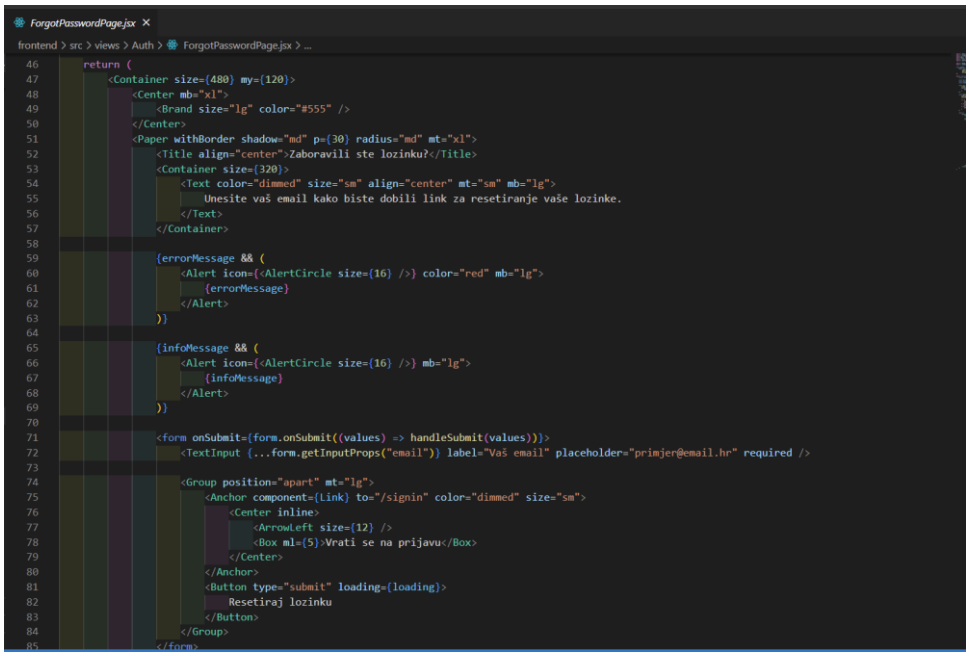


```jsx
import { Alert, Anchor, Box, Button, Center, Container, Group, Paper, PasswordInput, Text, TextInput, Title } from "@mantine/core";
import { useForm } from "@mantine/form";
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { AlertCircle, Heartbeat } from "tabler-icons-react";

import { Brand } from "../../components/shared/Brand/Brand";
import { useAuthData } from "../../stores/AuthContext";

export function SignInPage() {
    const { signInUser } = useAuthData();
    const navigate = useNavigate();

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

    const form = useForm({
        initialValues: {
            email: "",
            password: ""
        }
    });

    const handleSubmit = async ({ email, password }) => {
        setError("");
        setLoading(true);

        try {
            await signInUser(email, password);
            navigate("/");
        } catch (error) {
            setError("Uneseni podaci nisu ispravni");
        } finally {
            setLoading(false);
        }
    };

    return (
        <Container size={460} my={120}>
            <Center mb="xl">
```

```jsx
    return (
        <Container size={460} my={120}>
            <Center mb="xl">
                <Brand size="lg" color="#555" />
            </Center>
            <Paper withBorder shadow="md" p={30} radius="md">
                <Title order={2} align="center" mb={32}>
                    Prijava korisnika
                </Title>

                {error && (
                    <Alert icon={<AlertCircle size={16} />} color="red" mb="lg">
                        {error}
                    </Alert>
                )}

                <form onSubmit={form.onSubmit((values) => handleSubmit(values))}>
                    {/*  <Group direction="column" grow> */}
                    <TextInput
                        {...form.getInputProps("email")}
                        required
                        label="Email"
                        onChange={(event) => {
                            setError("");
                            form.setFieldValue("email", event.currentTarget.value);
                        }}
                    />
                    <PasswordInput
                        {...form.getInputProps("password")}
                        required
                        mt="md"
                        label="Lozinka"
                        onChange={(event) => {
                            setError("");
                            form.setFieldValue("password", event.currentTarget.value);
                        }}
                    />
                    {/*   </Group> */}
```

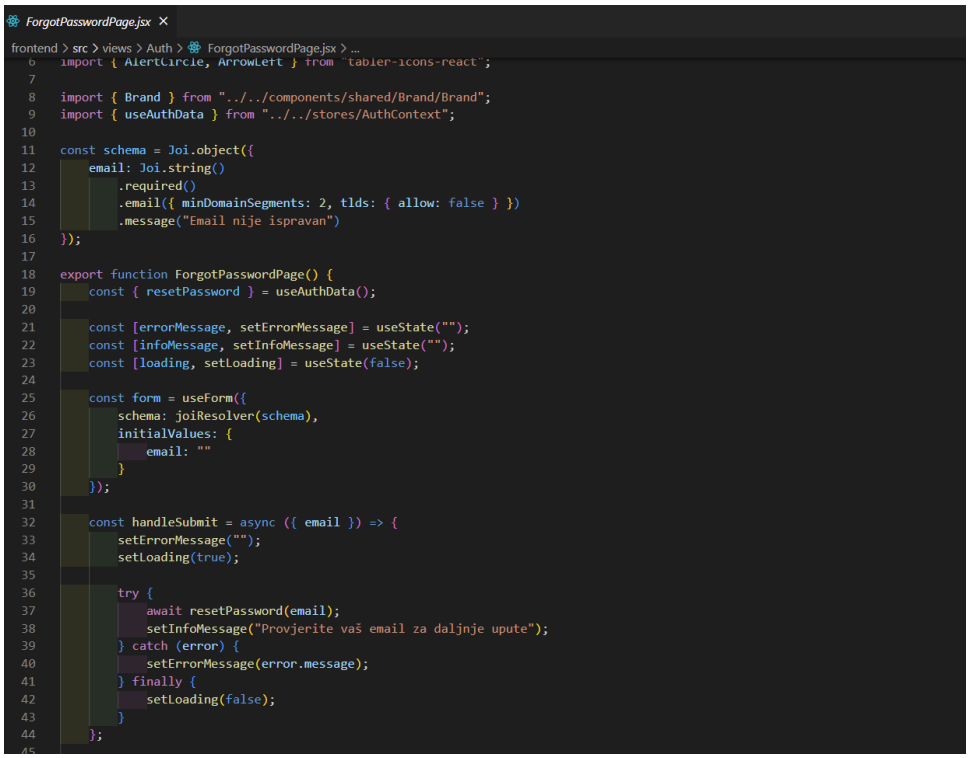
***Slika 4.6**. prikaz signin.jsx datoteke*

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.



