Docconnect: web aplikacija za umrežavanje liječnika

Ivanda, Tomislav

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Tomislav Ivanda

DocConnect: Web aplikacija za umrežavanje liječnika

Diplomski rad

Pula, rujan 2022. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Tomislav Ivanda

DocConnect: Web aplikacija za umrežavanje liječnika

Diplomski rad

JMBAG: Tomislav Ivanda, 0303069498

Studijski smjer: Informatika

Predmet: Napredni algoritmi i strukture podataka

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: Izv.prof.dr.sc. Tihomir Orehovački

Pula, rujan 2022. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Tomislav Ivanda, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da nikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Ivanda

_____

U Puli, 20. 09. 2022 godine

IZJAVA

o korištenju autorskog djela

Ja, Tomislav Ivanda dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom DocConnect: Web aplikaciju za umrežavanje liječnika koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 20.9.2022

Potpis

Ivanda

_____

Sadržaj

1. UVOD

U ovom diplomskom radu kao tema bit će predstavljena izrada web aplikacije pod nazivom DocConnect. Aplikacija je zamišljena i izrađena kao društvena mreža s naglaskom na umrežavanje liječnika. U doba u kojem internetske tehnologije polako uzimaju sve veći obol u svim sferama društva ostalo je mjesta za još napredovanja u nekim područjima. Medicina kao znanost jako je bitna zona interesa svih ljudi, a liječnici kao takvi imaju veliku ulogu u današnjem svijetu. Svaki radnik kao i liječnik pogriješe u svome poslu, što je sasvim u redu, ali težina koju liječnik nosi svojom krivom odlukom puno je veća nego kod nekog običnog radnika. Iz članka objavljenog na portalu DW Made for minds stoji konstatacija koja kaže da je „svaka sedma liječnička dijagnoza pogrešna“ [6]. Iz prizme običnog čovjeka ta se informacija čini ne baš pretjerano bitna. Zato je svrha ove aplikacije smanjiti broj krivih dijagnoza umrežavanjem liječnika iz cijele Hrvatske, a cilj aplikacije je razmjena znanja, upoznavanje ljudi iz iste interesne sfere i rješavanje težih slučajeva davanjem drugog mišljenja na temelju dijeljenih informacija. Aplikacija sadrži većinu stvari koje sadrži stranica npr. Facebook , ali neke nevažne funkcije su namjerno izostavljene kao npr. igrice, praćenje raznih novinskih portala. To je namjerno izostavljeno zato što ova aplikacija želi prije svega bili ozbiljno mjesto na internetu sa svrhom pomaganja drugima. Dosta doktora već prakticira razmjenu mišljenja o drugim slučajevima , ali putem drugih digitalnih kanala. Uobičajeno koriste druge kanale kao što su Viber ili Whatsapp za razmjenu mišljenja što ne pridonosi tome da veći broj liječnika vidi sami problem koji taj liječnik iznese. Zato ova aplikacija ima za cilj povezati što veći broj liječnika stvarajući grupu stručnih i kompetentnih osoba umanjujući tako postotak pogreške. Samim time što je to mreža za sebe koja ima jednu svrhu bazira ljude da svoju pažnju posvete samo tome što se na njoj nalazi dok na mrežama koje se danas koriste ima dosta faktora ometanja koju mogu umanjiti njihov odgovor. Ovakav tip aplikacije koji se bavi umrežavanjem liječnika ne postoji na internetu stoga je za pretpostaviti da je to jako dobar početak i motiv za daljnji rad i razvoj same aplikacije, jer mogućnosti za unapređenje je mnogo zato što sve prelazi u digitalni oblik.

U prvom dijelu rada opisane su tehnologije koje su se koristile pri izradi same aplikacije, kakva je njihova uloga u aplikaciji te je prikazan primjer koda gdje se one koriste. Nakon toga prezentiran je prikaz pojedinačnih stranica ove aplikacije i kako bi one trebale izgledati na kraju samog projekta kada se sve napravi. U drugom dijelu opisane su klijent i server strana aplikacije i sve datoteke koje se nalaze u njima prezentiraju se i opisuje se njihov rad u aplikaciji. Zadnji dio ovog projekta prikazuje dodavanje samog Socketa.IO za komunikaciju između samih korisnika govorom. Na poveznici se može pogledat cijeli kod aplikacije https://github.com/tivanda/DocConnectFinall.

2. KORIŠTENA RAZVOJNA OKRUŽENJA ZA IZRADU APLIKACIJE

Prilikom izrade same aplikacije prvo je potrebno proučiti područje koje odgovara samim zahtjevima aplikacije i koje su sve tehnologije potrebne. S obzirom da se u ovom slučaju radi o web aplikaciji naglasak je bio na tom kriteriju.

2.1 React

React (također poznat kao React.js ili ReactJS) besplatna je frontend JavaScript biblioteka otvorenog koda za izgradnju korisničkih sučelja temeljenih na UI komponentama. Održavaju ga Meta (bivši Facebook) i zajednica pojedinačnih programera i tvrtki (React, n.d.). React se može koristiti kao baza u razvoju jednostranih, mobilnih ili poslužiteljskih aplikacija s okvirima poput Next.js. Međutim, React se bavi samo upravljanjem stanjem i renderiranjem tog stanja u DOM-u, tako da stvaranje React aplikacija obično zahtijeva korištenje dodatnih biblioteka za usmjeravanje, kao i određene funkcionalnosti na strani klijenta. JSX ili JavaScript Syntax Extension proširenje je sintakse JavaScript jezika. Izgledom sličan HTML-u, JSX pruža način strukturiranja renderiranja komponenti pomoću sintakse poznate mnogim programerima. React komponente obično su napisane pomoću JSX-a, iako ne moraju biti (komponente mogu biti napisane i u čistom JavaScriptu). JSX je sličan drugoj sintaksi proširenja koju je stvorio Facebook za PHP pod nazivom XHP [9]. Na slici 1 prikazan je kostur jedne React komponente korištene u ovom radu .

```
import React from "react";

import "./Nav.css";

const Nav = () => {
  return (
    <div className="Nav"> Nav</div>
  );
};

export default Nav;
```

Slika 1 React komponente

2.2 Express

Express.js, ili jednostavno Express, pozadinski je okvir web aplikacije za Node.js, objavljen kao besplatni softver otvorenog koda pod MIT licencom. Dizajniran je za izradu web aplikacija i API-ja. Nazvan je de facto standardni okvir poslužitelja za Node.js. Izvorni autor, TJ Holowaychuk, opisao ga je kao poslužitelja nadahnutim Sinatrom , što znači da je relativno minimalan s mnogo značajki koji su dostupni kao dodaci [4]. Express je pozadinska komponenta popularnih razvojnih stekova kao što su MEAN, MERN ili MEVN stack, zajedno sa softverom baze podataka MongoDB i JavaScript front-end okvirom ili bibliotekom. Express je stvoren za izradu API-ja i web aplikacija. Štedi mnogo vremena kodiranja gotovo upola, a još uvijek čini web i mobilne aplikacije učinkovitima. Još jedan razlog za korištenje Expressa je taj što je napisan u JavaScriptu jer je JavaScript jednostavan jezik čak i ako nemate prethodnu znanje bilo kojeg jezika. Express omogućuje mnogim novim programerima da uđu u područje web razvoja. Benefiti korištenja Express.js-a su : vremenski učinkovit, brz, ekonomičan i lako se uči . Na slici 2 ispod vidimo implementiranje samog Expressa za spremanje svih slika koje korisnik postavi na profilu.

```
const app = express();

app.use(express.static("public"));
app.use("/images", express.static("images"));
```

Slika 2 Express logo

2.3 MongoDB

MongoDB je više platformski program baze podataka orijentiran na dokumente koji su dostupni na izvornom jeziku. Klasificiran kao NoSQL program baze podataka, MongoDB koristi dokumente slične JSON-u s izbornim shemama. MongoDB je NoSQL program za upravljanje bazom podataka otvorenog koda. NoSQL se koristi kao alternativa tradicionalnim relacijskim bazama podataka. NoSQL baze podataka vrlo su korisne za rad s velikim skupovima distribuiranih podataka [2]. MongoDB je alat koji može upravljati informacijama o dokumentima, pohranjivati ili dohvaćati informacije. Na slici 3 ispod prikazano je spajanje strane poslužitelja s MongoDB-em. Spajanje je učinjeno unošenjem podataka veze koje je dobiveno s njihove službene stranice .

Slika 3 MongoDB logo

2.4 NodeJs

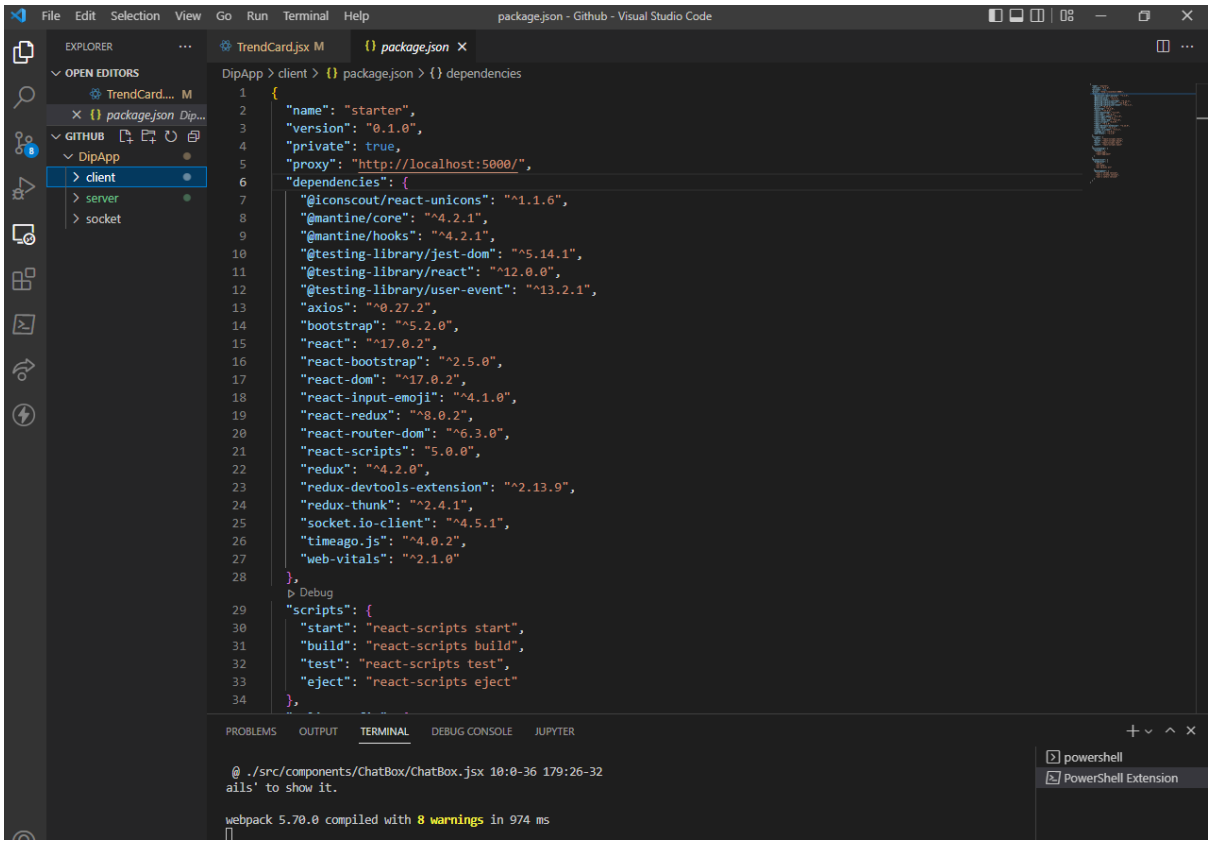
Node.js je okruženje za izvršavanje JavaScripta otvorenog koda i više platformi. To je popularan alat za gotovo sve vrste projekata. Node.js pokreće V8 JavaScript mehanizam, jezgru Google Chromea, izvan preglednika [4]. To omogućuje Node.js-u da bude vrlo učinkovit. Aplikacija Node.js radi u jednom procesu. Node.js pruža skup asinkronih I/O primitiva u svojoj standardnoj biblioteci koje sprječavaju blokiranje JavaScript koda i općenito su biblioteke u Node.js napisane korištenjem ne blokirajućih paradigmi, čineći ponašanje blokiranja iznimkom, a ne normom. Kada Node.js izvodi I/O operaciju, poput čitanja s mreže, pristupa bazi podataka ili datotečnom sustavu, umjesto blokiranja niti i trošenja CPU ciklusa na čekanje, Node.js će nastaviti s operacijama kada se odgovor vrati. To omogućuje Node.js da rukuje tisućama istodobnih veza s jednim poslužiteljem bez uvođenja tereta upravljanja istovremenošću niti, što bi moglo biti značajan izvor grešaka. Node.js ima jedinstvenu prednost jer milijuni frontend programera koji pišu JavaScript za preglednik sada mogu pisati kod na strani poslužitelja uz kod na strani klijenta bez potrebe da uče potpuno drugačiji jezik [1]. U Node.js novi ECMAScript standardi mogu se koristiti bez problema, jer ne morate čekati da svi korisnici ažuriraju svoje preglednike – korisnici su zaduženi da odlučuju koju će ECMAScript verziju koristiti promjenom Node.js verzije, a također može se omogućiti određene eksperimentalne značajke pokretanjem Node.js sa zastavicama. Na slici 4 ispod prikazan je Node.js u kreiranju porta i povratne poruke pri pokretanju.

```
const express = require("express");
const app = express();

app.get("/", (req, res) => {
  res.send("Success!");
});

const PORT = process.env.PORT || 8080;

app.listen(PORT, console.log(`Server started on port ${PORT}`));
```

