MessageMe-web stranica za komunikaciju putem poruka

Maričić, Emanuel

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

Emanuel Maričić

MessageMe – web stranica za komunikaciju putem poruka

ZAVRŠNI RAD

Osijek,2023. godina

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

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 27.04.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime Pristupnika:	Emanuel Maričić
Studij, smjer:	Preddiplomski stručni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR 4735, 19.07.2019.
OIB Pristupnika:	80870177609
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Miljenko Švarcmajer, mag. ing. comp.
Naslov završnog rada:	MessageMe-web stranica za komunikaciju putem poruka
Znanstvena grana završnog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rada	Stranica omogućuje registriraciju i prijavu korisnika, te će svaki korisnik imati svoj profil u kojem će biti prikazane njegove informacije. Također će moći pretraživati profile drugih korisnika i komunicirati s njima. Glavna funkcionalnost će biti razmjenjivanje poruka u pravom vremenu (eng. Real-time), i biti će napravljena u ReactJS okruženju. Rezervirano za: Emanuel Maričić
Prijedlog ocjene pismenog dijela ispita (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: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	27.04.2023.

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, 16.05.2023.

Ime i prezime studenta:	Emanuel Maričić
Studij:	Preddiplomski stručni studij Računarstvo
Mat. br. studenta, godina upisa:	AR 4735, 19.07.2019.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: MessageMe-web stranica za komunikaciju putem poruka**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

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

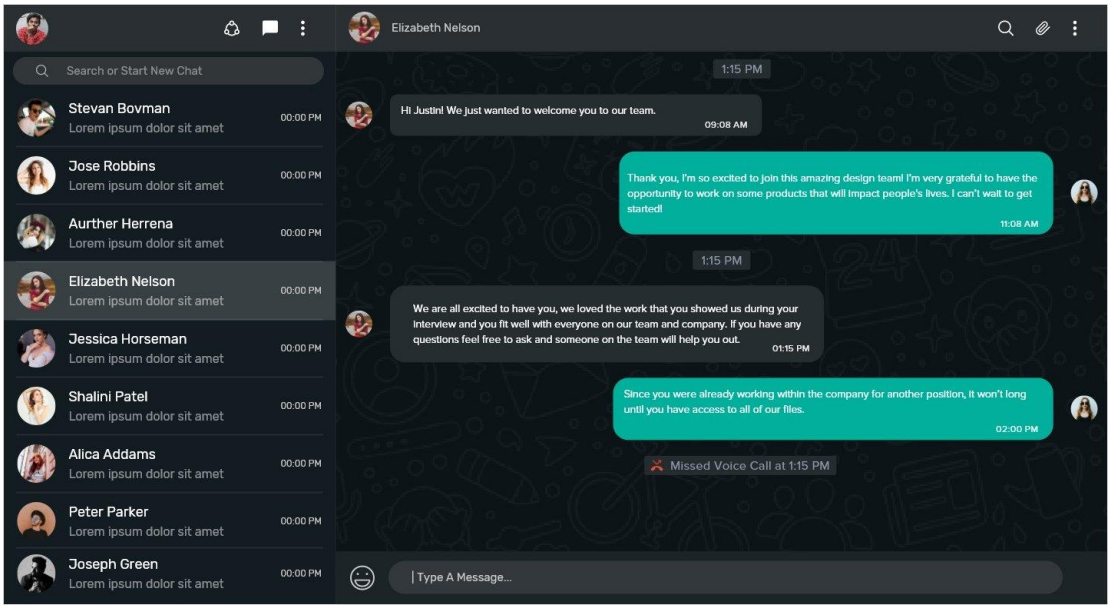
Aplikacije za razmjenu poruka postale su bitan dio naše svakodnevne komunikacije. S porastom tehnologije i pametnih telefona, aplikacije za razmjenu poruka postale su sastavni alat za ljude da ostanu povezani sa svojom obitelji, prijateljima i kolegama te čak da steknu nova poznanstva. Od osnovnih tekstualnih poruka do multimedijskih poruka, aplikacije za razmjenu poruka značajno su se razvile tijekom godina i nastavljaju nuditi nove značajke i funkcije za poboljšanje korisničkog iskustva. Posljednjih godina bilježe eksplozivan rast i sada ih koriste milijarde ljudi diljem svijeta. To je rezultiralo oštrim natjecanjem među programerima aplikacija za razmjenu poruka u stvaranju najbolje moguće aplikacije. Konkurencija je dovela do razvoja novih značajki kao što su videopozivi, glasovni pozivi. Međutim, uz sve veći broj dostupnih aplikacija za razmjenu poruka, korisnici se suočavaju s izazovom odabira prave aplikacije za razmjenu poruka koja odgovara njihovim potrebama. Stoga je bitno razumjeti različite dostupne aplikacije za razmjenu poruka te njihove značajke i funkcionalnosti. U ovom je radu cilj pružiti korisniku sučelje za razmjenu poruka.

1.1 Zadatak rada

Stranica omogućuje registriraciju i prijavu korisnika, te će svaki korisnik imati svoj profil u kojem će biti prikazane njegove informacije. Također će moći pretraživati profile drugih korisnika i komunicirati s njima. Glavna funkcionalnost će biti razmjenjivanje poruka u pravom vremenu (eng. Real-time), i biti će napravljena u ReactJS okruženju.