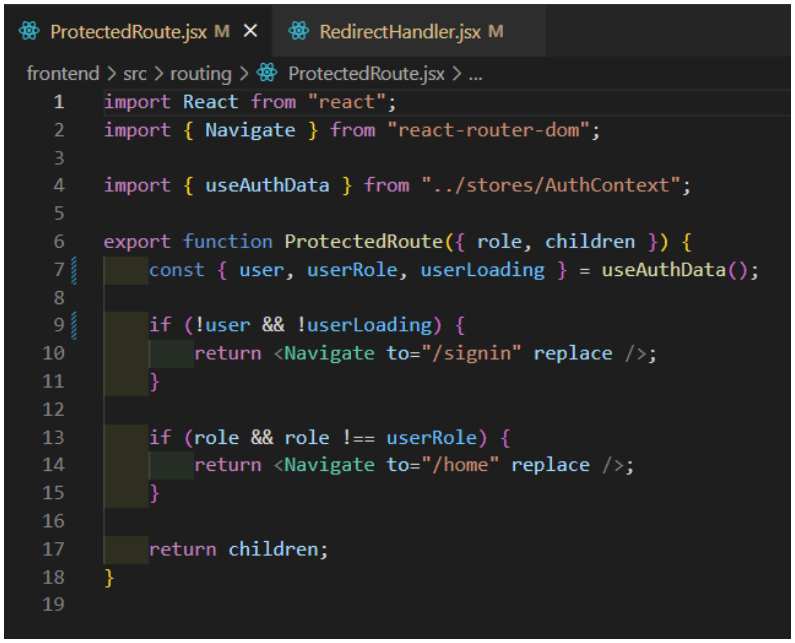
Slika 4.7. *ForgotPassword datoteka*

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.

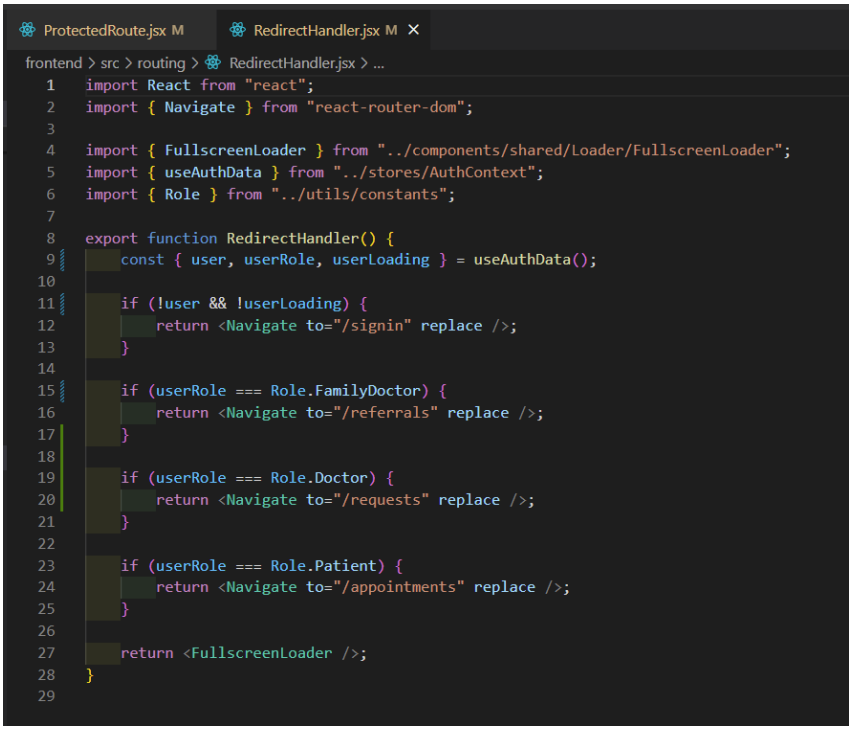


Slika 4.8 *logika reset Password stranice*

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* dohvatit i koristit tako što ćemo obuhvatit rute koje želimo zaštititi da im korisnik ne može pristupit ako prije toga nije obavio prijavu.

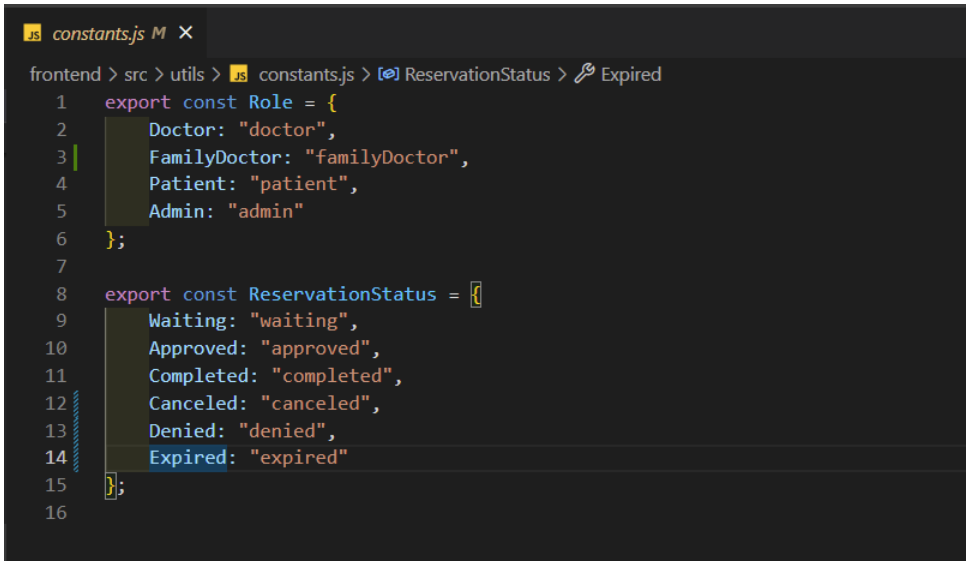


Slika 4.9 *ProtectedRoute.jsx datoteka*



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 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.



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 otkaže termin pomoću funkcije *cancleReservation* (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);
            setResevationToCancel(undefined);
        });
};
```

Slika 4.12 *funkcija cancleReservation*

```
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          setResevationToUpdate(undefined);
81      }
82  };
```

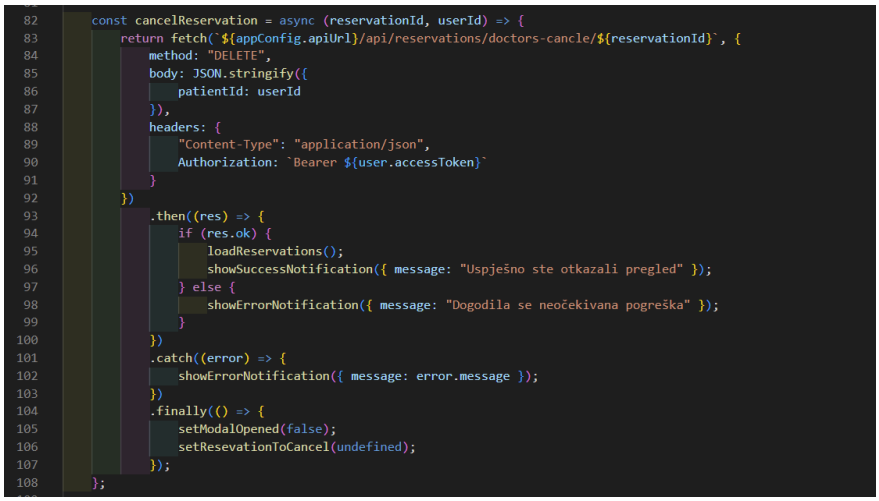
Slika 4.13 *funkcija handleBookingUpdate*

Kao što možemo vidjeti na slikama pozivom funkcije šaljemo 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).

***4.14**. promjena termina na backendu*

***Slika 4.15** otkazivanje termina na backendu*

Nakon rezervacije termina pacijent čeka potvrdu od doktora specijalista dali termin odgovara ili ne. U *SpecilaistDoctor* 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 otkaže termin ili potvrdi termin uz pomoć funkcija *cancleReservatio*n (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.