Slika 4 Node.js logo

2.5 Visual Studio Code

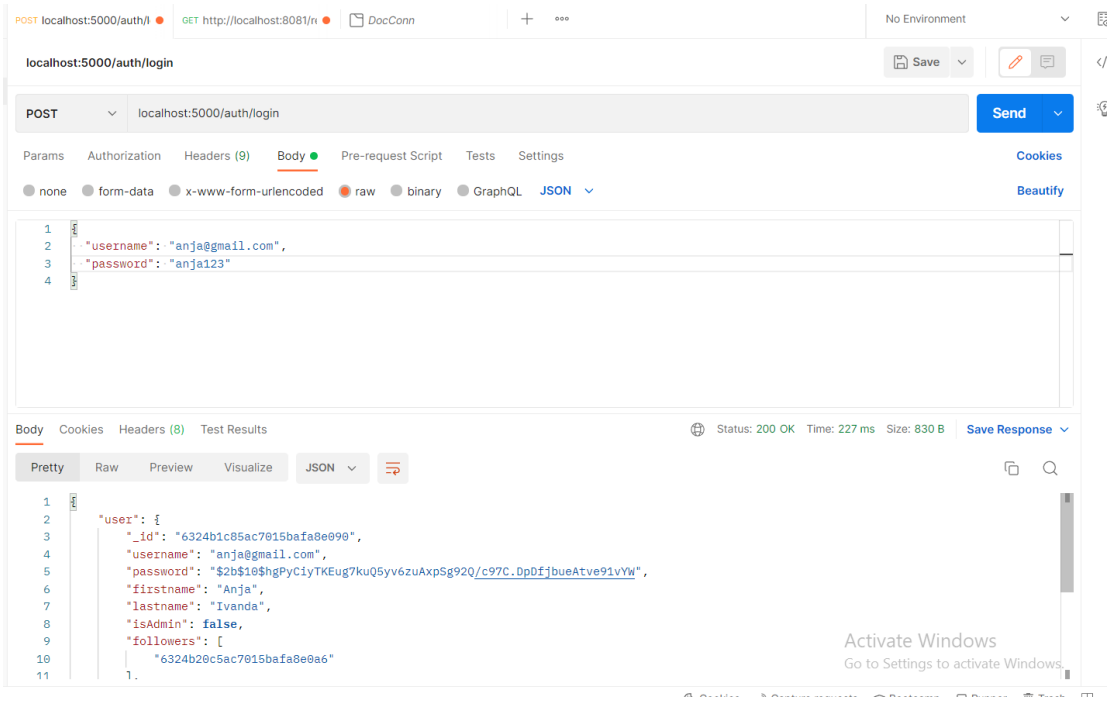
Visual Studio code je uređivač koda koji je izradio Microsoft za Windows , Linux , macOS. Značajka njega je što uključuje podršku za otklanjanje pogreški, isticanje sintakse refaktoriranje koda i ugrađeni Git s kojim je omogućeno postavljati nove promjene na vlastiti github nakon dovršetka rada . Na slici 5 prikazan je izgled programa Visual Studio u radu koji se sastoji od terminala, alatne trake, lijevog izbornika koji sadrži popis datoteka i srednjeg prozora koji prikazuje otvorene datoteke.



Slika 5 Visual Studio Code

2.6 Postman

Postman je aplikacija koja omogućuje testiranje API-ja korištenjem grafičkog korisničkog sučelja. Neke od prednosti Postmana uključuju značajku prikupljanja i mogućnost stvaranja različitih okruženja za testiranje. Postman je user-friendly alat koji nam pomaže optimizirati vrijeme prilikom izvođenja testova. Postman je aplikacija koja nam omogućuje testiranje API-ja korištenjem grafičkog korisničkog sučelja što je i prikazano na slici 6. Neke od prednosti Postmana uključuju značajku prikupljanja i mogućnost stvaranja različitih okruženja za testiranje. Postman je user-friendly alat koji nam pomaže optimizirati naše vrijeme prilikom izvođenja testova. Na slici 6 prikazano je testiranje login forme.



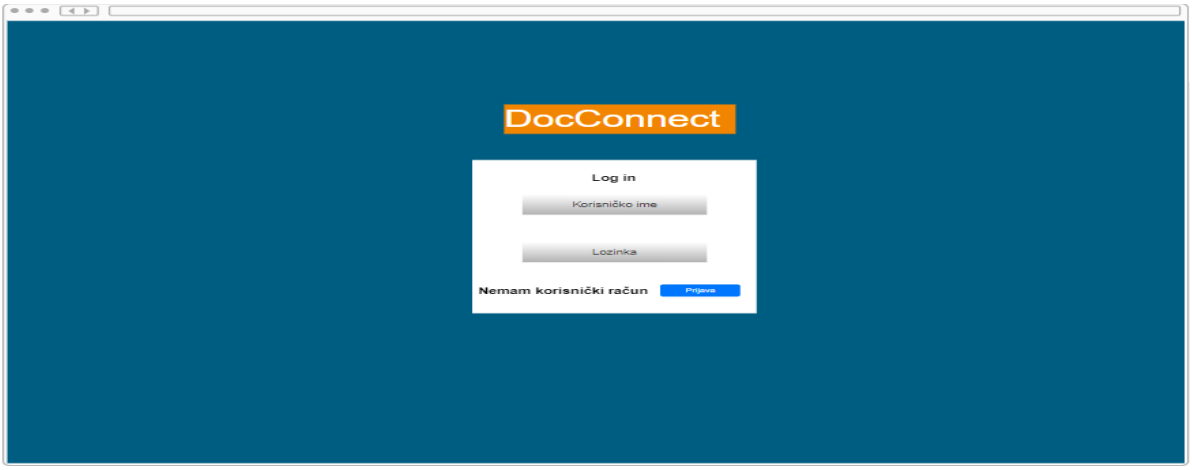
Slika 6 Postman

3. PROTOTIP STRANICE

Svaka stranica prije svoje implementacije u stvarni svijet i prije samog pisanja koda mora početi od nekakve zamisli i ideje koju sam kreator u tome trenutku ima. Za realizaciju misli u dijela prvo za čime posežemo je izrada nekakve skice u ovom slučaju prototipa stranice. Prototip stranice je vizualno predočavanje programerima kako bi sama stranica trebala izgledati i koje bi funkcionalnosti trebala sadržavati. Postoje mnogi alati koji im u tome pomažu, a to su Adobe XD, Figma i mnogi drugi. Konkretno za izradu ovog diplomskog rada korištena je Proto.io. Kao što je u samom uvodu rečeno ova stranica zamišljena je da izgleda kao svaka društvena mreža. Kao takva sadrži stranice za prijavu, glavnu stranicu, profil stranicu, stranicu s porukama i stranicu za hitne slučajeve.

3.1 Stranica za prijavu

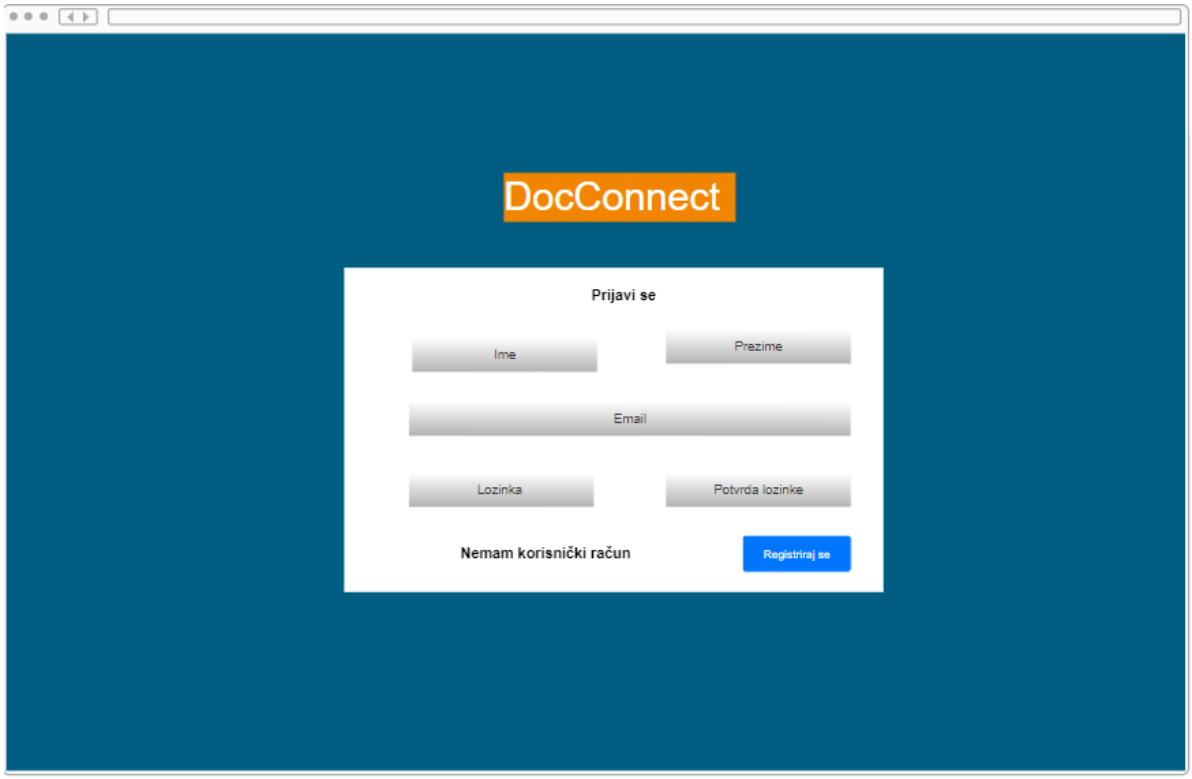
Slika 7 prikazuje stranicu za prijavu koja se sastoji od naslova, korisničkog imena i lozinke te samog gumba koji služi za prijavu na stranicu, ako korisnik nema postojeći korisnički račun tada klikom na gumb "Nemam korisnički račun „ bude proslijeđen na stranicu za prijavu.



Slika 7 Prototip „Login" stranice

3.2 Stranica za registraciju

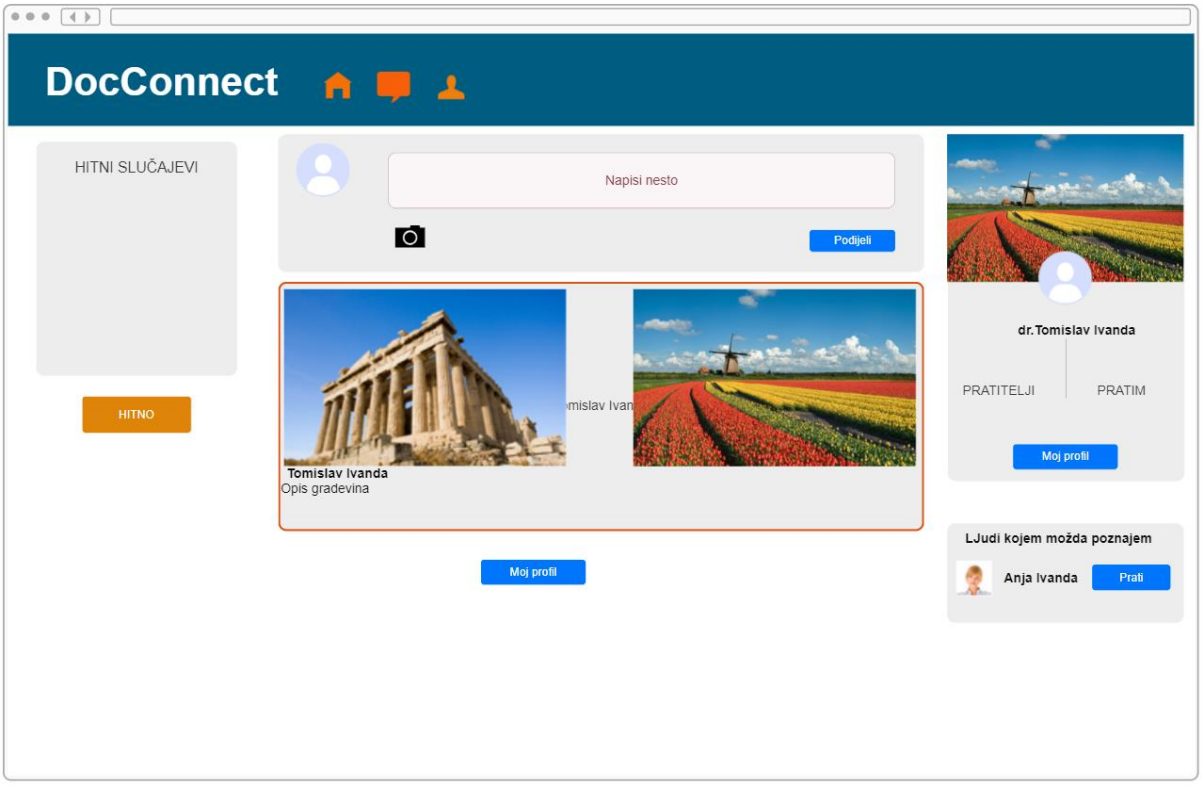
Slika 8 prikazuje stranicu za registraciju koja se sastoji od imena, prezimena, e-maila, lozinke i potvrde lozinke. Sam korisnik unosi svoje podatke te nakon toga korištenjem e-maila i lozinke vraćanjem na login stranicu unosi svoje podatke i pristupa svome korisničkom računu.