2. PREGLED PODRUČJA TEME

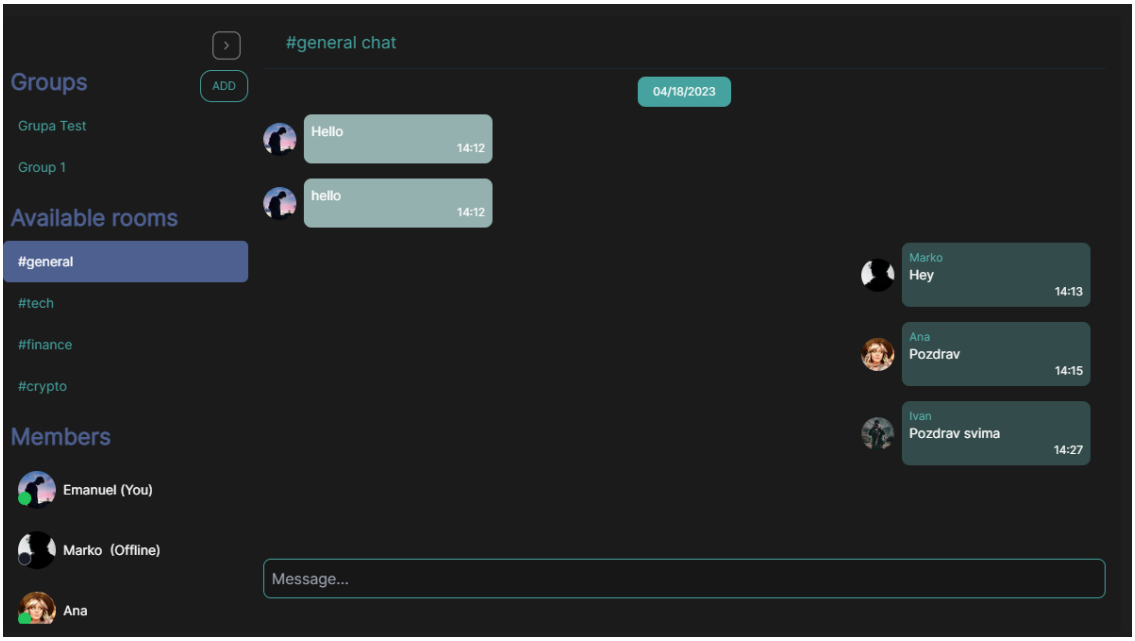
Zadaća ovog završnog rada je izraditi web aplikaciju za razmjenu poruka kao što su WhatsApp i Facebook Messenger koji su jedne od najkorištenijih aplikacija za razmjenu poruka u svijetu. One sadržavaju razmjenu poruka u pravom vremenu, te mnoge druge funkcionalnosti kao kreiranje grupa, slanje fotografija i videozapisa. MessageMe aplikacija će sadržavati korisničko sučelje i jednostavan navigacijski sustav. Aplikacija će nuditi pravovremenu komunikaciju između registriranih korisnika, vidljivost korisnika koji su trenutno na mreži te će svi korisnici imati mogućnost razmjena poruka u generalnim grupama. Aplikacija će biti dizajnirana s end-to-end enkripcijom, koja će osigurati da su poruke sigurne i da ih treće strane ne mogu presresti. Jedna od jedinstvenih značajki vaše aplikacije za razmjenu poruka biti će direktna komunikacija s ChatGPT robotom (napredni robot temeljen na umjetnoj inteligenciji koji je treniran da razgovara s ljudima). Sama aplikacija se sastoji od pet ruta a to su: *home*, *chat, chatify*, *login* i *register*. Kroz sve rute aplikacije proteže se zaglavlje i podnožna traka. Pomoću zaglavlja se navigira kroz samu aplikaciju, te ima logo koji označuje stranicu, dok se podnožje aplikacije sastoji od informacija o kreatoru projekta, te resursima koji su korišteni. Početna stranica je zamišljena kao prikaz funkcionalnosti aplikacije samom korisniku. Sastoji se od teksta koji opisuje korisniku što se nalazi u aplikaciji, *Get Started* tipke koja navigira na prijavu ili registraciju i fotografije koja prikazuje poruke između osoba.



Slika 2.1 Primjer prikaza WhatsApp korisničkog sučelja

Zatim postoji *chat* ruta koja sadrži lijevu bočnu traku i prozorčić u kojemu se nalaze poruke.

U lijevoj bočnoj traci prikazane su grupe u kojima je korisnik, generalne sobe kojima svi korisnici mogu pristupiti, te sami korisnici aplikacije sa svojim statusom (prijavljen, neprijavljen). U prozorčiću gdje se nalaze poruke, svaka poruka ima sliku korisnika koji je napisao poruku, sami tekst i vrijeme kada je napisao poruku. Poruke su sortirane prema datumima i dizajn je napravljen tako da korisnik može razlikovati svoje poruke od tuđih.



Slika 2.2 Prikaz sučelja razgovora

Chatify ruta omogućuje razgovor sa umjetnom inteligencijom, kojoj se može postaviti bilo koje pitanje te će ona na to imati odgovor. Sučelje se sastoji od prozorčića u kojemu su prikazane poruke između korisnika i umjetne inteligencije. Zatim aplikacija MessageMe još sadržava prijavu i registraciju koja omogućuje da korisnik pristupi aplikaciji, sučelje se može vidjeti na slici 3.8. Cilj same aplikacije je pružiti sigurnu i brzu komunikaciju između ljudi, tako da imaju mogućnost steći nova poznanstva.

3. TEORIJSKA PODLOGA I DETALJI O PROJEKTU

Sama aplikacija se dijeli na dva dijela, to su client i server dio. Klijent dio pokriva korisničko sučelje (prednja strana, engl. frontend). To je izvršni dio koji korisnici vide i sa kojima interagiraju. Server dio se odnosi na dio web aplikacije ili softverskog sustava koji korisnici ne vide (pozadinska strana ili backend), koristi se za obradu i pohranu podataka te za pružanje funkcionalnosti korisnicima preko prednje strane (engl. frontend-a). Pozadinska strana (engl. Backend) uključuje sve dijelove aplikacije koji rade na serveru, uključujući bazu podataka, sigurnosne provjere, upravljanje sesijama i autorizaciju, te komunikaciju s drugim aplikacijama ili servisima.