```
const cancelReservation = async (reservationId, userId) => {
    return fetch(`${appConfig.apiUrl}/api/reservations/doctors-cancle/${reservationId}`, {
        method: "DELETE",
        body: JSON.stringify({
            patientId: userId
        }),
        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);
            setResevationToCancel(undefined);
        });
};
```

Slika 4.16. *cancaleReservation u requests datoteci*

```
const completeReservation = async (reservationId, userId) => {
    return fetch(`${appConfig.apiUrl}/api/reservations/doctor-complete/${reservationId}`, {
        method: "PATCH",
        body: JSON.stringify({
            patientId: userId
        }),
        headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${user.accessToken}`
        }
    })
        .then((res) => {
            if (res.ok) {
                loadReservations();
                showSuccessNotification({ message: "Uspješno ste potvrdili termin" });
            } else {
                showErrorNotification({ message: "Dogodila se neočekivana pogreška" });
            }
        })
        .catch((error) => {
            showErrorNotification({ message: error.message });
        })
        .finally(() => {
            setCompleteModalOpened(false);
            setResevationToComplete(undefined);
        });
};
```

Slika 4.17 *completeResrvation u requests datoteci*

```
const confirmReservation = async (reservationId, userId) => {
    return fetch(`${appConfig.apiUrl}/api/reservations/doctor-confirm/${reservationId}`, {
        method: "PATCH",
        body: JSON.stringify({
            patientId: userId
        }),
        headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${user.accessToken}`
        }
    })
        .then((res) => {
            if (res.ok) {
                loadReservations();
                showSuccessNotification({ message: "Uspješno ste potvrdili termin" });
            } else {
                showErrorNotification({ message: "Dogodila se neočekivana pogreška" });
            }
        })
        .catch((error) => {
            showErrorNotification({ message: error.message });
        })
        .finally(() => {
            setUpdateModalOpened(false);
            setResevationToUpdate(undefined);
        });
};
```

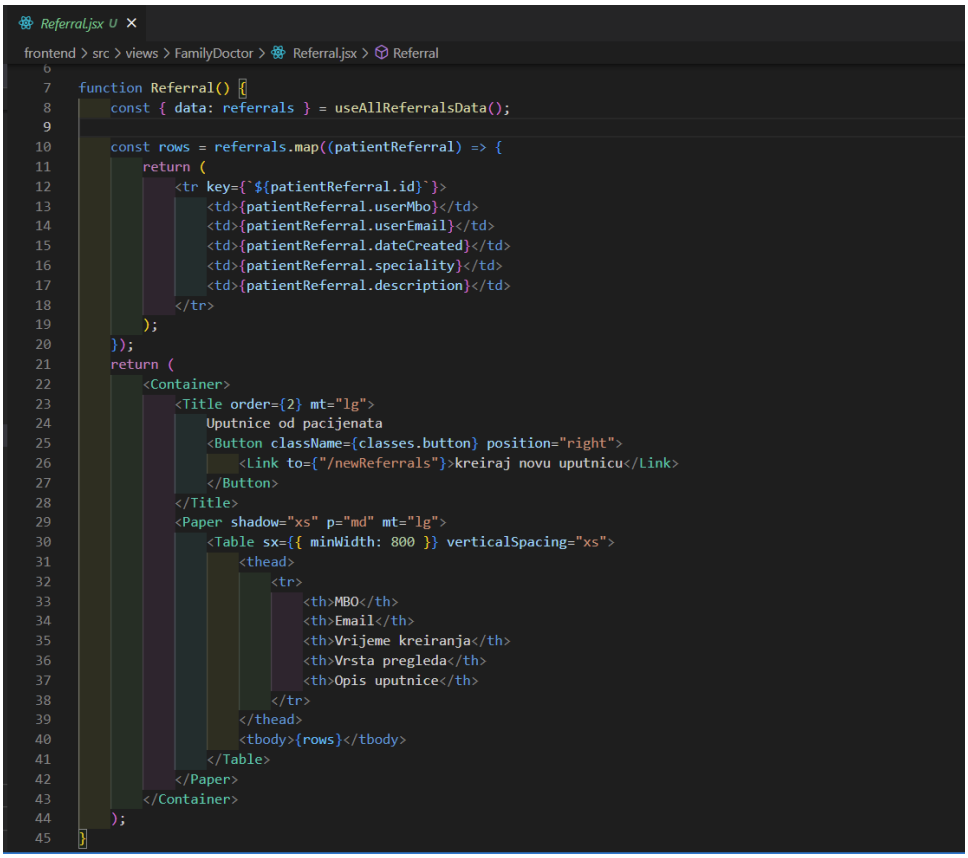
Slika 4.18 *confirmReservation u requests datoteci*

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

```
router.delete("/doctors-cancle/:reservationId", authorization(Role.Doctor), async (req, res, next) => {
    try {
        await doctorDeniedReservation(req.params.reservationId, req.body.patientId);
        return res.status(200).send();
    } catch (err) {
        next(err);
    }
});
router.patch("/doctor-confirm/:reservationId", authorization(Role.Doctor), async (req, res, next) => {
    try {
        await updateReservation(req.params.reservationId, req.body.patientId);
        return res.status(200).send();
    } catch (err) {
        next(err);
    }
});

router.patch("/doctor-complete/:reservationId", authorization(Role.Doctor), async (req, res, next) => {
    try {
        await completeReservation(req.params.reservationId, req.body.patientId);
        return res.status(200).send();
    } catch (err) {
        next(err);
    }
});

  const confirmReservation = async (reservationId, userId) => {
      const existingReservationsForUser = await getExistingReservationsForUser(userId);

      const reservationToUpdate = existingReservationsForUser.find((it) => it.id === reservationId);
      if (reservationToUpdate) {
          reservationToUpdate.status = ReservationStatus.Approved;
      }

      await db.ref("reservations").child(userId).set(existingReservationsForUser);
  };

  const completeReservation = async (reservationId, userId) => {
      const existingReservationsForUser = await getExistingReservationsForUser(userId);

      const reservationToComplete = existingReservationsForUser.find((it) => it.id === reservationId);

      if (reservationToComplete) {
          reservationToComplete.status = ReservationStatus.Completed;
      }

      await db.ref("reservations").child(userId).set(existingReservationsForUser);
  };

const doctorDeniedReservation = async (reservationId, patientId) => {
    const existingReservationsForUser = await getExistingReservationsForUser(patientId);

    const reservationToDeny = existingReservationsForUser.find((it) => it.id === reservationId);
    if (reservationToDeny) {
        reservationToDeny.status = ReservationStatus.Denied;
    }