Slika 8 Prototip „Signup“ stranice

3.3 Početna stranica

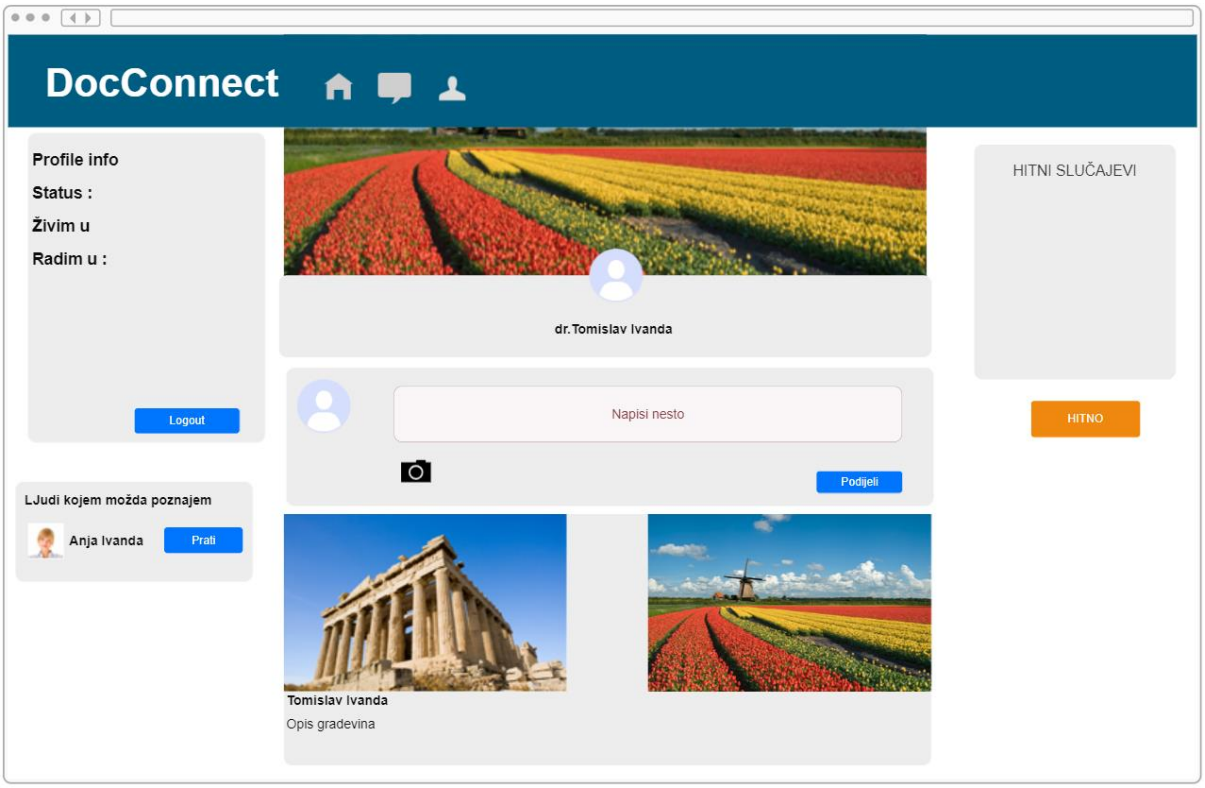
Početna se stranica sastoji od gornje trake, glavnog dijela koji se dijeli na lijevu, srednju i desnu sekciju kao što je prikazano na slici 9. Gornja se strana sastoji od navigacijske trake koja sadrži gumb pretraživanja, naziv stranice i gumbova za odlazak na neku od stranica. Lijeva strana glavne sekcije sastoji se od kartice za hitne slučajeve gdje će svaki korisnik moći klikom na gumb „Hitno“ otići na posebnu stranicu i vidjeti sve hitne slučajeve koji su drugi liječnici stavili. Srednja strana glavne sekcije sastoji se od samih objava koje su napisali svi korisnici mreže i od dijela gdje sami korisnik može napisati svoju osobnu objavu. Desni dio sastoji se od slike profila korisnika, gumba za odlazak na profil i posebne kartice koja predlaže ostala liječnike koji se koriste ovom stranicom .



Slika 9 Prototip „Home“ stranice

3.4 Profilna stranica

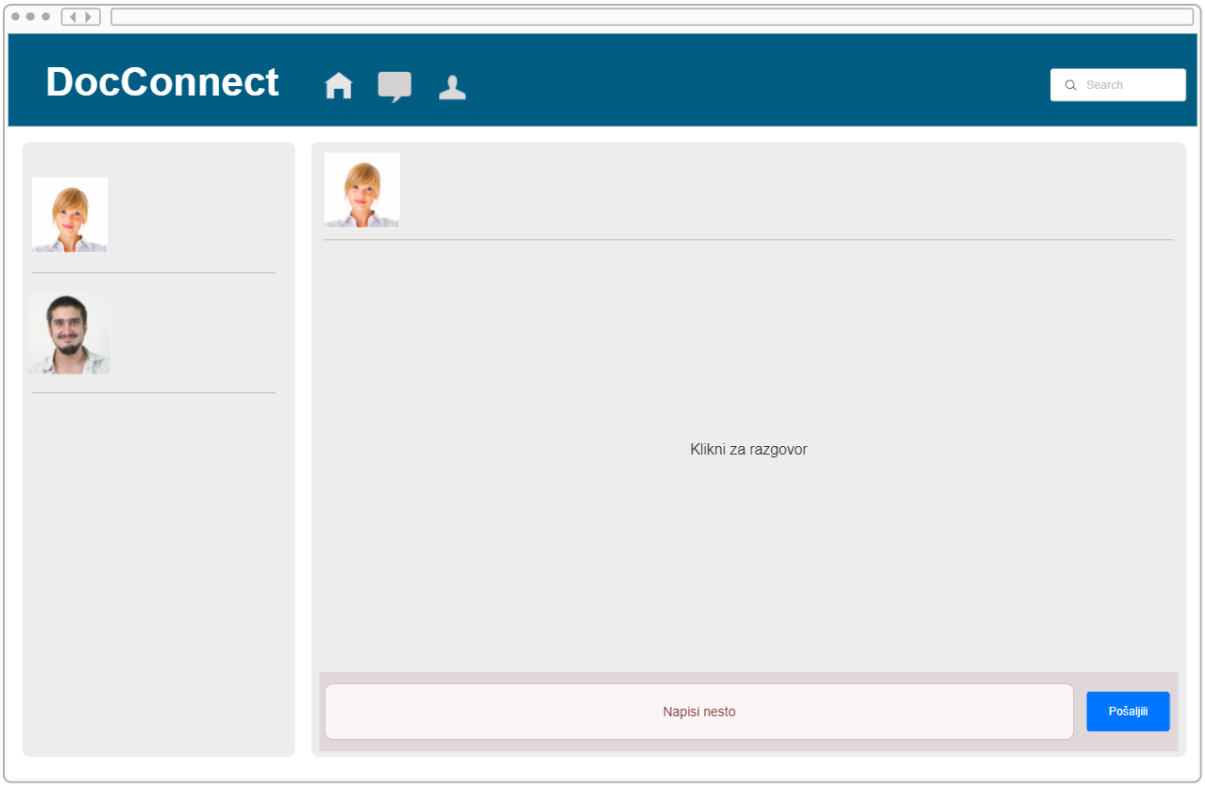
Profilna se stranica sastoji isto kao i početna stranica od gornje trake, glavnog dijela koji se dijeli na lijevu, srednju i desnu sekciju prikazano na slici 10. Gornja se strana sastoji od navigacijske trake koja sadrži gumb pretraživanja, naziv stranice i gumbova za odlazak na neku od stranica . Lijeva strana glavne sekcije sastoji se od podataka samoga korisnika, gumba za odjavu korisnika sa stranice i prikaza liječnika koje se može pratiti. Srednja strana glavne sekcije sastoji se od slike korisnika i objava koje je sami korisnik kreirao. Desna strana glavne sekcije sastoji se od kartice za hitne slučajeve gdje će svaki korisnik moći klikom na gumb „Hitno“ otići na posebnu stranicu i vidjeti sve hitne slučajeve koji su drugi liječnici učitali u sustav.



Slika 10 Prototip „Profilna“ stranice

3.5 Stranica s porukama

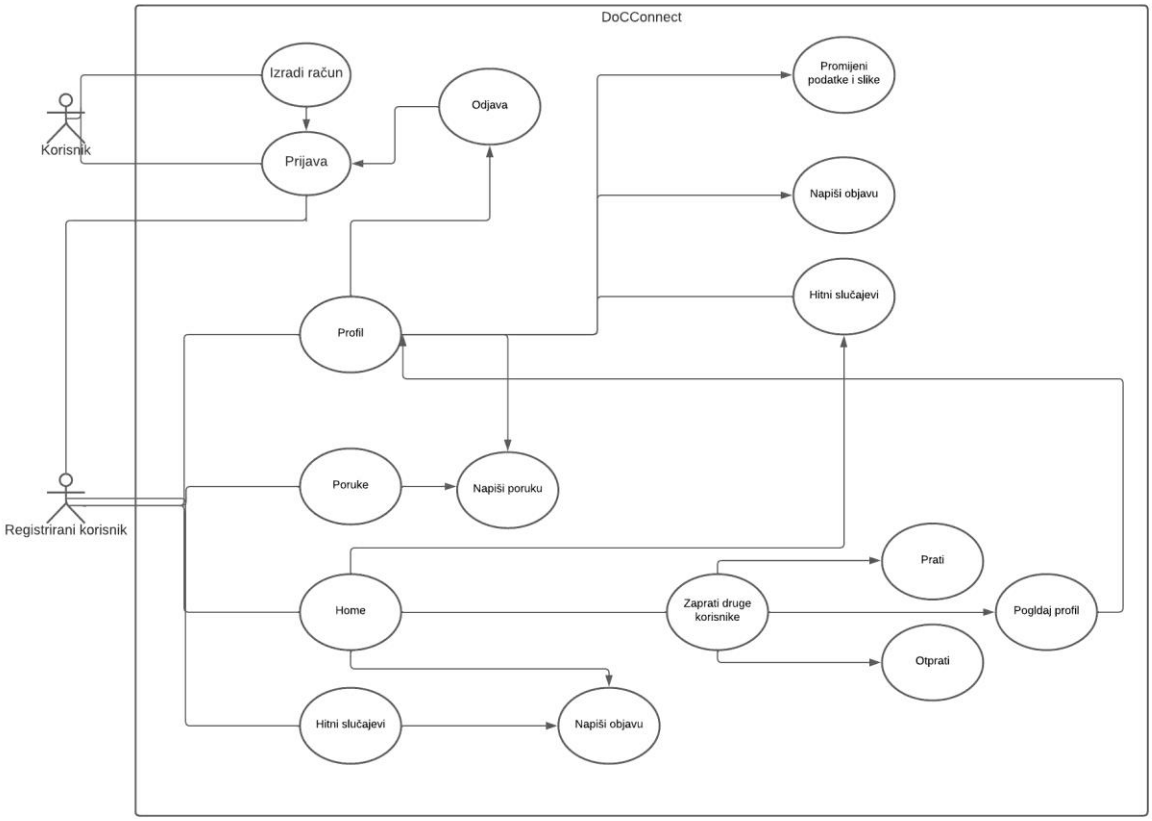
Stranica s porukama sadrži gornju traku kao i sve ostale glavne stranice. Glavna sekcija sastoji se od lijeve strane koja sadrži popis osoba s kojima korisnik razgovara, dok desna strana sadrži sekciju samoga razgovora što se može vidjeti na slici 11 .



Slika 11. Prototip „Message“ stranice

3.6 Dijagram obrasca upotrebe

Dijagram obrasca upotrebe prikazuje kako se sustav ponaša pri samoj interakciji sa akterima koji koriste aplikaciju. Na slici 12 je vidljivo kako samo prijavljeni korisnici mogu koristi sve funkcionalnosti koje sadržava aplikacija. U samim krugovima su prikazane funkcije i kako se one dalje granaju.