3.1. Client

Za izradu samog client dijela će se koristiti NextJs kao primarnu tehnologiju za izradu web-stranice. Next.js je framework otvorenog izvora (engl. open source) za izgradnju React aplikacija, posebno za izgradnju server strane prikaza (engl. server-side rendered (SSR)) i generiranje statičnih stranica (engl. static site generated (SSG)) web stranica. Next.js omogućava razvoj bržih i skalabilnijih React aplikacija tako što uključuje mnoge karakteristike koje olakšavaju razvoj i održavanje aplikacija. Jedna od najvećih prednosti Next.js-a je mogućnost generiranja statičkih HTML stranica koje se mogu izravno dostaviti korisniku, što može dovesti do znatnog ubrzanja učitavanja stranica. Također ima mogućnost izrade dinamičkih aplikacija koristeći server strane prikaza, što znači da se HTML stranica generira na serveru prije nego što se pošalje korisniku [1]. Da bi pojednostavili izradu aplikacije, unutar same NextJs aplikacije moramo dodati (instalirati) određene pakete, a to su:
Tailwind CSS – framework koji olakšava razvoj responzivnog i brzog korisničkog sučelja. Uz pomoć Tailwind CSS-a, razvijatelji mogu brzo i jednostavno stvarati prilagođene dizajne bez pisanja puno CSS koda. Tailwind CSS dolazi sa setom klasa koje se mogu primijeniti na HTML elemente kako bi se primijenio stil. Ove klase se temelje na namjernom dizajnu, što znači da se klasa koja opisuje izgled elementa naziva prema svrsi elementa, a ne prema stilu koji primjenjuje. Na primjer, klasa bg-red-500 postavlja pozadinu elementa na crvenu boju sa intenzitetom 500. Tailwind CSS također nudi mogućnosti za prilagođavanje klasa, dodavanje novih klasa i korištenje promjenjivih varijabli za olakšano ažuriranje stilova kroz čitavu aplikaciju [2].

Redux-toolkit – služi za dohvaćanje i spremanje podataka u aplikaciji, te da imamo mogućnost dohvaćanja podataka kroz samu aplikaciju. On uključuje nekoliko funkcija koje pojednostavljuju najčešće slučajeve korištenja Reduxa, uključujući postavljanje trgovine (engl. store-a), definiranje reduktora, nepromjenjivu logiku ažuriranja, pa čak i stvaranje cijelih *odsječaka* stanja odjednom bez ručnog pisanja kreatora radnji ili tipova radnji. Također uključuje najčešće korištene Redux dodatke, poput Redux Thunk za asinkronu logiku i Reselect za pisanje funkcija odabira, tako da se mogu odmah koristiti [3].

Socket-io client – omogućuje nam primanje i dohvaćanje samih podataka u pravom vremenu *slušajući* server stranu.

React hook form – omogućuje jednostavno i intuitivno upravljanje stanjem formi, validacijom i rukovanjem podataka bez potrebe za velikom količinom koda [4].

Yup – jako se dobro slaže s React hook form bibliotekom, te služi za validaciju formi.

React Toastify – služi za prikazivanje obavijesti (notifikacija) u aplikaciji na način da ne remete korisničko iskustvo i ne zahtjeva pažnju korisnika na duže vrijeme.

3.2. Server

Internet za izradu samog server dijela će se koristiti NodeJS kao primarna tehnologija u kombinaciji s Express frameworkom. „Node.js je open-source platforma za izgradnju aplikacija na serverskoj strani. Omogućuje programerima izgradnju brzih, skalabilnih i visoko performantnih aplikacija. Temelji se na događajnoj petlji (event loop) koja omogućuje asinkroni I/O operacije, što znači da aplikacije mogu nastaviti raditi dok se događa I/O operacija poput učitavanja datoteke ili izvršavanja HTTP zahtjeva“ [1]. Node.js ima veliku zajednicu programera koji su izgradili velik broj modula, biblioteka i okvira za rad s Node.js-om. To čini Node.js fleksibilnim i pogodnim za različite vrste aplikacija, uključujući web aplikacije, aplikacije za razmjenu podataka u stvarnom vremenu i API-je. Node.js također ima integrirani paketni menadžer (engl. npm), koji olakšava instaliranje, ažuriranje i upravljanje modulima i bibliotekama za aplikacije izgrađene na Node.js-u. Kao bazu podataka koristiti će se MongoDB. „MongoDB je besplatna baza podataka koja je dizajnirana da bude fleksibilna i visoko dostupana. To je open-source NoSQL baza podataka koja koristi JSON-sličan format za pohranu podataka. MongoDB se često koristi za izgradnju web aplikacija i drugih

softverskih sustava koji zahtijevaju skalabilnost i brz pristup podacima. To omogućuje programerima da lako skaliraju bazu podataka prema potrebi i prilagode je svojim potrebama“ [7]. Također kako bi pojednostavili izradu aplikacije, unutar same NodeJS aplikacije moramo dodati (instalirati) određene pakete, a to su:
Express – „framework za server aplikacije u NodeJS-u koji pruža mnoge ugrađene funkcije, uključujući usmjeravanje (engl. routing), obradu HTTP zahtjeva i odgovora, upravljanje pogreškama, podršku za sredinu (engl. middleware) i još mnogo toga. Omogućuje programerima da brzo i jednostavno izgrade aplikacije bez potrebe za puno koda“ [6].

Bcrypt - omogućuje sažimanje lozinke (engl. password hashing) za sigurno pohranjivanje lozinki. Namijenjen je zaštiti lozinki od brute-force napada i drugih vrsta napada,

Nodemon - alat koji omogućuje programerima da brzo ažuriraju svoju aplikaciju bez potrebe da ručno ponovno pokreću server nakon svake promjene,

Validator – služi za validaciju varijabli koje smo primili sa strane clienta,

Socket.io – „omogućuje pravovremenu komunikaciju između klijenta i servera bez potrebe za stalnim slanjem zahtjeva i odgovora. Pruža funkcionalnosti poput višestrukih soba, emitiranja događaja i automatskog ponovnog povezivanja“ [8].