    await db.ref("reservations").child(patientId).set(existingReservationsForUser);
};
```

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.



Slika 4.20 *Prikaz Referrals stranice*

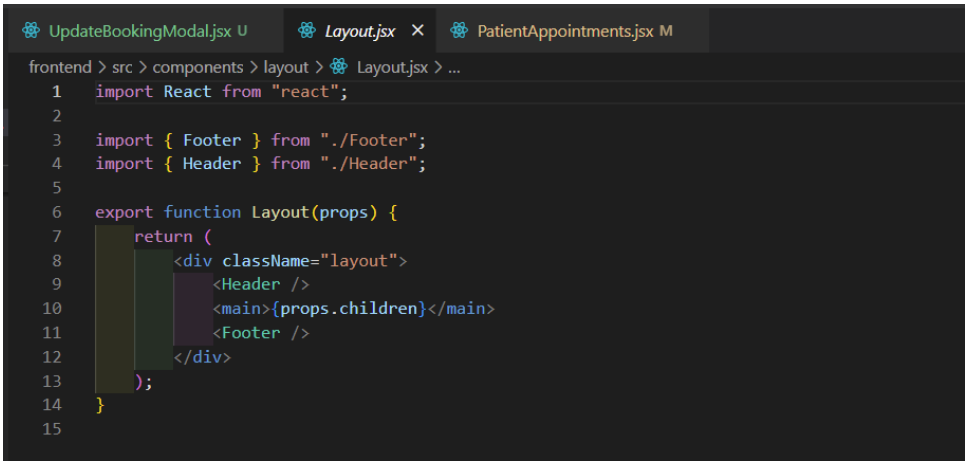
Kreiranje nove uputnice radi na principu da doktor opće prakse preko MBO broja pronalazi pacijenta iz baze podataka (slika 4.21) , te shodno tome automatski se popunjavanju podaci o tom pacijentu izvučeni iz Firebasea. Doktoru ostaje da dovršiti uputnicu tako da će birati za čega se uputnica upućuje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

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.



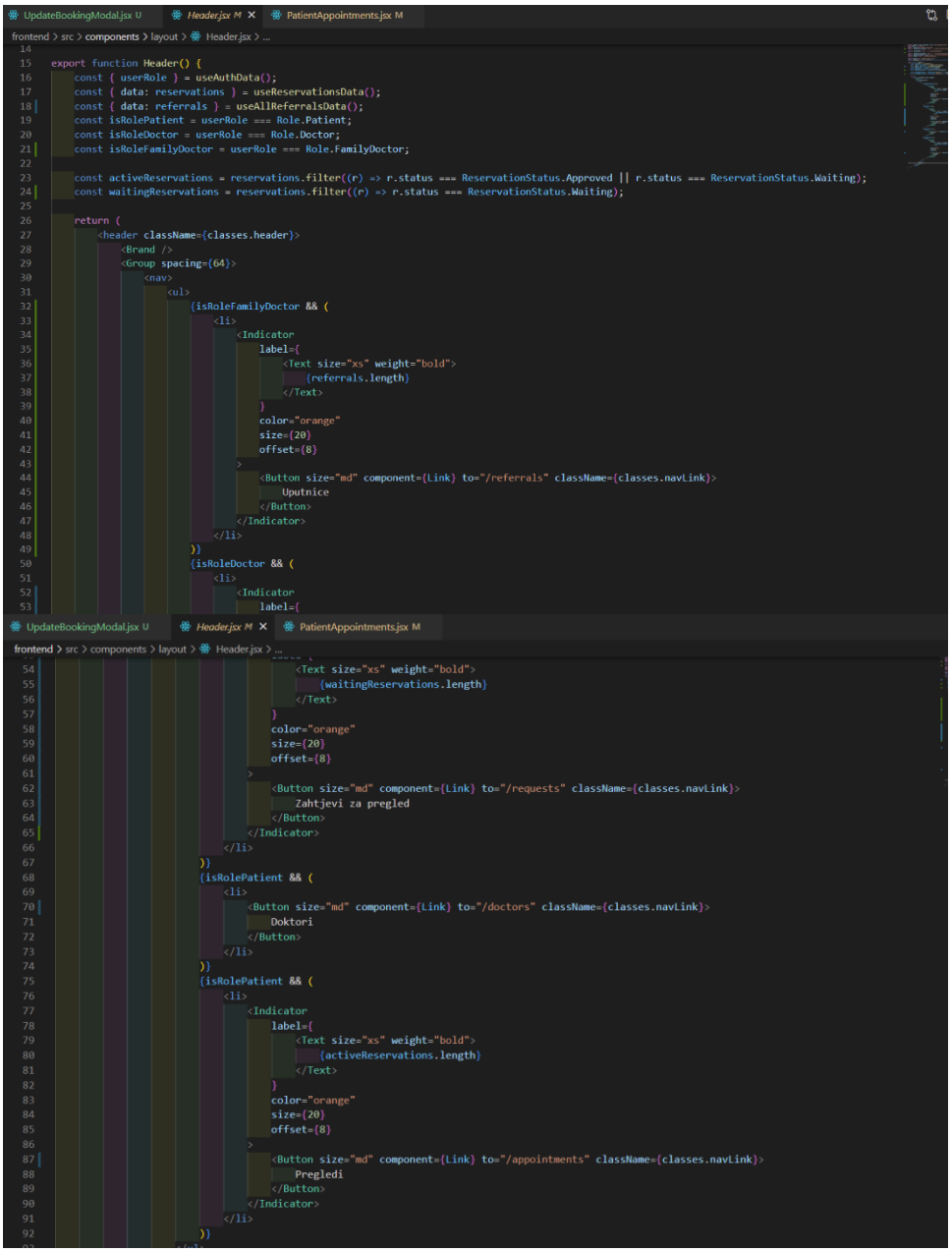
UpdateBookingModal.jsx U | Layout.jsx | PatientAppointments.jsx M

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)



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).

```jsx
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 > ...