Slika 12. Dijagram obrasca upotrebe

4.STRANA KLIJENTA

Klijent strana ovog projekta zapravo je korisničko sučelje na kojem se radi izgled stranice i dizajn. Ovdje su smještene sve stranice koje su potrebne za izradu aplikacije. Korisničko sučelje najbolje se može opisati kao stranica koju vidi krajnji korisnik kada pristupi stranici, a za sve one detalje oko izgleda zaslužan je JavaScript. Što se tiče dizajna same stranice korišten je stilski jezik CSS(eng. Cascading Style Sheets). Ukratko, CSS je dizajnerski jezik koji web stranicu čini privlačnijom od običnih. Na slici 13 može se vidjeti da se korisničko sučelje ovog projekta sastoji od više mapa. Svaki projekt rađen u React-u ili Vue.js-u sadrži gotovo iste mape, a to su node_modules, public , src, package-json. Ove sve datoteke osim src mape vidjet će se i na poslužiteljskoj strani ove aplikacije.

Slika 13. „Client“ mapa

4.1 Node_modules

Datoteka Node_modules može se zamisliti kao pred memoriju za vanjske module o kojima ovisi projekt. Kada ih npm instalira, oni se preuzimaju s weba i kopiraju u mapu node_modules, a Node.js osposobljen je da ih tamo traži kada ih se uveze (bez određenog puta). Nazvati se može pred memorijom jer se mapa node_modules može potpuno ponovno stvoriti od nule u bilo kojem trenutku samo ponovnim instaliranjem svih zavisnih modula.

4.2 Public mapa

Ova mapa sadrži 3 datoteke kao što su. favicon.ico - to je datoteka ikone koja sadrži ikonu koja se prikazuje u pregledniku. index.html - to je važna datoteka. Zbog ove datoteke, javnu mapu, poznatu kao „root mapa", na kraju poslužuje web poslužitelj. Index.html je jedna HTML stranica prisutna u ovom projektu.

4.3 Package-json

Datoteka package.json neka je vrsta manifesta za ovaj projekt. Može učiniti puno stvari, potpuno nepovezano. Na primjer, to je središnje spremište konfiguracije za alate. To je također mjesto gdje npm i yarn pohranjuju imena i verzije za sve instalirane pakete. Ako se rade neke ručne preinake package-json-u potrebno je obrisati node_modules datoteku i ponovno ju instalirati.

4.4 Src datoteka

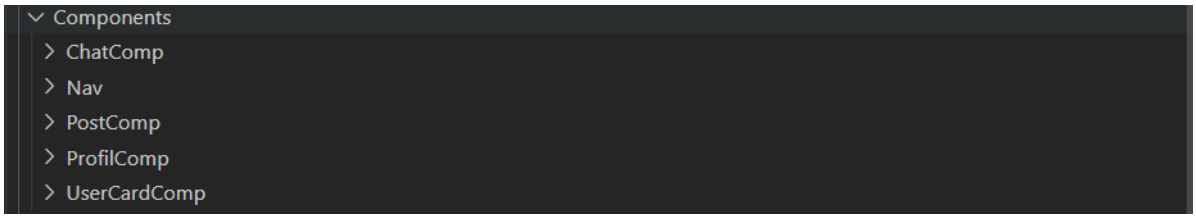
Ova mapa sadrži stvarni izvorni kod za programere. Ovo je mjesto gdje je prisutna React aplikacija. Unutar nje mogu se stvoriti pred direktoriji unutar ovog direktorija. Za bržu ponovnu izgradnju, datoteke unutar ove mape obrađuju Webpack. Na slici 14 može se vidjeti struktura datoteke src koja u sebi sadrži:

- App.css - ova datoteka daje neke CSS klase ili možemo reći neki stil koji koristi App.js datoteka.
- App.js - je ogledna React komponenta pod nazivom "App" koju dobivamo besplatno prilikom izrade nove aplikacije.
- Indeks.js - pohranjuje aplikacijski glavni Render poziv iz ReactDOM-a, a ona vozi komponentu App.js s kojom se počinje i govori Reactu gdje je mora renderirati.
- Components – je mapa u koju spremamo i kreiramo izglede same stranice . One služe za lakše razdvajanje dijelova stranice na manje cjeline, a one se onda pozivaju na glavnu stranicu preko export defaulta. Uz sve komponente dolazi i njihova CSS datoteka u koju svaku tu kreiranu komponentu kasnije dizajniramo kako bi oku bila što ljepša.

Slika 14. " Src" mapa

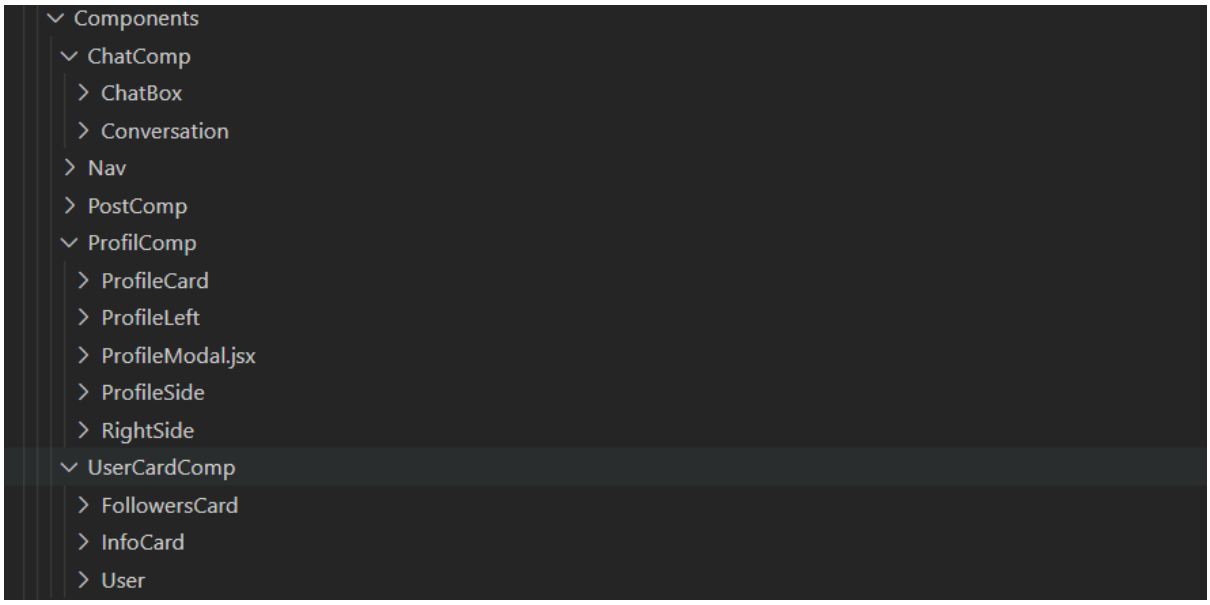
4.4.1 Komponente

Za lakše pisanje koda, za lakšu preglednost cijelog projekta i ne pretrpavanje pojedinačnih datoteka koriste se komponente koje su prikazane na slici 15. Pri izradi komponente na zadnju liniju koda stavljamo export default i naziv komponente. Export default služi za pozivanje datoteke u neku drugu u kojoj će se sama komponenta prikazati. Sami rad komponenti svodi se na to da se nakon kreiranja njene jsx i css datoteke ona poziva u neku od glavnih stranica. Najbolji primjer toga je sama datoteka MainPages koja se sastoji od glavnih stranica ove aplikacije Home,Profile, Emergency , Message .



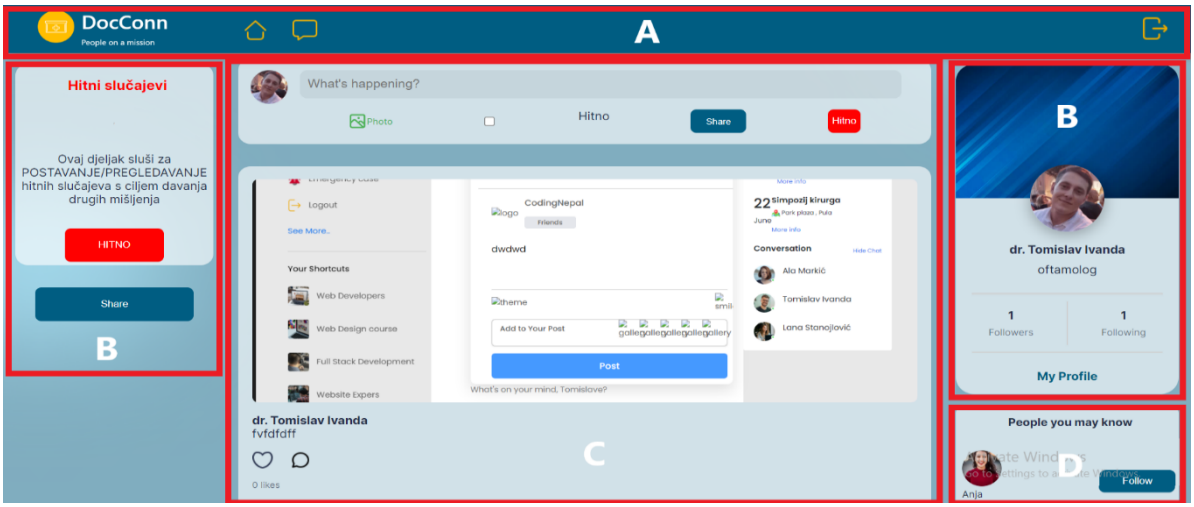
Slika 15. „Components“ mapa

Datoteka Component sastoji se od više mapa koje u sebi sadrže određene datoteke. Naziv mapa sa slike 16, sugerira na to o kojem se dijelu prikaza radi.



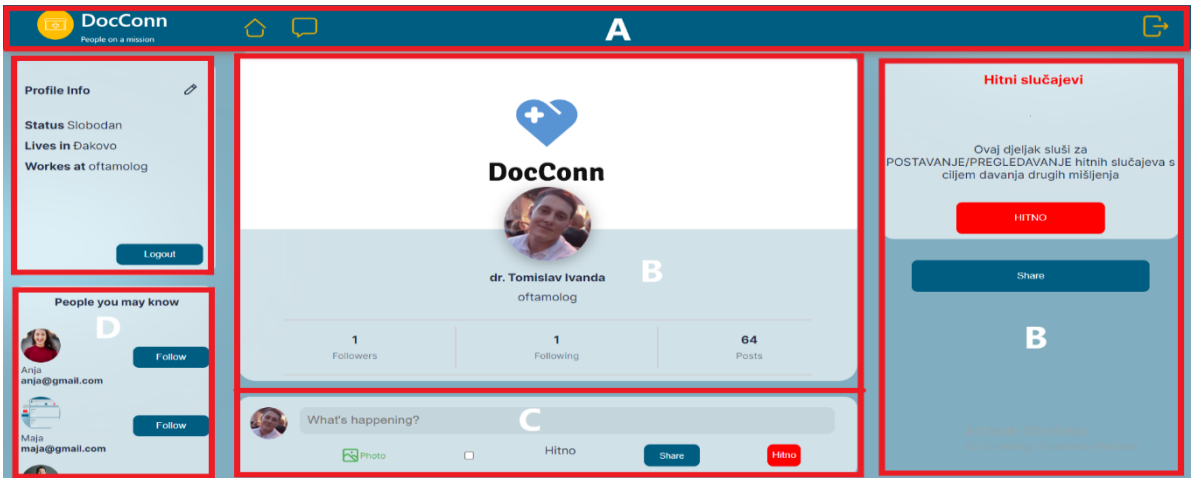
Slika 16. Prikaz svih komponenti

Na slikama 17, 18 i 19 vidi se prikaz stranice podijeljene na zasebne komponente, a u daljnjem tekstu objasnit će se koje komponente su pripojene kojem dijelu Home, Profil, Message, Emergency stranice. Ovdje je prikazana Home stranica koja se sastoji od Nav, ProfilComp, PostComp i UserCardComp komponenti.



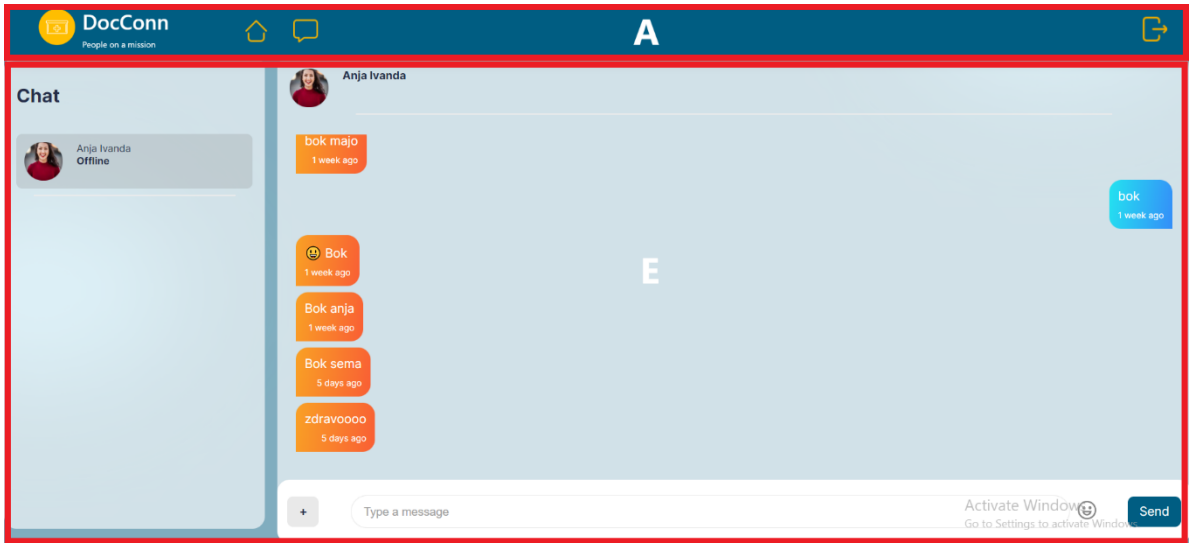
Slika 17. Home stranica podijeljena na komponente

Profile stranica sadrži Nav, ProfileComp, PostComp, UserCardComp komponente.