4. PROCES IZRADE APLIKACIJE

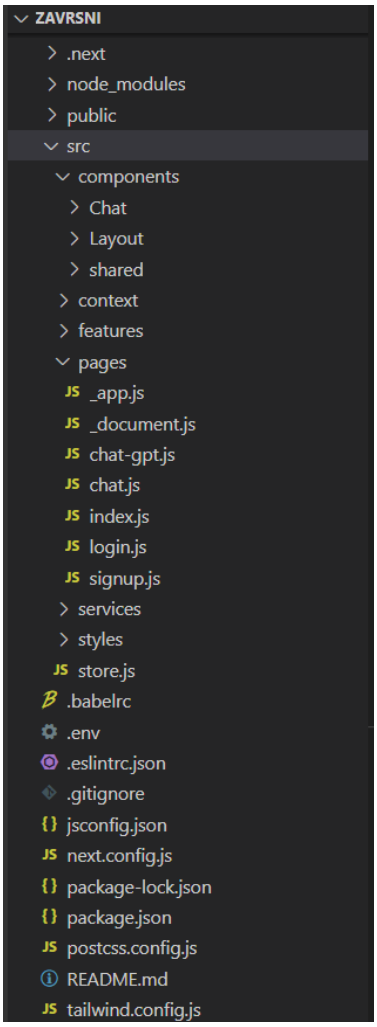
Samu proceduru izrade aplikacije mogu razdijeliti na 3 dijela, a to su: dizajn korisničkog sučelja, izrada korisničkog sučelja (engl. frontend), izrada pozadinske aplikacije (engl. backend-a). Svaki proces je važan kako bi aplikacija bila kvalitetna te ugodna za korištenje krajnjem korisniku.

4.1. Dizajn korisničkog sučelja

Kako bi aplikacija bila ugodna korisniku vrlo je važno izdvojiti vremena i izabrati dobar dizajn, jer to je ipak ono što korisnik vidi i koristi. Važno je znati da se dizajn korisničkog sučelja ne odnosi samo na izgled aplikacije, već i na njezinu funkcionalnost i korisničko iskustvo. U izgledu same aplikacije je važno izabrati odgovarajuće boje, u ovoj aplikaciji je izabrana metoda primarne i naglasne boje, gdje se dvije boje protežu kroz cijelu aplikaciju, a pozadinsku boju sam odlučio uzeti tamniju jer osobno mislim da je ugodnije oku. Sučelje je izrađeno tako da korisnik jednostavno navigira kroz aplikaciju i također je responzivno za mobilne uređaje.

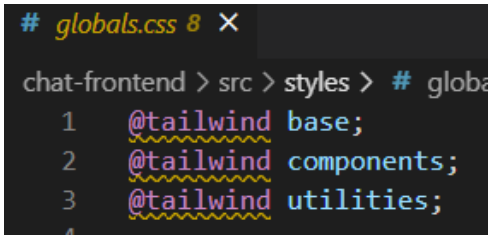
4.2. Korisničko sučelje (engl. Frontend)

Kao što je već prethodno napomenuto za izradu tzv. frontend-a će se koristiti NextJs. Kako bi kreirali projekt u NextJs-u mora biti već prethodno instaliran node na računalu koji pruža npm paket. Da se kreira početni projekt potrebno je koristiti naredbu *npm create-next-app@latest* ., koja će kreirati početni projekt u direktoriju u kojem se nalazimo. Kada se projekt kreira, potrebno je instalirati sve pakete koji će se koristiti te prilagoditi projekt. Vrlo je važno napraviti urednu strukturu datoteka i direktorija kako bi dalje bilo što lakše raditi projektom. NextJs nam pruža već dio strukture, kao što je *pages* direktorij u kojem kreiramo sve rute za aplikaciju i u kojem se nalazi Next-ov ugrađeni *routing* sustav koji nam pomaže da navigiramo kroz aplikaciju. Uz to je potrebno izraditi *components* direktorij gdje će se spremati sve komponente koji će se koristiti u aplikaciji.



Slika 4.1. Struktura datoteka i direktorija na završnici projekta

Kada se napravi struktura potrebno je postaviti *tailwind naredbama npm install -D tailwindcss* i „npx tailwindcss init“ koje će nam kreirati tailwind.config.js datoteku. Kako bi se omogućilo pisanje css-a u tailwindu moramo u *globals.css* datoteki dodati tailwind-ove klase kao na slici 3.2.



Slika 4.2. Prikaz dodavanja tailwind-ovih klasa u „globals.css“ datoteku

4.2.1. Postavljanje „react-redux“ trgovine (engl. store)

Sljedeći korak postavljanja projekta je postavljanje *react-redux-a*. Kao što je već prethodno navedeno on nam služi da pamtimo stanja određenih varijabli kroz cijelu aplikaciju. U ovoj aplikaciji react-redux će služiti za pamćenje korisnika kroz aplikaciju i upravljanje „api end-point-ima“ (api endpoint-i su rute na backend-u pomoću kojih dohvaćamo podatke). Prvo se mora kreirati samu trgovinu u kojoj se koriste određene varijable i funckije iz *react-redux* i *@reduxjs/toolkit* paketa. Mora se kreirati tzv. *reducer* koja je zapravo funkcija koja definira kako se aplikacijsko stanje mijenja u odgovoru na akcije koje su poslane u trgovinu. *Reducer* prima trenutno stanje aplikacije i akciju koja se izvršava, a zatim vraća novu verziju stanja. U ovoj aplikaciji reducer se sastoji od *userSlice* objekta koja zapravo služi za upravljanje obavijestima kada korisniku stignu nove poruke, te upravljanju *api* rutama.

```
4   export const userSlice = createSlice({
5     name: "user",
6     initialState: null,
7     reducers: {
8       addNotifications: (state, { payload }) => {
9         if (state.newMessages[payload]) {
10          state.newMessages[payload] = state.newMessages[payload] +
11        } else {
12          state.newMessages[payload] = 1;
13        }
14      },
15      resetNotifications: (state, { payload }) => {
16        delete state.newMessages[payload];
17      },
18    },
19
20    extraReducers: (builder) => {
21      // save user after signup
22      builder.addMatcher(
23        appApi.endpoints.signupUser.matchFulfilled,
24        (state, { payload }) => payload
25      );
26      // save user after login
27      builder.addMatcher(
28        appApi.endpoints.loginUser.matchFulfilled,
29        (state, { payload }) => payload
30      );
31      // logout: destroy user session
32      builder.addMatcher(appApi.endpoints.logoutUser.matchFulfilled
33    },
34  });
```