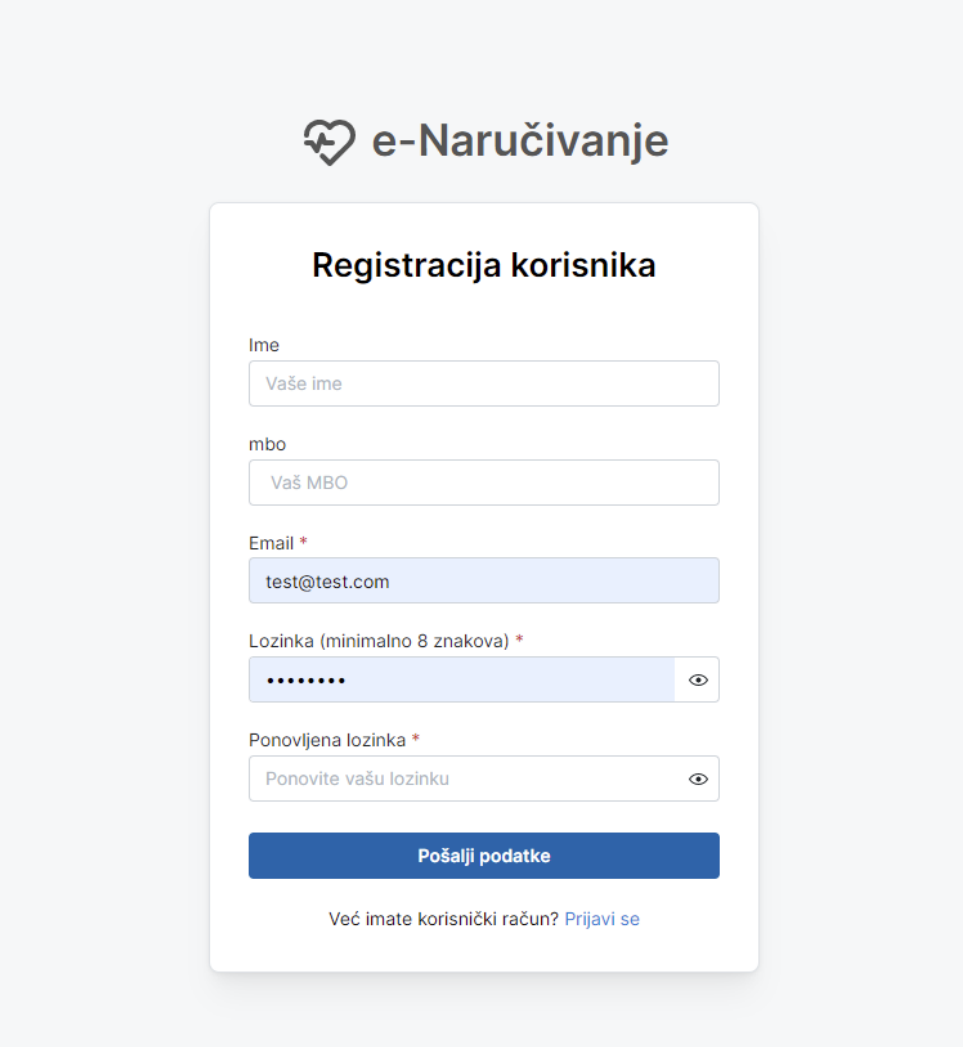
```jsx
import { usePersonData } from "../../stores/PersonContext";
import { isOibValid } from "../../utils/validations";

const MIN_AGE_YEARS = 18;
const maxDatePickerDate = dayjs().startOf("day").subtract(MIN_AGE_YEARS, "years").toDate();

const schema = Joi.object({
    name: Joi.string().min(3).message("Ime mora sadržavati minimalno 3 znaka"),
    oib: Joi.string().custom((value, helper) => {
        if (value.length !== 11) {
            return helper.message("OIB mora imati 11 znakova");
        }

        return true;
    }),
    dateOfBirth: Joi.date().max(maxDatePickerDate).message("Datum rođenja nije ispravan"),
    address: Joi.string().min(5).message("Adresa mora sadržavati minimalno 5 znakova"),
    phoneNumber: Joi.string()
        .min(10)
        .message("Broj telefona mora imati minimalno 10 znakova")
        .pattern(/^[+ 0-9]+$/)
        .message("Broj telefona nije ispravnog formata")
}).unknown();
```

Slika 4.25 *validacija unosa u Profile.jsx datoteci*

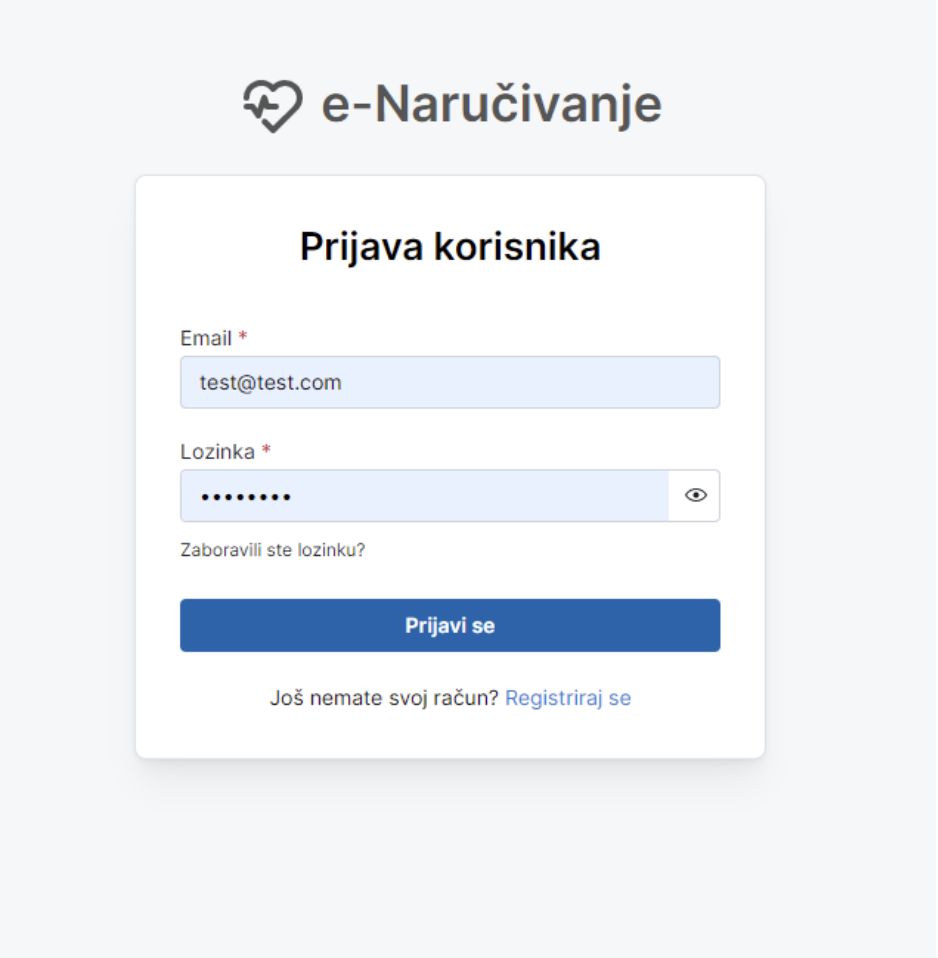
5. IZGLED APLIKACIJE

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.