Slika 18. Profil stranica podijeljena na komponente

Message stranica jedina se razlikuje od ostalih zato što ne sadrži toliko komponenti kao prethodne, a jedina komponenta koja ju veže s ostalima je Nav komponenta. Od ostalih komponenti ona sadrži samo ChatComp



Slika 19. Message stranica podijeljena na komponente

A- Nav – ova komponenta kao što joj i skraćeni naziv kaže „nav“ je navigacijska traka koja se obično nalazi na gornjem dijelu stranice i služi za brzi prelazak sa stranice na stranicu

B- ProfileComp sastoji se od ProfileCard, ProfileRight, ProfileModal, ProfileSide, RightSide,EmergencyCard mape. Ove komponente prikazuju profilnu stranicu ove web aplikacije i omogućuju funkcionalnosti kao dodavanje slika, promjena slika, promjena opisa profila.

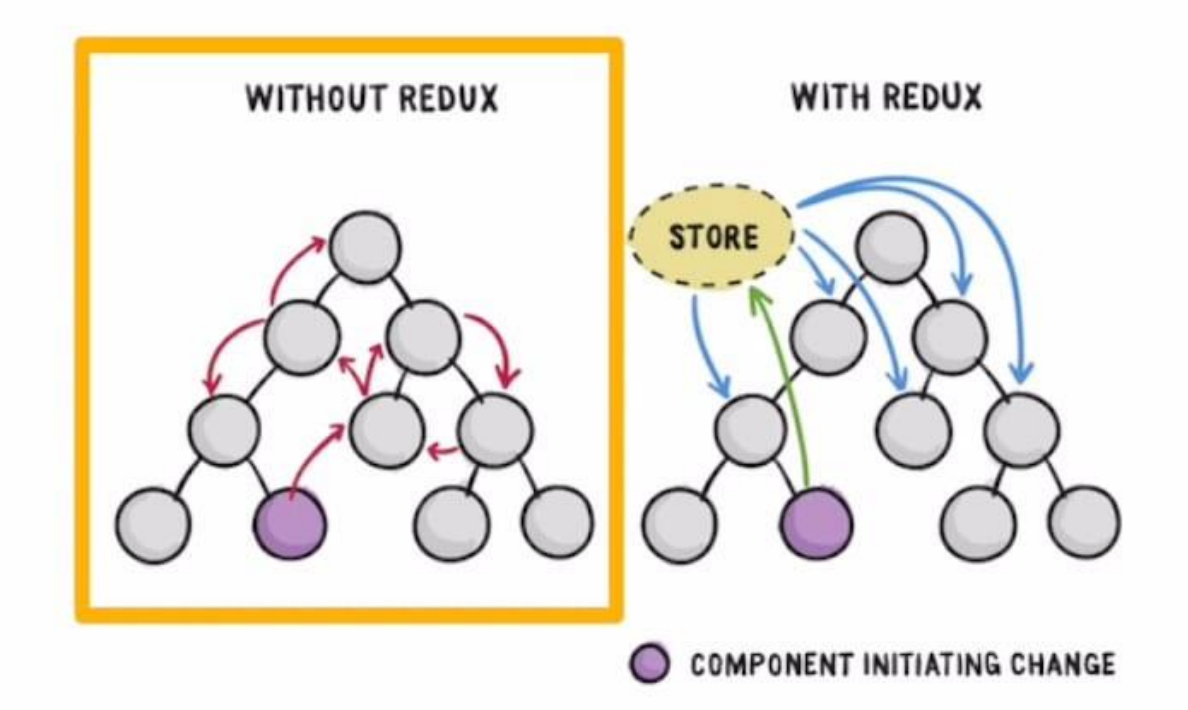
C- PostComp – sastoji se od Post, Posts, PostShare i PostSide mape . Sve ove komponente čine objavu vidljivom na zaslonu , a to je središnji dio profile i home page stranice. Ove su komponente zaslužne za to da krajnji korisnik vidi objave drugih korisnika i svoje vlastite dok sami PostShare omogućuje korisniku kreiranje vlastitih objava kroz svoje funkcionalnosti.

D- UserCardComp – sastoji se od FollowersCard, User, InfoCard mapa. Ove komponente prikazuju manje stavke prikaza kao što su lista ljudi koji prate trenutnog korisnika , kartica s informacijama o broju pratitelja .

E- ChatComp – sastoji se od ChatBox i Conversation mape . U njoj je kreiran message box koji prikazuje stranicu samih poruka između dva korisnika. Ove se komponente pozivaju na glavnu Chat stranicu.

4.4.2 Redux

Redux je JavaScript biblioteka otvorenog koda za upravljanje i centraliziranje stanja aplikacije. Najčešće se koristi s knjižnicama kao što su React ili Angular za izgradnju korisničkih sučelja [10]. Reduktori traže određenu radnju za promjenu ili stvaranje novog globalnog stanja i prateći ovu akciju mogu primiti stanje s dvije strane ili od komponente react ili od API-ja. Na slici 20 vidi se prikaz rada aplikacije bez reduxa i s reduxom.



Slika 20. Without store, With store

Izvor : https://levelup.gitconnected.com/a-reduced-explanation-of-redux-reducers-actions-and-store-9b7819c5d064?gi=34e5b7bde80, 2022

4.4.3 Actions

Actions su običan JavaScript objekt koji sadrži informacije. Actions su jedini izvor informacija za store. Actions imaju polje vrste koje govore koju vrstu akcije treba izvršiti, a sva ostala polja sadrže informacije ili podatke. Komponenta iz react-a prima globalna stanja iz stora i u slučaju da bilo koja komponenta želi manipulirati bilo kojom vrstom globalnog stanja tada će se pozvati određenu vrstu akcije. Akcija kao funkcija koja je povezane sa storom koristi dispatch. Za instaliranje ovoga paketa korisnik mora instalirati redux paket (npm i redux redux-thunk react-redux). U sintaksi, Action objekt ima dva svojstva. Prvo svojstvo je "svojstvo tipa" i treba ga definirati u ograničenju niza i obavezno je uključiti svojstvo tipa u objekt. Drugi je "payload", Payload pohranjuje dodatne informacije o tome što se dogodilo [5]. Akcije nisu ništa drugo nego prijenosnici podataka iz vaše aplikacije u trgovinu. Datoteke koje sadrži Action datoteka :

- **AuthenticationAction.js** (logIn, signUp, logOut)
- **PostAction.js** (getTimelinePosts, createComment, deleteComment)
- **UploadAction.js** (uploadImage, uploadPost)
- **UserAction.js** (updateUser,followUser,unFollowUser)

4.4.3.1 AuthenticationAction.js

Authentication akcija pokreće provjeru koja se akcija treba izvršiti, a u ovom slučaju kao na slici 21, a to su registracija i prijava. U slučaju da je registracija/prijava u red u tip akcije je „AUTH_SUCCES“, a ako jedošlo do krivo unesenih informacija tip je „AUTH_FAIL“.

```
export const logIn = (formData, navigate) => async (dispatch) => {
  dispatch({ type: "AUTH_START" });
  try {
    const { data } = await AuthApi.logIn(formData);
    dispatch({ type: "AUTH_SUCCESS", data: data });
    navigate("../home", { replace: true });
  } catch (error) {
    console.log(error);
    dispatch({ type: "AUTH_FAIL" });
  }
};

export const signUp = (formData, navigate) => async (dispatch) => {
  dispatch({ type: "AUTH_START" });
  try {
    const { data } = await AuthApi.signUp(formData);
    dispatch({ type: "AUTH_SUCCESS", data: data });
    navigate("../home", { replace: true });
  } catch (error) {
    console.log(error);
    dispatch({ type: "AUTH_FAIL" });
  }
};

export const logOut = ()=> async(dispatch)=> {
  dispatch({type: "LOG_OUT"})
}
```

Slika 21. AuthenticationAction.js

4.4.3.2 PostAction.js

Post action sadrži provjeru kreiranog komentara kao i provjeru brisanja komentara koji su vidljivi na slici 22. U slučaju da je sve u redu i da se ne događa problem pro izvršavanju vraća se „RETREIVING SUUCCES“, a u drugom slučaju dobije se „RETRIEVING FAIL“

```
export const getTimelinePosts = (id) => async (dispatch) => {
  dispatch({ type: "RETREIVING_START" });
  try {
    const { data } = await PostApi.getTimelinePosts(id);

    dispatch({ type: "RETREIVING_SUCCESS", data: data });
  } catch (error) {
    console.log(error);
    dispatch({ type: "RETREIVING_FAIL" });
  }
};

export const createComment = (postId, userId, text) => async (dispatch) => {
  try {
    const { data } = await PostApi.createComment(postId, userId, text);

    dispatch({ type: "CREATE COMENT", data: data, postId: postId });
  } catch (error) {
    console.log(error);
    dispatch({ type: "RETREIVING_FAIL" });
  }
};

export const deleteComment = (postId, userId) => async (dispatch) => {
  try {
    const { data } = await PostApi.deleteComment(postId, userId);

    dispatch({ type: "DELETE COMENT", data: data, postId: postId });
  } catch (error) {
    console.log(error);
    dispatch({ type: "RETREIVING_FAIL" });
  }
};
```

Slika 22. PostAction

4.4.3.3 UploadAction.js

Upload akcija koja se nalazi na slici 23 prikazuje pozivanje dvije funkcije koje su zadužene za upload slike i postanje posta , a to su uploadImage i uploadPost koje se nalaze u uploadRequestu.

```
import * as UploadApi from "../api/UploadRequest";

export const uploadImage = (data) => async (dispatch) => {
  try {
    console.log("SLika je ucitana")
    await UploadApi.uploadImage(data);
  } catch (error) {
    console.log(error);
  }
};

export const uploadPost = (data) => async (dispatch) => {
  dispatch({ type: "UPLOAD_START" })
  try {
    const newPost =await UploadApi.uploadPost(data)
    dispatch({ type: "UPLOAD_SUCCESS", data: newPost.data })
  } catch (error) {
    console.log(error);
    dispatch({ type: "UPLOAD_FAIL" })
  }
};
```

Slika 23. UploadAction

4.4.3.4 UserAction.js

User action sadržava provjeru za ažuriranja podataka o korisniku i kakvo je stanje u vezi praćenja osoba drugih korisnika . Ovisno o situaciji izvršavanja akcije dobije se povratna informacija u obliku type i ona govori je li sve u redu ili pokušaj nije uspio što se vidi sa slike 24.

```
export const updateUser = (id, formData) => async (dispatch) => {
  dispatch({ type: "UPDATING_START" });
  try {
    const { data } = await UserApi.updateUser(id, formData);

    dispatch({ type: "UPDATING_SUCCESS", data: data });
  } catch (error) {
    dispatch({ type: "UPDATING_FAIL" });
  }
};

export const followUser = (id, data) => async (dispatch) => {
  dispatch({ type: "FOLLOW_USER" });
  UserApi.followUser(id, data);
};

export const unFollowUser = (id, data) => async (dispatch) => {
  dispatch({ type: "UNFOLLOW_USER" });
  UserApi.unFollowUser(id, data);
};
```

Slika 24. UserAction

Kod Action datoteka radi se o principu da se prvo kreira obrazac ovisno o kakvom se zahtjevu radi, a sami su podaci obrasca parametri prijave koji su zaprimljeni od AuthenticationAction, PostAction, UploadAction, UserAction datoteka i oni će biti asinkroni poziv samog redux thunk-a. Zato se ovdje svugdje nalazi dispatch. U samom Action-u mora se odvijati komunikacija s Api-em jer će on pozvati kranju točku prema poslužitelju ovisno o tome o kojoj se datoteci u Action-u radi. Da bi se radilo sa zahtjevima, mora se instalirati paket axios. Ovaj se paket u osnovi koristi za slanje zahtjeva na client strani samog poslužitelja. Prije korištenja axiosa naprave se preinake u package.json-u i proxy se postavi na port 5000.