Slika 4.3. Prikaz „userSlice“ objekta

Također, drugi objekt koji ulazi u *reducer-a* je *appApi* objekt koji sadrži i upravlja svim autentikacijskim rutama koje ćemo koristiti prema „backend-u“, a to su registracija, prijava i odjava korisnika.

Kada se kreira reducer onda se mora kreirati reducer koji pamti stanje kroz aplikaciju tzv. *persistedReducer* kako ne bi izgubili korisnikovu login sesiju kada se stranica osvježi. Potom kreiramo samu trgovinu.

chat-frontend > src > JS store.js > ...

```
1   import { configureStore } from "@reduxjs/toolkit";
2   import userSlice from "./features/userSlice";
3   import appApi from "./services/appApi";
4
5   // persist our store
6   import storage from "redux-persist/lib/storage";
7   import { combineReducers } from "redux";
8   import { persistReducer } from "redux-persist";
9   import thunk from "redux-thunk";
10
11  // reducers
12  const reducer = combineReducers({
13    user: userSlice,
14    [appApi.reducerPath]: appApi.reducer,
15  });
16
17  const persistConfig = {
18    key: "root",
19    storage,
20    blackList: [appApi.reducerPath],
21  };
22
23  // persist our store
24  const persistedReducer = persistReducer(persistConfig, reducer);
25
26  // creating the store
27  const store = configureStore({
28    reducer: persistedReducer,
29    middleware: [thunk, appApi.middleware],
30  });
31
32  export default store;
33
```

Slika 4.4. Prikaz kreiranja „redux“ trgovine

4.2.2. Autentikacija

Kako bi korisnik mogao pristupiti aplikaciji potrebno je omogućiti korisniku da se registrira, te također prijavi na postojeći račun. Za registraciju i prijavu se koriste slični paketi, funckije i varijable. Za registraciju je potrebno kreirati formu gdje će se korisnika upitati njegovo ime, email i lozinka, te profilna fotografija ako korisnik želi unjeti. Za upravljanje formom koristi se *react-hook-form* paket u kompinaciji s *yup* paketom. Oni nam omogućuju validaciju forme, ukoliko

korisnik ne unese svoje ime, email ili lozinku, prikazat će mu se greška da nije unio određene stvari. Paket „react-hook-form“ nam također omogućuje spremanje varijabli koje je korisnik unio pri pozivu funkcije. U registraciji je potrebno napomenuti da imamo unos slike. Slike znaju ponekad predstavljati problem zato što zauzimaju znatno više memorije nego tekst, pa ih moramo negdje spremiti. Za spremanje slike se koristi *cloudinary api* sustav koji nam omogućuje besplatno spremanje slika u njihovu bazu. Za spremanje slike koristi se funkcija *uploadImage* u kojoj provjeravamo jel korisnik unio sliku, ako je onda zovemo *cloudinary* api te spremamo sliku u bazu.

```
const uploadImage = async () => {
  const data = new FormData();
  data.append("file", image);
  data.append("upload_preset", "f53ivazm");
  try {
    setUploadingImg(true);
    const res = await fetch(
      "https://api.cloudinary.com/v1_1/ddn2hr9cn/image/upload",
      {
        method: "post",
        body: data,
      }
    );
    const urlData = await res.json();
    setUploadingImg(false);
    return urlData.url;
  } catch (error) {
    setUploadingImg(false);
  }
};
```

Slika 4.5. Spremanje slike u „cloudinary“ bazu

Kada korisnik unese sve potrebne podatke, klikom na *Sign up* tipku se poziva funkcija *onSubmit* koja kreira korisnika u bazu. U funkciji *onSubmit* se uzimaju svi potrebni podatci iz forme, provjerava se jel korisnik unio sliku, te spremamo sliku pozivom na *uploadImage* funkciju. Kada se slika spremi zove se backend ruta *signupUser* i šalju se svi podatci, potom se korisnik kreira.

```
const onSubmit = async (data) => {
  const { email, password, name } = data;
  if (!image) {
    toast.error("Please upload your profile picture", {
      position: toast.POSITION.TOP_RIGHT,
    });
  }

  const url = await uploadImage(image);
  signupUser({ name, email, password, picture: url }).then(({ data }) => {
    if (data) {
      router.push("chat");
    }
  });
  reset();
};
```