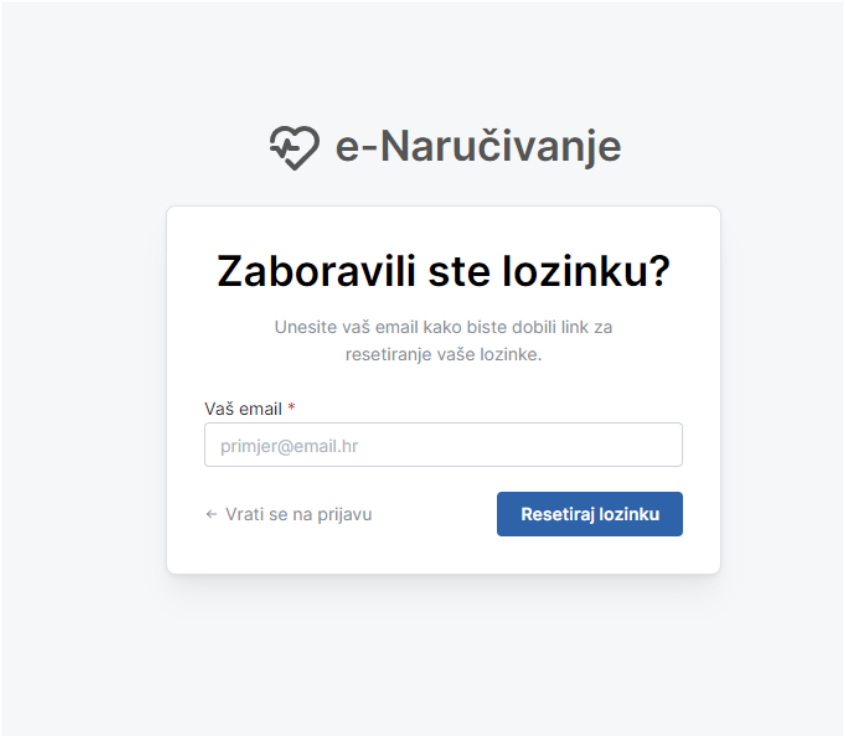
Pri samom otvaranju aplikacije pod nazivom e-naručivanje prikazano nam je prvi dio aplikacije koji predstavlja registraciju (slika 4.26), prijavu (slika 4.27), te resetiranje lozinke (slika 4.28).



Slika 4.26 *prikaz registracije korisnika*

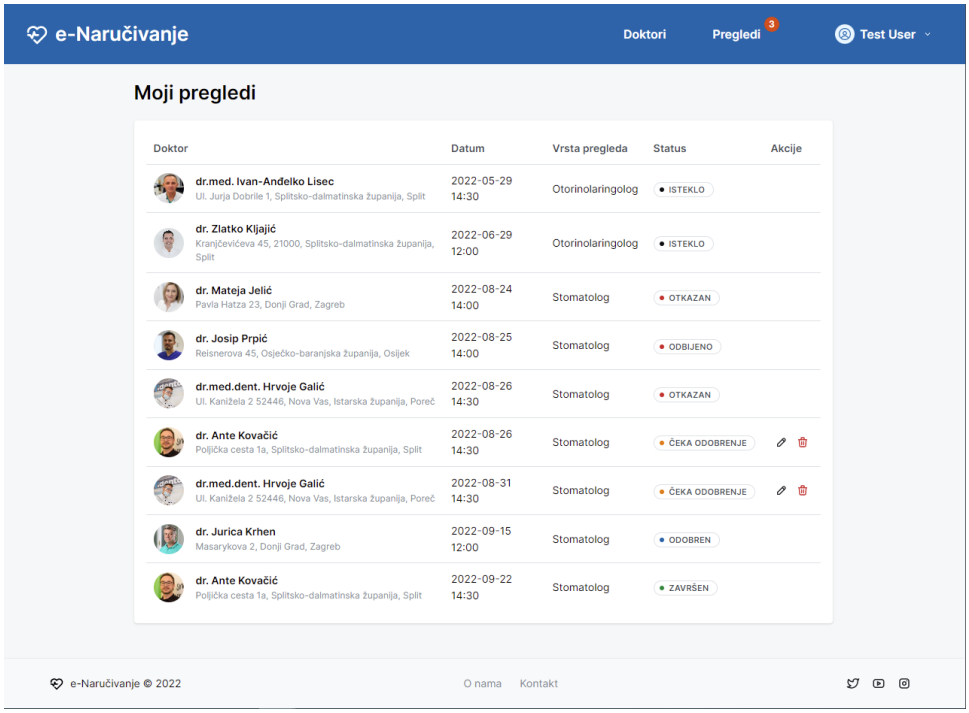


Slika 4.27 *prikaz prijave korisnika*



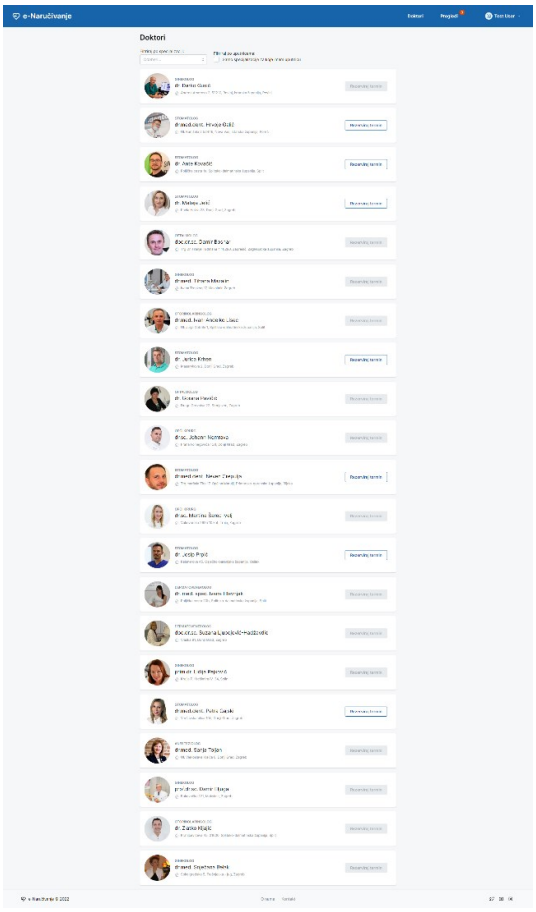
Slika 4.28 *prikaz resetiranje lozinke korisnika*

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.