4.4.4 API

API ima ulogu da šalje novo stanje prema akciji, a akcija šalje ovaj zahtjev reduktoru ako se govori o reakcijskoj komponenti. Unutar api-a kreira se osnovni URL samog proxyja koji za sami api znači da svaki put kada korisnik napravim zahtjev on će pozvati broj porta 5000 koji je zadan. Dolje na sljedećim slikama prikazane su datoteke :

- **AuthenticationRequest.js** (logIn, signUp)
- **ChatRequest.js** (userChat)
- **MessageRequest.js** (getMessage, addMessage)
- **PostRequest.js** (getTimelinePosts, createComment, deleteComment)
- **UploadRequest.js** (uploadImage, uploadPost)
- **UserRequest** (getUser,updateUser, getAllUser, followUser, unFollowUser

Svaka Api datoteka sadrži osnovni url , što znači svaki put kada se napravi zahtjev bit će pozvan port 5000 koji je ranije dodan u package.json. Prave se funkcije za prijavu, chat, message, post, upload request i user follow,

4.4.4.1 AuthenticationRequest.js

Da bi se radilo sa zahtjevima u svaku datoteka u API-u treba importat axios koji se prethodno instalira. Axios služi za upućivanje zahtjeva poslužitelj strani. Na slici 25 ispod prikazano je kreiranje funkcije za prijavu i registraciju . Podaci prijave i registracije moraju biti jednaki podacima obrasca koji se primanju od strane akcije.

```
import axios from 'axios'

const API = axios.create({ baseURL: 'http://localhost:5000' });

export const logIn= (formData)=> API.post('/auth/login',formData);

export const signUp = (formData) => API.post('/auth/register', formData);
```

Slika 25. AuthenticationRequest

4.4.4.2 ChatRequest.js

ChatRqeust prikazuje kreiranje funkcije za korisničke razgovore kao što se može vidjeti iz slike 26.

```
import axios from 'axios'

const API = axios.create({baseURL: 'http://localhost:5000'})

export const userChats = (id) => API.get(`/chat/${id}`)
```

Slika 26. ChatRequest

4.4.4.3 MessageRequest.js

Na slici 27 prikazan je MessageReques koji prikazuje kreiranje funkcije za dobivanje poruka i dodavanje poruka.

```
import axios from "axios";

const API = axios.create({ baseURL: "http://localhost:5000" });

export const getMessages = (id) => API.get(`/message/${id}`);
export const addMessage = (data) => API.post("/message/", data);
```

Slika 27. MessageRequest

4.4.4.4 PostRequest.js

PostRequest prikazuje kreiranje funkcije za dobivanje samih postova preko određenog id-a , kreiranje komentara, brisanje komentara i brisanje posta što je prikazano na slici 28.

```
import axios from "axios";

const API = axios.create({ baseURL: "http://localhost:5000" });

export const getTimelinePosts = (id) => API.get(`/post/${id}/timeline`);

export const createComment = (postId, userId, text) =>
  API.post(`/post/${postId}/${userId}/comment`, { text: text });

export const deleteComment = (postId, commentId) =>
  API.delete(`/post/${postId}/${commentId}`);

//proba
export const deletePost = (postId) => API.delete(`/post/${postId}`);
```

Slika 28. PostRequest

4.4.4.5 UploadRequest.js

UploadRequest prikazuje kreiranje funkcije za učitavanje slika i posta prikazano na slici 29.

```
import axios from "axios";

const API = axios.create({ baseURL: "http://localhost:5000" });

export const uploadImage = (data) => API.post("/upload", data);
export const uploadPost = (data) => API.post("/post", data);
```

Slika 29. UploadRequest

4.4.4.6 UserRequest.js

UserRequest prikazuje kreiranje funkcije za dobivanje svih korisnika ,dobivanje jednog korisnika i prati/otprati korisnika prikazano na slici 30.

```
const API = axios.create({ baseURL: "http://localhost:5000" });

export const getUser = (userId) => API.get(`/user/${userId}`);
export const getAllUser = () => API.get("/user");
export const followUser = (id, data) => API.put(`/user/${id}/follow`, data);
export const unFollowUser = (id, data) => API.put(`/user/${id}/unfollow`, data);
```

Slika 30 UserRequest

4.4.5 Reducer

Reducer je funkcija koja prihvaća trenutno stanje i radnju te vraća novo stanje. Svakom radnjom/operacijom upravlja jedan ili više reduktora koji ažuriraju jedan store. Kao najbolji primjer toga može se zamisliti prava trgovina u koju se stavljaju različite stvari za buduću upotrebu. Trgovina je beskorisna ako se stvari nasumično stave. Osoba može provesti sate unutra, ali možda neće pronaći ništa. Jednostavno rečeno, reduktori upravljaju načinom na koji se podaci čuvaju u pohrani koju pružaju akcije [10]. Datoteka Reducer sastoji se od:

- **AuthenticationReducer.js**
- **PostReducer.js**

Na slikama 30 i 31 vidimo redukciju koja je će biti povezana u cilju komunikacije s reduktorima slanjem određene vrste radnji. Svaka datoteka koja se nalazi u reduceru sadrži podatke za provjeru stanja korisnika u AuthenticationrReduc.js i objava u PostReduceru.js. Radnja unutar ovih datoteka u osnovi govori koje vrste reduktora su pozvane od radnji i unutar korisnika koja je vrsta akcije pozvana. U globalu se koristi naredba switch koja ovisi o tipu akcije i unutar njih rade se provjere (Authntication,Post..).

4.4.5.1 AuthenticationReducer.js

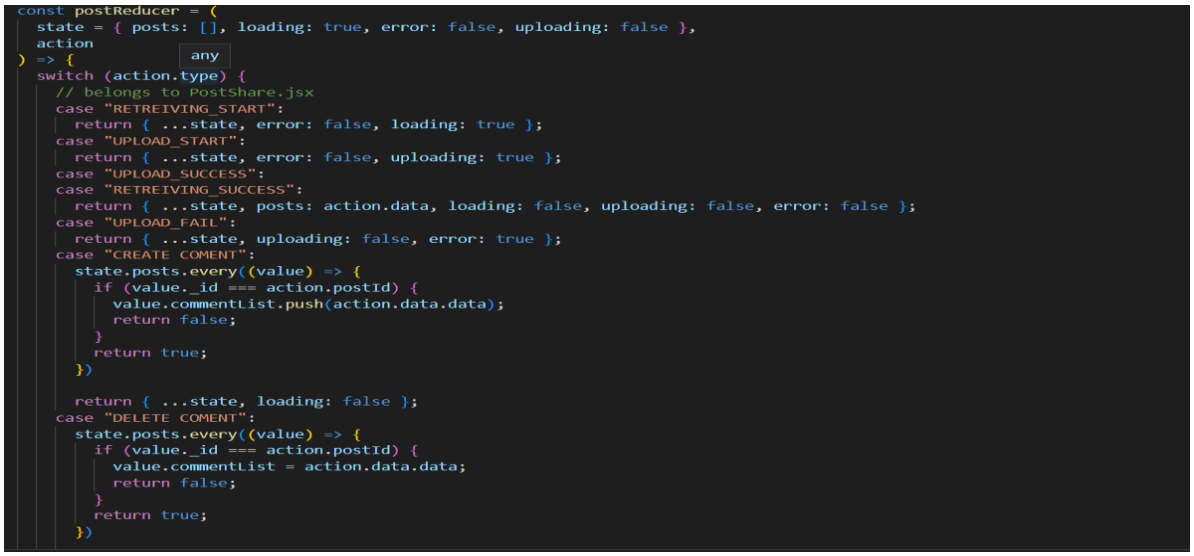
Na slikama 31 vidimo redukciju koja je će biti povezana u cilju komunikacije s reduktorima slanjem određene vrste radnji, u ovom slučaju tu se radi o autentifikaciji i korisnici koji prate jedan drugoga.

```
const authReducer = (state = { authData : null, loading: false , error: false}, action) =>
    {
        switch(action.type){
            case "AUTH_START":
                return{...state, loading: true, error: false}
            case "AUTH_SUCCESS":
                localStorage.setItem("profile", JSON.stringify({...action?.data}));
                return{...state, authData: action.data, loading:false , error: false}
            case "AUTH_FAIL":
                return {...state, loading: false, error: true}

            case "UPDATING_START":
                return {...state, updateLoading: true , error: false}

            case "UPDATING_SUCCESS":
                localStorage.setItem('profile', JSON.stringify({...action?.data}))
                    return {...state, authData: action.data, updateLoading: false, error: false}


            case "UPDATING_FAIL":
                    return {...state, updateLoading: true, error: true}


            case "FOLLOW_USER":
            return {...state, authData: {...state.authData, user: {...state.authData.user, following: [...state.authData.user.following,

           case "UNFOLLOW_USER":
      return {...state, authData: {...state.authData, user: {...state.authData.user, following: [...state.authData.user.following.filter


            case "LOG_OUT":
                localStorage.clear()
    return {...state,authData: null, loading: false, error: false}
              default:
                  return state
        }
    }
```

Slika 31. AuthenticationReducer

4.4.5.2PostReducer.js

Radnja sa slike 32 u osnovi govori koje vrste reduktora su pozvane od radnji i unutar korisnika koja je vrsta akcije pozvana. U globalu se koristi naredba switch koja ovisi o tipu akcije i unutar njih rade se provjere.



Slika 32. PostReducer

4.4.4 Store

Za kreiranje stora uvoze se važne biblioteke koje su prikazane na slici 33. Za izradu se mora primijeniti sami redux-thunk middleware, dakle ovdje se uvozi sam redux thunk middleware za asinkronu izvedbu i ovdje se sve reducira. Ovdje se nalazi funkcija za spremanje samih statova u lokalnu pohranu koja serijalizira sve klijentove podatke dostupne u pohrani dok se oni prosljeđuju u parametre i dajemo ih svim lokalnim pohranama. Druga funkcija kao što samo ime kaže učitava i dohvaća podatke iz lokalne pohrane.

```
import {
  legacy_createStore as createStore,
  applyMiddleware,
  compose,
} from "redux";

import thunk from "redux-thunk";
import { reducers } from "../reducers";

function saveToLocalStorage(store) {
  try {
      const serializedStore = JSON.stringify(store);
      window.localStorage.setItem('store', serializedStore);
  } catch(e) {
      console.log(e);
  }
}

function loadFromLocalStorage() {
  try {
      const serializedStore = window.localStorage.getItem('store');
      if(serializedStore === null) return undefined;
      return JSON.parse(serializedStore);
  } catch(e) {
      console.log(e);
      return undefined;
  }
}
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
const persistedState = loadFromLocalStorage();

const store = createStore(reducers, persistedState, composeEnhancers(applyMiddleware(thunk)));

store.subscribe(() => saveToLocalStorage(store.getState()));

export default store;
```

Slika 33. Store

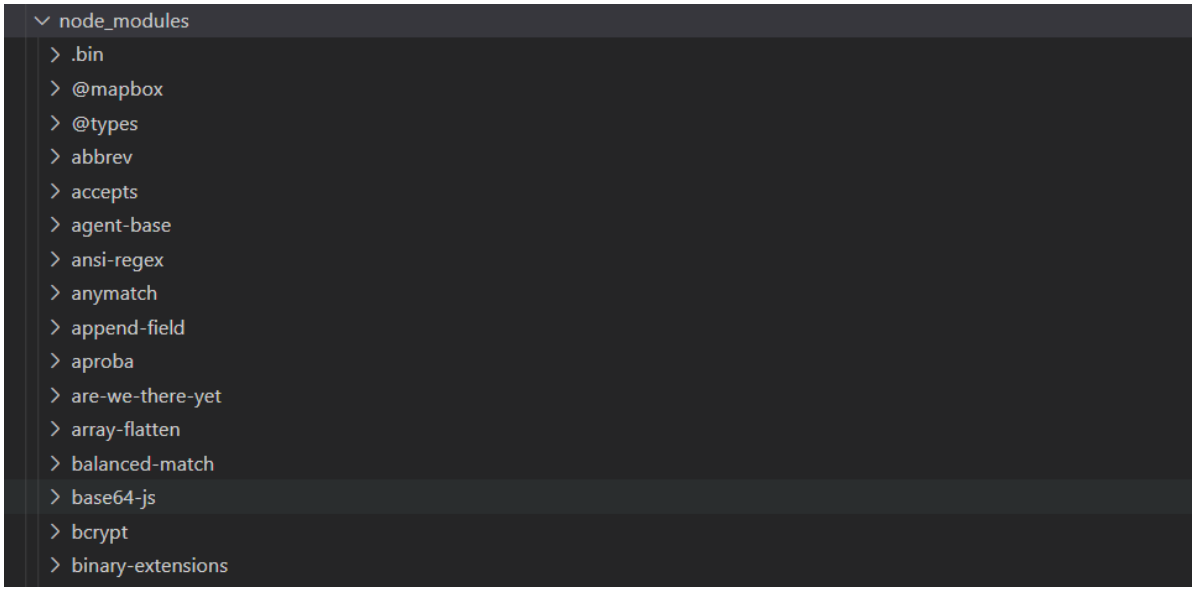
5. STRANA POSLUŽITELJA

Slično kao i na strani klijenta, 'strana poslužitelja' znači sve što se događa na poslužitelju, umjesto na klijentu. U prošlosti se gotovo sva poslovna logika izvodila na strani poslužitelja, a to je uključivalo renderiranje dinamičkih web stranica, interakciju s bazama podataka, autentifikaciju identiteta i push obavijesti. Problem s hostingom svih ovih procesa na strani poslužitelja je taj što svaki zahtjev koji uključuje jedan od njih mora putovati cijelim putem od klijenta do poslužitelja, svaki put. Ovo uvodi veliku latenciju što znači vrijeme čekanja koje protekne od trenutka zahtijeva za podacima do trenutka dok se ti podaci ne pojave. Iz tog razloga, suvremene aplikacije pokreću više koda na strani klijenta. Jedan je od slučajeva upotrebe renderiranje dinamičkih web stranica u stvarnom vremenu pokretanjem skripti unutar preglednika koje mijenjaju sadržaj, a koji korisnik vidi. Kao i kod 'frontenda' i 'backend' također izraza za procese koji se odvijaju na poslužitelju, iako se backend odnosi samo na vrste procesa, a server-side se odnosi na lokaciju na kojoj se procesi izvode. Na slici 34 može se vidjeti struktura server mape koja izgleda ovako:

Slika 34. Poslužitelj mapa

5.1 Node_modules

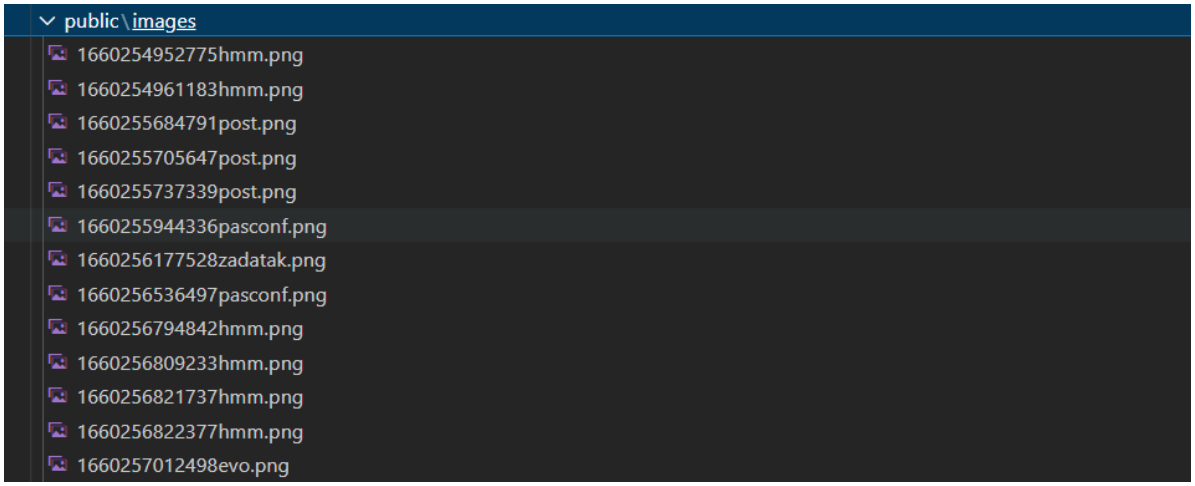
Datoteka Node_modules se može zamisliti kao pred memoriju za vanjske module o kojima ovisi projekt. Kada ih npm instalira, oni se preuzimaju s weba i kopiraju u mapu node_modules, a Node.js je osposobljen da ih tamo traži kada ih se uveze (bez određenog puta). Nazvati se može pred memorijom jer se mapa node_modules može potpuno ponovno stvoriti od nule u bilo kojem trenutku samo ponovnim instaliranjem svih zavisnih modula. Na slici 35 prikazana je datoteka node_modules i prikazan je samo dio njenih paketa.



Slika 35. Node_modules datoteka

5.2 Public datoteka

Ova datoteka slike 36 sadrži sve slike koje je korisnik podijelio na svome profilu pri kreiranju vlastitog posta, a na nju možemo gledati kao spremište slika .



Slika 36. Public datoteka

5.3 RMMC datoteka

RMMC datoteka se sastoji od više datoteka, a to su Router, Middleware, Modules i Controller mapa. Na slici 37 prikazane su njene datoteke.

Slika 37. RMMC datoteka

5.3.1 Routes

Routes je proces u kojem se korisnik usmjerava na različite stranice na temelju svoje radnje ili zahtjeva. ReactJS Router uglavnom se koristi za razvoj jednostranih web aplikacija. React Router koristi se za definiranje više ruta u aplikaciji. Kada korisnik unese određeni URL u preglednik i ako se ovaj URL put podudara s bilo kojom 'rutom' unutar datoteke usmjerivača, korisnik će biti preusmjeren na tu određenu rutu. React Router je standardni bibliotečni sustav izgrađen na vrhu Reacta i koristi se za kreiranje usmjeravanja u React aplikaciji koristeći React Router Package. Omogućuje sinkroni URL u pregledniku s podacima koji će biti prikazani na web stranici. Održava standardnu strukturu i ponašanje aplikacije i uglavnom se koristi za razvoj web aplikacija na jednoj stranici. Router datoteka sadrži sljedeće datoteke:

- **AuthenticationRoute.js** (registerUserm , loginUser)
- **ChatRoute.js** (userChats , createChat, findChat)
- **MessageRoute.js** (addMessage, getMessages)
- **PostRequest.js**(getTimelinePosts,createComment,deleteComment, createPost, getPost, deletePost)
- **UploadRequest.js** (uploadImage, uploadPost)
- **UserRequest** (getUser, getAllUser, followUser, unFollowUser)

Svaka od ovih datoteka sadrži svoj određeni put do nečega i povezana je s drugim datotekama u samom Controllers folderu .

5.3.1.1 AuthenticationRoute.js

Slika 38 prikazuje usmjeravanje AuthenticationRouta prema registraciji i prijavi.

```
import express from "express";
import { loginUser, registerUser } from "../Controllers/AuthController.js";

const router = express.Router()


router.post('/register', registerUser)
router.post('/login', loginUser)
export default router
```

Slika 38. AuthenticationRoute

5.3.1.2 ChatRoute.js

Slika 39 prikazuje usmjeravanje ChatRouta prema kreiranju razgovora, korisničkog razgovora i pronalasku razgovora.

```
import express from 'express'
import { createChat, findChat, userChats } from '../Controllers/ChatController.js'

const router = express.Router()

router.post("/", createChat)
router.get("/:userId", userChats)
router.get("/find/:firstId/:secondId", findChat)
export default router
```

Slika 39. ChatRoute

5.3.1.3 MessageRoute.js

Slika 40 prikazuje usmjeravanja MessageRouta prema dodavanju poruka i dohvaćanju poruka.

```
import express  from "express";
import { addMessage, getMessages } from "../Controllers/MessageController.js";
const router = express.Router()

router.post('/', addMessage)
router.get('/:chatId', getMessages)

export default router
```

Slika 40. MessageRoute

5.3.1.4 PostRoute.js

PostRoute sadrži dosta ruta jer je sam po sebi specifičan, a to su kreiranje posta dobivanje posta, brisanja posta, kreiranje komentara i brisanje komentara, a to se sve vidi na slici 41.

```
import express from "express";
import {
  createPost,
  deletePost,
  getPost,
  getTimelinePosts,
  updatePost,
  createComment,
  deleteComment,
} from "../Controllers/PostController.js";
const router = express.Router();

router.post("/", createPost);
router.get("/:id", getPost);
router.put("/:id", updatePost);
router.delete("/:id", deletePost);
router.get("/:id/timeline", getTimelinePosts);
router.post("/:id/:userId/comment", createComment);
router.delete("/:id/:userId", deleteComment);

export default router;
```

Slika 41. PostRoute

5.3.1.5 UploadRoute.js

UploadRoute sadrži rutu za spremanje slika stavljenih na korisničku objavu, a sprema ju u public datoteku na poslužiteljskoj strani što se vidi i na slici 42.

```
import express from 'express'
const router = express.Router()
import multer from 'multer'

const storage = multer.diskStorage({
    destination: (req, file, cb) => {
      cb(null, "public/images");
    },
    filename: (req, file, cb) => {
      cb(null, req.body.name);
    },
  });
const upload = multer({ storage: storage });


router.post("/", upload.single("file"), (req, res) => {
    try {
      return res.status(200).json("File uploded successfully");
    } catch (error) {
      console.error(error);
    }
  });

export default router
```

Slika 42 UploadtRoute

5.3.1.6 UserRoute.js

User route sadrži rutu za dobivanje svih korisnika , dobivanje jednog korisnika i za pratit/otpratit što se vidi na slici 43.

```
import express from "express";

import {  followUser, getUser,getAllUsers, unfollowUser} from "../Controllers/UserController.js";
import authMiddleWare from "../MiddleWare/authMiddleWare.js";

const router = express.Router();

router.get('/', getAllUsers)
router.get('/:id', getUser)
router.put('/:id/follow', authMiddleWare, followUser)
router.put('/:id/unfollow',authMiddleWare,  unfollowUser)

export default router;
```

Slika 43 UserRoute

5.3.2 Models

Models datoteka i sve što je u njemu su javascript scheme koje se koriste u Controller datotekama za izvođenje određene vrste funkcionalnosti. Mongoose Schema je bubnjar u bendu — ona definira strukturu dokumenta, zadane vrijednosti, njegove validatore i druge stvari. Shema čini korištenje podataka upotrebljivijim. Shema se primjenjuje na model, a mongoose to primjenjuje na MongoDB bazu podataka.

5.3.2.1 ChatModel.js

Chat model je jako mal zato što on kao svoj parametar uzima članove samog razgovora i to je dosta što je vidljivo na slici 44.

```
import mongoose from 'mongoose'

const ChatSchema = mongoose.Schema({
    members: {
        type: Array,
    },
},
{
    timestamps: true,

}
);

const ChatModel = mongoose.model("Chat", ChatSchema);
export default ChatModel
```

Slika 44 ChatModel

5.3.2.2UserModel.js

Schema samog korisnika koja je prikazana na slici 45 sadrži sve parametre koje bi jedan korisnik trebao sadržavati , a to su korisničko ime, ime, prezime, lozinka, profilna slika, pozadinska slika, gdje živi, gdje radi, država i pratitelji . Svi su ovi podaci bitni jer će svaki korisnik imati svoje jedinstvene podatke za prijavu, slike i korisničke informacije.

```
import mongoose from "mongoose";

const UserSchema = mongoose.Schema(
    {
        username: {
            type: String,
            required: true
        },
        password : {
            type: String,
            required: true
        },
        firstname: {
            type: String,
            required: true
        },
        lastname : {
            type: String,
            required: true
        },
        isAdmin : {
            type: Boolean,
            default: false,
        },
        profilePicture: String,
        coverPicture: String,
        about: String,
        livesin: String,
        workAt: String,
        relationship: String,
        country: String,
        followers: [] ,
        following: []
    },
    {timestamps: true}
)

const UserModel= mongoose.model("Users", UserSchema);
export default UserModel
```

Slika 45. UserModel

5.3.2.3 PostModel.js

Sami post model u svojoj schemi sadrži podatke koji će sačinjavati jednu objavu od samog korisnika zato je potreban userId i ostali podaci slični kao u UserModelu jer nam trebaju informacije o tome tko je nešto objavio i što je objavio, a prikaz samog koda vidi se na slici 46.

```
import mongoose from "mongoose";

const postSchema = mongoose.Schema(
  {
    userId: { type: String, required: true },

    firstname: String,
    lastname: String,
    commentList: [],
    checked: Boolean,
    desc: String,
    likes: [],
    image: String,
  },

  {
    timestamps: true,
  }
);

var PostModel = mongoose.model("posts", postSchema);

export default PostModel;
```

Slika 46. PostModel

5.3.2.4 MessageModel.js

MessageModel sa slike 47 ima definirane podatke za slanje poruke jednog korisnika te kako bi se on mogao odvijati potrebno je imati chatId što znači sami id od razgovora, id od pošiljatelja i text poruke koju pošiljatelj šalje .

```
import mongoose from "mongoose";


const MessageSchema = new mongoose.Schema({
    chatId: {
        type: String
    },
    senderId: {
        type: String
    },
    text: {
        type: String
    },
},
{
    timestamps: true,
}
)

const MessageModel = mongoose.model("message", MessageSchema)
export default MessageModel
```

Slika 47. MessageModel

5.3.3 MiddleWare

Implementiranje middleware-a za provjeru autentičnosti korisničkog JWT-a [7]. Unutar authMiddleWare slika 48, importiran je sami jwt iz „jsonwebtokena“-a. Pri korištenju obrađenih varijabli okoline potrebno je bilo dohvatiti vlastiti jwt ključ, jer se nakon toga pravi osobni middleware za provjeru autentičnosti. Prije svega ovdje se uzeo token iz korisničkog zaglavlja i gleda se da se vidi što je token, ako je token prisutan onda ga se korigira iz vlastitog json web tokena koji je već napravio algoritam dok se generirao, tako da u osnovi ovaj algoritam ovisi o korisničkom tajnom ključu.

```
import jwt from "jsonwebtoken";
import dotenv from "dotenv";

dotenv.config();
const secret = process.env.JWT_KEY;
const authMiddleWare = async (req, res, next) => {
  try {
    const token = req.headers.authorization.split(" ")[1];
    console.log(token)
    if (token) {
      const decoded = jwt.verify(token, secret);
      console.log(decoded)
      req.body._id = decoded?.id;
    }
    next();
  } catch (error) {
    console.log(error);
  }
};

export default authMiddleWare;
```

Slika 48. authMiddleware

5.3.4 Controllers

Controllers-a nema bez ruta. Rute su u osnovi putevi do određenog controlera . Ovo nije ništa drugo nego samo put za postizanje određene vrste funkcionalnosti ugrađene u controler. Controlers nisu ništa drugo nego javascript funkcija koja koristi scheme koje se nalaze u mapi models (chat,message,post,user) za specifične vrste funkcionalnosti