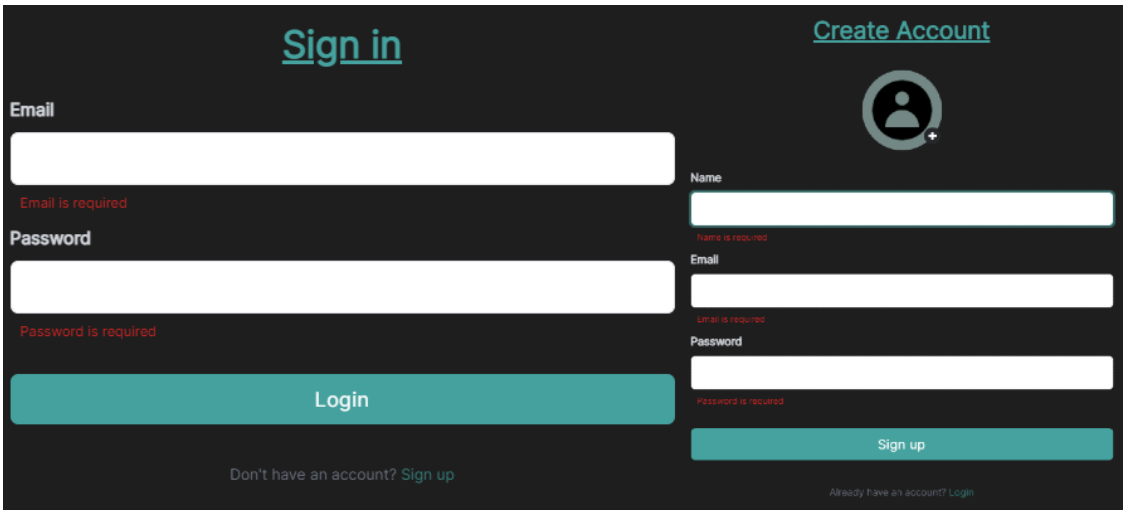
Slika 4.6. Poziv funkcije „onSubmit“ u registraciji

Prijava korisnika je vrlo slična registraciji korisnika, ali postoji nekoliko promjena. Za prijavu korisnika od korisnika tražimo email i lozinku postojećeg računa kako bi pristupio aplikaciji. U prijavi korisnika također ima funckcija *onSubmit* koja zove backend rutu *loginUser* u kojoj se šalje e-mail i lozinka koju je korisnik unio. U koliko je funkcija uspješno prošla navigira se korisnika na chat stranicu pomoću funkcije *router.push()*, te se javlja socketu da je došao novi korisnik pomoću funckije *socket.emit(„new-user“)* u kojoj zovemo tzv. *event* koji se naziva *new-user*. U koliko se ne javi socketu da je došao novi korisnik, drugim korisnicima se neće prikazati da je novoprijavljeni korisnik na mreži.

```
const onSubmit = (data) => {
  const { email, password } = data;
  loginUser({ email, password }).then(({ data }) => {
    if (data) {
      // socket work
      socket.emit("new-user");
      //navigate to the chat
      router.push("chat");
    }
  });
  reset();
};
```

Slika 4.7. Poziv funkcije „onSubmit“ u prijavi

Sučelja formi za prijavu i registraciju su vrlo jednostavna i minimalistična. Stil je izrađen pomoću *tailwind* klasa koje omogućuju brzo pisanje stilova. Primjeri formi se mogu vidjeti na slici 4.8. Primjer pisanja u *tailwind-u* se može vidjeti na slici 4.9.



Slika 4.8. Prikaz korisničkog sučelja formi

```
<div className="mt-6">
  <button className="w-full px-4 py-2 tracking-wide text-white transition-colors duration-200 transform bg-primary rounded-md
    Login
  </button>
</div>
```

Slika 4.9. Prikaz stiliziranja pomoću „tailwind“ klasa

4.2.3. Chat stranica

Chat stranica vizualno se sastoji od lijeve bočne trake (engl. sidebar) u kojem se prikazuju sve generalne grupe, kreirane grupe od korisnika i svi korisnici koji su trenutno na mreži i glavne forme za poruke koja omogućava prikaz i slanje poruka određenih korisnika i grupa.

U *Sidebar* komponenti, kada je korisnik prijavljen i kada se aplikacija osvježi prvo što se događa, poziva se u*seEffect hook* u kojem se određuje što će se raditi kada se stranica učita prvi puta. U tzv. *useEffect hook-u* postavlja se varijabla *currentRoom* na *general* kako bi se korisniku odma prikazao neki razgovor. Poslije toga se dohvaćaju sve javne sobe u kojima korisnik može pisati poruke pomoću funckcije *getRooms()*. Zatim se pozivaja funkcija *socket.emit(„join-room“,“general“)* koja kada korisnik tek dođe na stranicu, korisnika postavlja u *general* sobu, te *socket-emit(„new-user“)* i *socket-emit(„new-group“)*, gdje se omogućuje da korisnik može vidjeti promjene ukoliko se prijavi novi korisnik i ukoliko se kreira nova grupa.

```
useEffect(() => {
  if (user) {
    setCurrentRoom("general");
    getRooms();
    socket.emit("join-room", "general");
    socket.emit("new-user");
    socket.emit("new-group");
  }
}, []);
```

Slika 4.10. Prikaz „useEffect hook-a“ u lijevom „sidebar-u“

U Sidebar komponenti također ima *slušanje* novih korisnika funckijom *socket.off("new-user").on("new-user", (payload) => {setMembers(payload);})*, koja *sluša new-user* događaj koji prati dolazak novih korisnika u aplikaciju i funkcija *socket.off("new-group").on("new-group", (payload) => {setGroups(payload);})* koja prati kreiranje nove grupe za korisnika, koje se naposlijetku prikazuju u sidebar-u. Sličnom funkcijom *socket.off("notifications").on ("notifications", (room) => {if (currentRoom !== room) dispatch(addNotifications(room));})*, *slušaju* se notifikacije ukoliko je korisnik dobio novu poruku. Vrlo važna funckija za napomenuti je *joinRoom* funkcija prikaza na slici 4.11., koja omogućuje korisniku ulazak i navigiranje iz jednog razgovora u drugi. Funkcija će ukoliko je korisnik imao nepročitanih poruka (notifikacija) obrisati obavijesti.

```
const joinRoom = (room, isPublic = true) => {
  socket.emit("join-room", room, currentRoom);
  setCurrentRoom(room);
  if (isPublic) {
    setPrivateMemberMsg(null);
  }
  // dispatch for notifications
  dispatch(resetNotifications(room));
};
```

Slika 4.11. Prikaz „joinRoom“ funkcije

Važno je napomenuti da se korisnici i sobe spremaju u polja *members* i *rooms* koja se na kraju iteriraju pomoću ugrađene *javascript* funckije *map* koja je prikazan na slici 4.12.