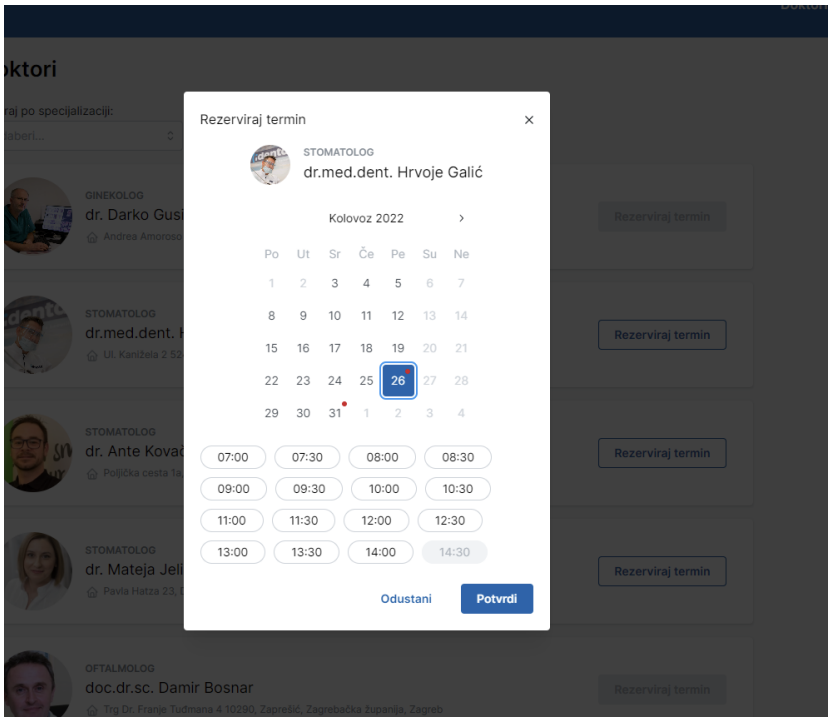


Slika 4.29 *prikaz rezerviranih termina*

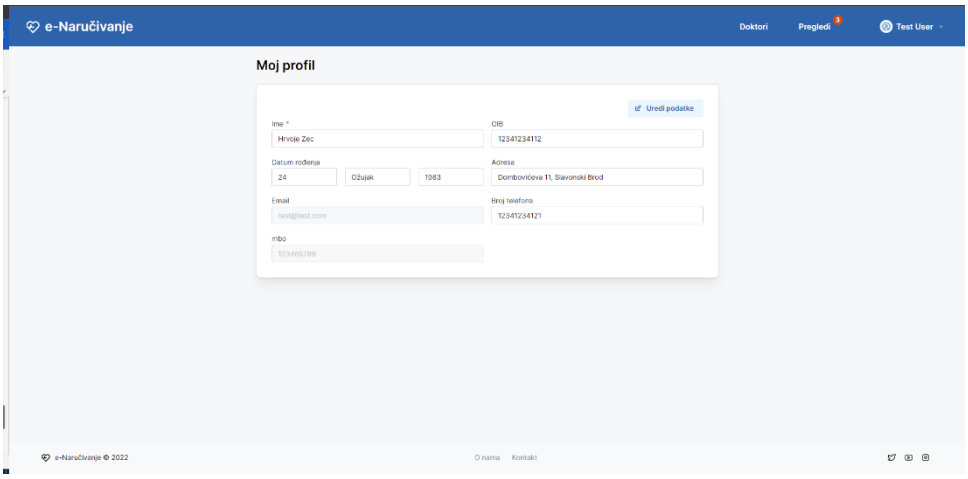
Korisniku se nude mogućnosti da rezervira termin na *doctors* stranici (slika 3.30), te će uz pomoć skočnog prozora (slika 4.31) izabrati željeni termin, nadalje korisniku se nudi opcija *Profile* (slika 4.32) gdje pacijent popunjava polja za unos, kao što su njegov OIB, broj mobitela, datum rođenja, te adresu. Polja kao što su MBO, ime korisnika i email koje je korisnik unijeo tijekom registracije ne mogu se mijenjati, te prema njima raspoznajemo 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 new*Referral* (slika 4.33) stranice koju može otvorit samo korisnik sa rolom *familyDoktor*. Doktoru opće prakse nudi se još i *referral* stranice gdje može vidjeti sve uputnice od pacijenata (slika 4.34)



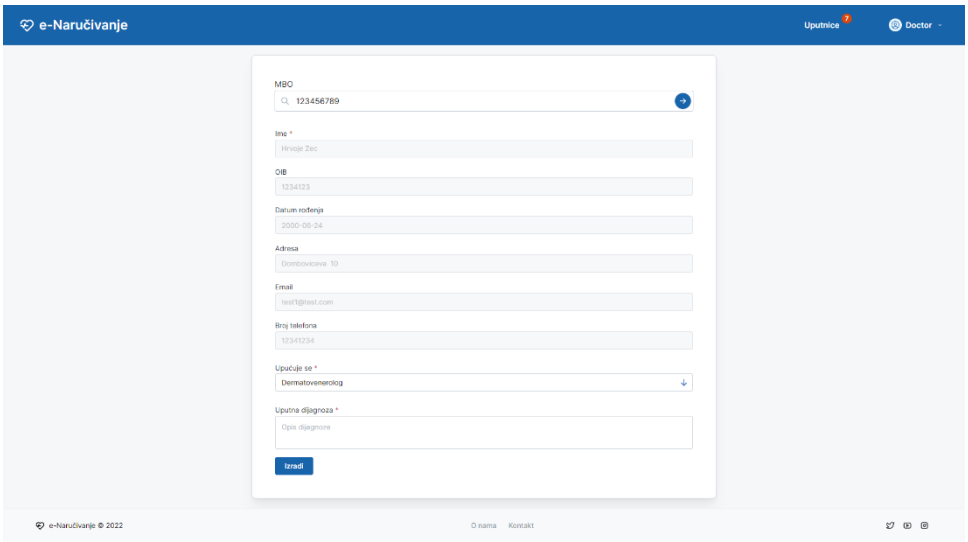
Slika 3.30 *prikaz doctors stranice*



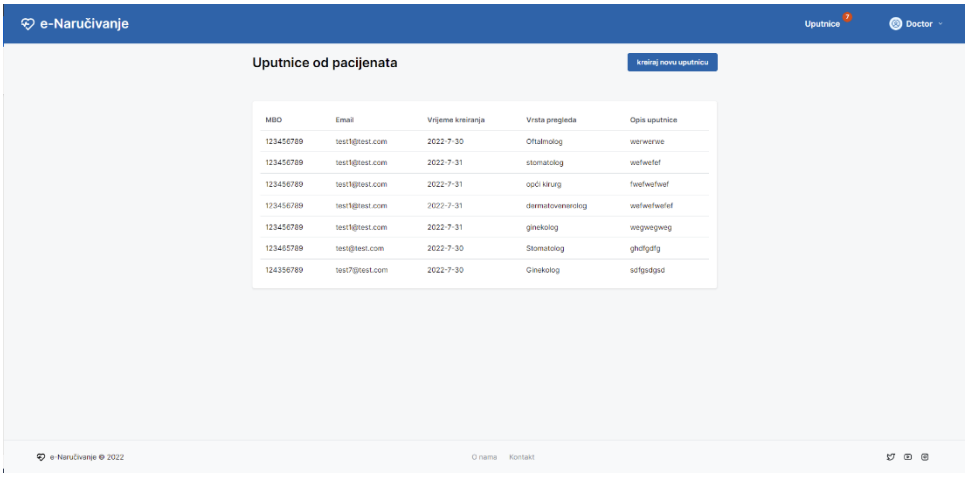
Slika 3.31 *skočni prozor za popunjavanje termina*