- **AuthenticationController.js** (registerUser , loginUser)
- **ChatController.js** (userChats , createChat, findChat)
- **MessageController.js** (addMessage, getMessages)
- **PostController.js**(getTimelinePosts,likePosts,createComment, deleteComment, createPost, getPost, deletePost)
- **UserRequest**(getUser, updateUser, getAllUser, followUser, unFollowUser)

5.3.4.1AuthenticationController.js

Authentication controller ima ulogu kontrolirati kao što vidimo sa slike 49, novog korisnika i prijavu postojećeg korisnika u sustav, uz prijavu i registraciju program javlja i pogreške ako se lozinka ne podudara ili ako su uneseni podaci već iskorišteni za registraciju nekog korisnika.

```
// Registering a new User
export const registerUser = async (req, res) => {
    try {
     const oldUser = await UserModel.findOne({username})

     if(oldUser){
       return res.status(400).json({message : "Username is already registered!"})
     }
     const user = await newUser.save();

     const token = jwt.sign({
       username: user.username, id: user._id
     },
      process.env.JWT_KEY, {expiresIn: '1h'})


     res.status(200).json({user, token});
   } catch (error) {
     res.status(500).json({ message: error.message });
   }
};


// login User
export const loginUser = async (req, res) => {
  const { username, password } = req.body;

  try {
    const user = await UserModel.findOne({ username: username });

    if (user) {
      const validity = await bcrypt.compare(password, user.password);

      if (!validity) {
        res.status(400).json("wrong password");
      } else {
        const token = jwt.sign(
          { username: user.username, id: user._id },
```

Slika 49. AuthenticationController

5.3.4.2 PostController.js

Na slici 50 prikazane su funkcije koje sadrži postcontroller, a to su kreiranje novog posta i dohvaćanje posta koji kao što se iz teksta prije vidi sadrži „PostSchemu“ koja je definirana.

```
export const createPost = async (req, res) => {
  const newPost = new PostModel(req.body);

  try {
    await newPost.save();
    res.status(200).json(newPost);
  } catch (error) {
    res.status(500).json(error);
  }
};
export const getPost = async (req, res) => {
  const id = req.params.id;

  try {
    const post = await PostModel.findById(id);
    res.status(200).json(post);
  } catch (error) {
    res.status(500).json(error);
  }
};
```

Slika 50. PostController

5.3.4.3UserController

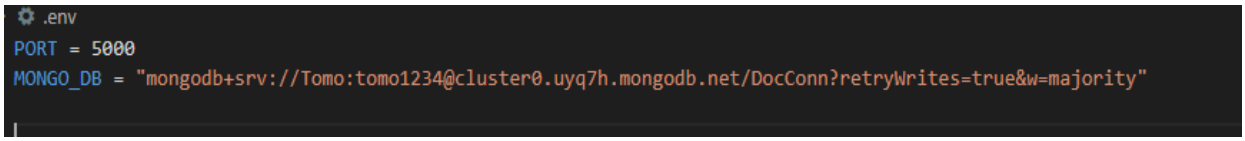
UserController sadrži zahtjeve koji su uobičajni za korisnika, a to znači dobivanje svih korisnika iz baze podataka i dobivanje samo jednog korisnika. Ovi su zahtjevi jako važni jer se radi o korisnicima o kojima ovisi cijela aplikacija. Na slici 51 vidimo korištenje UserModel-a i njegove scheme koja je kreirana .

```
export const getAllUser = async (req, res) => {
  try {
    let users = await UserModel.find();
    users = users.map((user) => {
      const { password, ...otherDetails } = user._doc;
      return otherDetails;
    });
    res.status(200).json(users);
  } catch (error) {
    res.status(500).json(error);
  }
};

export const getUser = async (req, res) => {
  const id = req.params.id;

  try {
    const user = await UserModel.findById(id);
    if (user) {
      const { password, ...otherDetails } = user._doc;

      res.status(200).json(otherDetails);
    } else {
      res.status(404).json("No such User");
    }
  } catch (error) {
    res.status(500).json(error);
  }
};
```

Slika 51.UserController

5.4 ENV

Env datoteka služi za skrivanje nekakvih podataka kao što je u ovome slučaju broj PORT-a i URL-a koji služi sa povezivanje sa MongoDB-om, a sadrži korisničko ime i lozinku kao što se vidi na slici 52.



Slika 52 env datoteka

Izvor : samostalna izrada , 2022

6. SOCKET.IO STRANA

Socket.IO je biblioteka koja omogućuje nisku latenciju, dvosmjernu komunikaciju koja se temelji na događajima između klijenta i poslužitelja. Izgrađen je na temelju WebSocket protokola i pruža dodatna jamstva kao što je povratak na HTTP dugo prozivanje ili automatsko ponovno povezivanje. Ključne razlike između WebSocketa i socket.io Omogućuje vezu preko TCP-a, dok je Socket.io biblioteka za apstrahiranje WebSocket veza. WebSocket nema rezervne opcije, dok Socket.io podržava rezervne opcije. WebSocket je tehnologija, dok je Socket.io biblioteka za WebSockets [8]. Kreiranje Chata na ovom projektu počinje sa Server strane kreiranjem ChatModel.js-a gdje se kao i za sve radi schema i povezuje se sa mongoDB-om te se nakon toga postavljaju parametri koje sadrži dijelove sheme za chat. Shema za te parametre uzima samo evidenciju članova(members) unutar chata što bi značilo ID korisnika i timestamps. Nakon ovoga napravljen je još jedna model koji sadržava druge parametre u schemi, a to su chatID(jedinstveni broj chata), senderID(jedinstveni broj pošiljatelja), text poruke i timestaps. Nakon server strane prelazak na client stranu gdje se odrađuje izgled same Message stranice. Slaganje JSX i CSS komponenta te primanjem funkcionalnosti sa server strane .

6.1 ChatModel

Kao što se vidi na slici 53 chatmodel schema sastoji se od membersa koji su type array i timestamps koji je uvijek true.

```
import mongoose from 'mongoose'

const ChatSchema = mongoose.Schema({
    members: {
        type: Array,
    },
},
{
    timestamps: true,

}
);

const ChatModel = mongoose.model("Chat", ChatSchema);
export default ChatModel
```

Slika 53. ChatModel.js

6.2 MesssageModel

Message mode sadrži schemu koja u sebi nosi id samog razgovora, id pošiljatelja i tekst koje vidimo na slici 54.

```
import mongoose from "mongoose";


const MessageSchema = new mongoose.Schema({
    chatId: {
        type: String
    },
    senderId: {
        type: String
    },
    text: {
        type: String
    },
},
{
    timestamps: true,
}
)

const MessageModel = mongoose.model("message", MessageSchema)
export default MessageModel
```

Slika 54. MessageModel.js

7. ZAKLJUČAK

Projektni zadatak koji je zapravo i diplomski rad obrađivao je temu iz vrlo kompleksne grane današnjeg svijeta, a to je medicina. Zbog važnosti poboljšanja komunikacijskih kanala između samih liječnika osmišljena je aplikacija DocConnect . Izrađenu aplikaciju korisnik koristi u svrhu povezivanja s ostalim liječnicima koji su registrirani na toj platformi. Mogućnosti su razne kao na primjer razmjena iskustava putem poruka, dogovaranje konferencija i pomaganje u hitnim slučajevima što je i glavni cilj. Pomoći nekome prikupljanjem informacija i mišljenja od drugih liječnika i tako povećavajući šansu za boljom dijagnozom što rezultira manjom pogreškom liječnika, a što za pacijenta znači brže otkrivanje bolest tj. brži početak odgovarajućeg liječenja. Prednosti ove aplikacije su te što je u današnje vrijeme sve digitalizirano i potrebne informacije liječnicima su na dohvat ruke. Sami liječnici ne moraju više slati ljude po druga mišljenja što i samim ljudima stvara dodatni stres i nervozu nego ovako preko DocConnect aplikacije liječnik to može obaviti za njih. Bitna stvar ovdje je stjecanje novih poznanstava među ljudima, komunikacija sa ljudima iz istog interesnog kruga. Nedostaci ove aplikacije su ti što je još dosta liječnika u starijoj dobi i izbjegava koristiti nove tehnologije koje im se nude, nego su vjerni sebi i svojoj intuiciji. Još jedan nedostatak koji može biti je loša informatička pismenost, ali i to se sve može proći kroz nekakve edukacije. Što se tiče mogućnosti za poboljšanje ove aplikacije kao i kod svega ostalog uvijek ima mjesta za napredak. Preciznije u ovom slučaju aplikaciju se može proširiti tako da ne budu samo za hrvatske liječnike nego da se njome mogu koristit i umrežavati svi liječnici u svijetu. To je izvedivo ubacivanjem raznih filtera među podatke i dodavanjem novih funkcionalnosti koje bi grupirale same liječnike i tako bi se otklonila jezična barijera koja bi postojala. Još jedna od mogućnosti unapređenja aplikacije je kreiranje njezine inačice za mobilne uređaje. S prelaskom na novu platformu liječnici ne bi bili ovisni o računalu nego bi mogli preko mobilnih uređaja koristiti samu aplikaciju i obavljati sve funkcionalnosti koje aplikacija pruža liječnicima. Jedina mana kod korištenja mobilnih uređaja opet bi bila informatička pismenost među samim liječnicima kojima bi trebalo više vremena za svladati samo korištenje ove aplikacije.

8.LITERATURA

[1] Express web framework (Node.js/JavaScript), Mozilla and individual contributors, 2021, dostupno na: https://developer.mozilla.org/en-US/docs/Learn/Serverside/Express_Nodejs, [pristupljeno: 10.08.2022]

[2] Chodorow Kristina (2013), MongoDB: The Definitive Guide, dostupno na: https://usuaris.tinet.cat/bertolin/pdfs/mongodb_%20the%20definitive%20guide%20-%20kristina%20chodorow_1401.pdf, [pristupljeno: 12.8.2022]

[3] Greg Lim (2019) ,Beginning Node.js, Express & MongoDB Development, dostupno na: https://www.scribd.com/book/547643514/Beginning-Node-js-Express-MongoDB-Development , [pristupljeno 3.8.2021]

[4] Mark Tielens Thomas (2018), React in Action, dostupno na: https://itbook.store/files/9781617293856/chapter1.pdf, [pristupljeno 11.8.2022]

[5] Mehmed Smajić (2011) Svaka sedma liječnička dijagnoza – pogrešna, dostupno na : https://www.dw.com/hr/svaka-sedma-lije%C4%8Dni%C4%8Dka-dijagnoza-pogre%C5%A1na/a-15132809, [pristupljeno:1.8.2022]

[6] Peyrott Sebastian (2016) Jwt Handbook, dostupno na: https://assets.ctfassets.net/2ntc334xpx65/o5J4X472PQUI4ai6cAcqg/13a2611de03b2c8edbd09c3ca14ae86b/jwt-handbook-v0_14_1.pdf, [pristupljeno 20.8.2022]

[7]Rohit Rai (2013), Socket.IO Real-time Web Application Development, dostupno na: https://khoapham.vn/KhoaPhamTraining/nodejs/snippet/Socket-IO-Realtime-Web-App-Development.pdf, [pristupljeno 15.8.2022]

[8] Hoque Shama (2020), Full-Stack React Projects Second Edition, dostupno na: https://sd.blackball.lv/library/Full-Stack_React_Projects_2nd_Edition_(2020).pdf, [pristupljeno 10.8.2022]

[9] Soham De Roy (2018) What is Redux? Store, Actions, and Reducers, dostupno na :https://www.freecodecamp.org/news/what-is-redux-store-actions-reducers-explained/, [pristupljeno 12.8.2022]

9. POPIS SLIKA

SAŽETAK

Ovaj diplomski rad sastoji se od teorijskog i praktičnog dijela . U teorijskom dijelu pomno su objašnjeni alati koji su služili u izradi ovoga projekta , a nakon toga uz detaljan dizajn prikazan je prototip same aplikacije. Nakon toga slijedio je dio u kojem se svaka komponenta razlučila na zasebne dijelove te se objasnila njezina uloga u samoj aplikaciji. Suština ovog rada bila je prikazati slikovno i pisano kako teče proces kreiranja aplikacije od prototipa pa sve do kraja i koja su razvojna okruženja koja u tome mogu pomoću, a s druge strane pokazati znanje stečeno kroz sve godine školovanja i pretočiti ga u nešto društveno korisno što može nekome pomoći u stvarnom životu.

KLJUČNE RIJEČI: React, MongoDb, Express, Node , Pojekt, Action, Components, Reducer , Api , Models , Router, Index , App, Api, Redux. Socket.IO;

ABSTRACT

This Master's thesis consists of a theoretical and a practical part. In the theoretical part, the tools that were used in the creation of this project are explained in detail, and after that, the prototype of the application itself is shown along with the detailed design. This was followed by a section in which each component was separated into separate parts and its role in the application itself was explained. The essence of this work was shown graphically and in writing, how the process of creating applications from the prototype to the end runs and what are the development environments that can help in this, and on the other hand, to show the knowledge acquired through all the years of schooling and turn it into something socially useful that can help someone in real life.

KEY WORDS: React, MongoDb, Express, Node , Project, Action, Components, Reducer , Api , Models , Router, Index , App, Api, Redux, Socket.IO;