Slika 4.12. Prikaz funkcije „map“

Osim *Sidebar* komponente, postoji i *messageForm* komponenta koja služi za prikaz razgovora i slanje poruka između korisnika. Kada korisnik klikne na neki razgovor u komponenti prvo što se dogodi jest da se dohvati trenutna soba na koju je korisnik kliknuo, te se pomoću socket funckije *socket.off("roommessages").on("room-messages", (roomMessages) => {setMessages(roomMessages);})* povezujemo na događaj *room-messages* koji omogućava *slušanje* poruka u samoj sobi, također i dohvaćanje poruka koje naposlijetku spremamo u *messages* varijablu. U samim porukama vrlo je važno prikazati u koje je vrijeme poruka poslana. Kako bi korisniku bilo lakše pročitati kada je poruka poslana mora se urediti datum. Funkcija koja izvršava uređivanje datuma zove se *getFormatedDate()*. Ona uzima trenutni datum u ISO formatu i pretvara ga u zapis „mm/dd/yyyy“ kao npr. „04/25/2023“. Funkcija *getFormattedDate()* prikazana je na slici 4.13.

```
const getFormattedDate = () => {
  const date = new Date();
  const year = date.getFullYear();

  let month = (1 + date.getMonth()).toString();
  month = month.length > 1 ? month : "0" + month;

  let day = date.getDate().toString();
  day = day.length > 1 ? day : "0" + day;

  return month + "/" + day + "/" + year;
};
const todayDate = getFormattedDate();
```

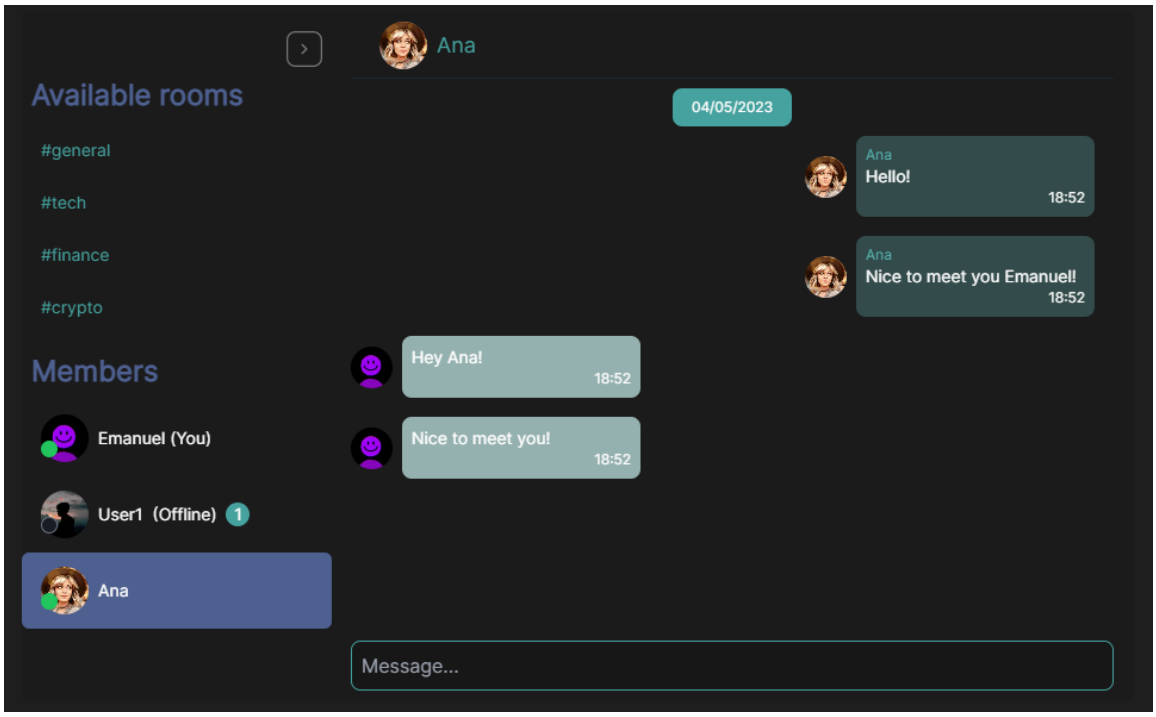
Slika 4.13. Prikaz funkcije „getFormattedDate“

Još jedna od funkcionalnosti za napomenuti u *MessageForm* komponenti je samo slanje poruka. Izvršava se funkcijom *onSubmit()* prikazanoj na slici 4.14. Funkcija uzima korisnikovu poruku, te provjerava ima li poruke, ako je prazna, daljni slijed funkcije neće se izvršiti. Zatim uzima se trenutno vrijeme u satima i minutama i pretvara se u olakšani napisani format „hh:mm“, npr. „16:40“. Potom uzimamo trenutnu sobu, te šaljemo događaj *socket-u* funkcijom *socket.emit("message-room", roomId, message, user, time, todayDate)* kojoj se prosljeđuju svi potrebni podatci za poruku koja se šalje.