Slika 4.32 *prikaz profila korisnika*

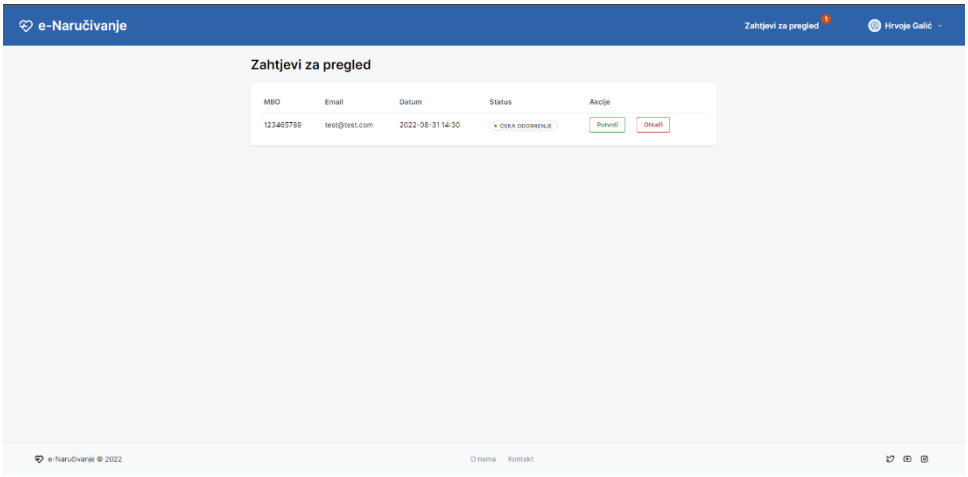


Slika 4.33 *prikaz ispunjavanja uputnice*



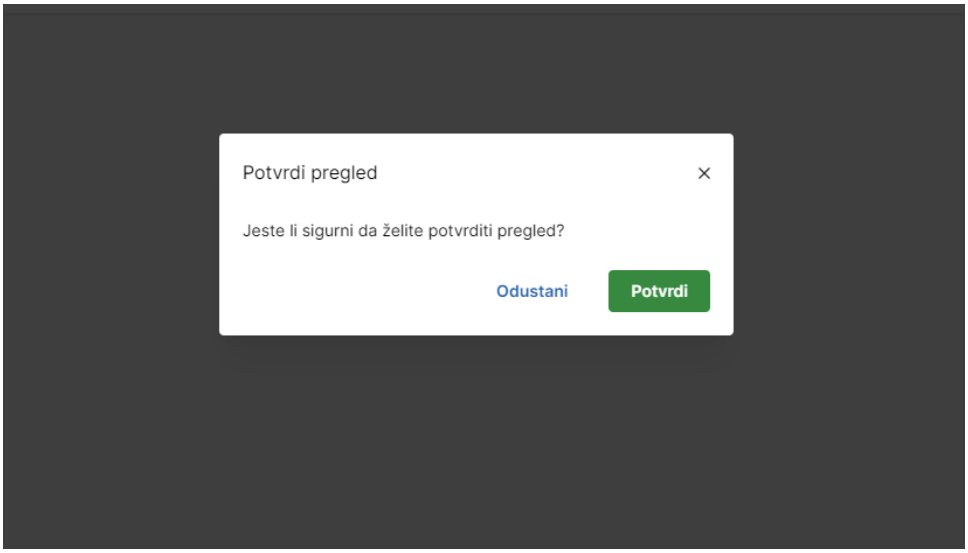
Slika 4.34 *prikaz svih uputnica*

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).

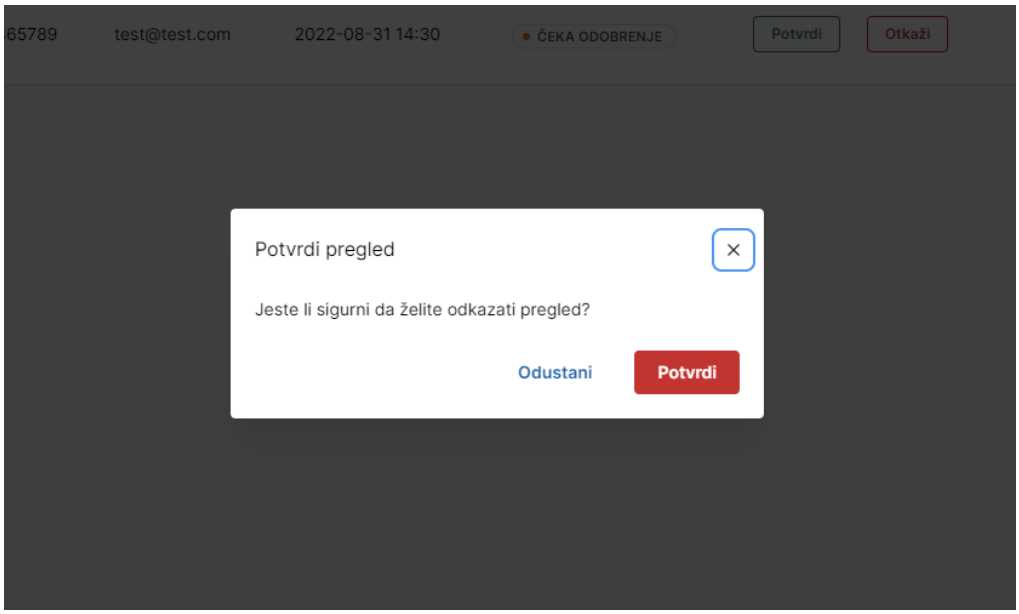


Slika 4.35 *zahtjevi za pregled*

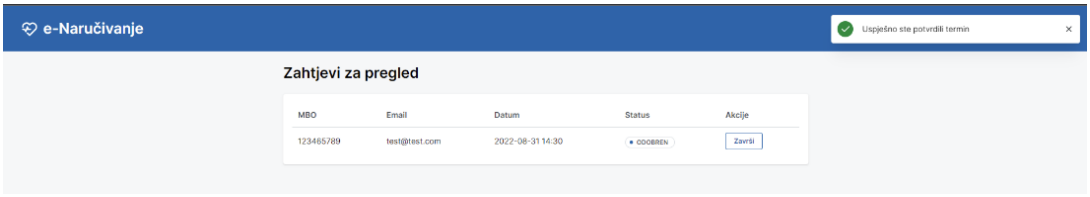
Klikom na gumb potvrdi (slika 4.36) ili otkaži (slika 4.37) otvara se skočni prozor radi potvrde željene akcije. Nadalje, nakon potvrde termina doktoru specijalistu se nudi dovršetak termina (slika 4.38) nakon što se termin završi kako bi znao da je obavio dani pregled.



Slika 4.36 *skočni prozor kod potvrde termina*

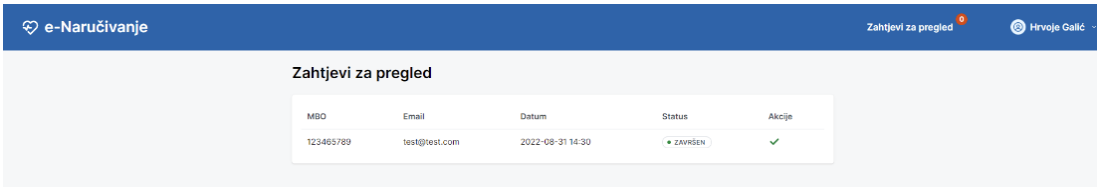


Slika 4.37 *skočni prozor kod otkazivanja termina*



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 praske 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.crnojaje.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] **E**xpress.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 vidjet, 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 mention 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 appointemnt, 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, se 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 patients diagnosis.The specialist doctor sees all the requests from the patient and thus can more easly organize his time and way of worknig. The specialist doctor is offered to cancle the requestes appointemnts or to confirm, after confrimation 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_Zavrsni_rad.git