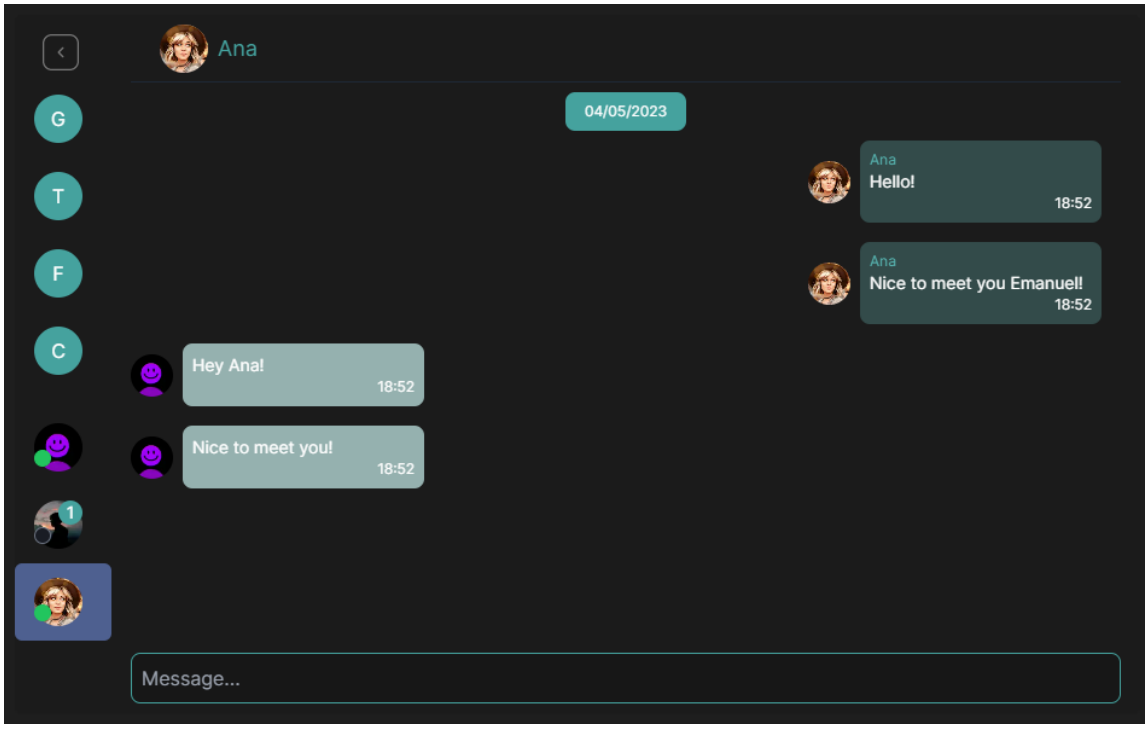
```
const onSubmit = (data) => {
  const { message } = data;
  if (!message) return;
  const today = new Date();
  const minutes =
    today.getMinutes() < 10 ? "0" + today.getMinutes() : today.getMinutes();
  const time = today.getHours() + ":" + minutes;
  const roomId = currentRoom;
  socket.emit("message-room", roomId, message, user, time, todayDate);
  reset();
};
```

Slika 4.14. Prikaz funkcije „onSubmit“ za slanje poruke

Korisničko sučelje samog chata prikazano je na slici 4.15. i 4.16. Sučelje je izrađeno minimalistički s ugodnim bojama te je responzivno za mobilne uređaje.



Slika 4.15. Prikaz chat aplikacije s otvorenom „ladicom“



Slika 4.16. Prikaz chat aplikacije s zatvorenom „ladicom“

4.2.4. „Chatify“

„Chatify“ je izrazito funkcionalna značajka aplikacije MessageMe koja omogućuje korisniku da razgovara sa umjetnom inteligencijom (eng. artificial intelligence - AI) koju je razvio „OpenAI“. OpenAI nudi svojim korisnicima da koriste chat te također nude web programerima da koriste njihov api kako bi implementirali njihov chat u svoju aplikaciju. *Istrenirali smo model pod nazivom ChatGPT koji komunicira na način razgovora. Format dijaloga omogućuje ChatGPT-u da odgovori na dodatna pitanja, prizna svoje pogreške, izazove netočne premise i odbije neprikladne zahtjeve* (openai.com). Implementacija „Chatify“ stranice je vrlo slična „Chat“ stranici. Izrađena je jednostavna forma za slanje poruka te prikaz prikaz trenutnog razgovora. Kada korisnik pritisne enter i pošalje poruku zove se funkcija *handleSend* koja provjerava je li korisnik unio poruku, ukoliko je, izrađuje se *newMessage* objekt koji sadrži u sebi sve potrebne informacije za poruku i dodaje ga u m*essages* polje koje sadrži sve napisane poruke.

```
const handleSend = async (e) => {
  e.preventDefault();
  if (!message) return;

  const newMessage = {
    message,
    direction: "outgoing",
    sender: "user",
    sentTime: format(new Date(), "hh:mm"),
  };

  const newMessages = [...messages, newMessage];

  setMessages(newMessages);

  setIsTyping(true);
  await processMessageToChatGPT(newMessages);
};
```

Slika 4.17. Prikaz funckije „handleSend“

Zatim se zove funkcija *processMessageToChatGPT(chatMessages)* koja uzima polje svih poruka i obavlja api poziv na „OpenAI“ chat. U api je bitno dodati autorizacijski ključ koji se može dobiti na „OpenAI“ stranici. No prije toga važno je svakoj poruci dodati *role* svojstvo kako bi razdijelili poruke koje je korisnik poslao. Poruke od umjetne inteligencije dodajemo pomoću funkcije *map*. Ukoliko je api vratio uspješan odgovor, sprema se nova poruka u *messages* polje.

```
async function processMessageToChatGPT(chatMessages) {
  let apiMessages = chatMessages.map((messageObject) => {
    let role = "";
    if (messageObject.sender === "Chatify") {
      role = "assistant";
    } else {
      role = "user";
    }
    return { role: role, content: messageObject.message };
  });

  const apiRequestBody = {
    model: "gpt-3.5-turbo",
    messages: [
      ...apiMessages, // The messages from our chat with ChatGPT
    ],
  };

  await fetch("https://api.openai.com/v1/chat/completions", {
    method: "POST",
    headers: {
      Authorization: "Bearer " + process.env.NEXT_PUBLIC_OPENAI_API_KEY,
      "Content-Type": "application/json",
    },
    body: JSON.stringify(apiRequestBody),
  })
    .then((data) => {
      return data.json();
    })
    .then((data) => {
      console.log(data);
      setMessages([
        ...chatMessages,
        {
          message: data.choices[0].message.content,
          sender: "Chatify",
          sentTime: format(new Date(), "hh:mm"),
        },
      ]);
      setIsTyping(false);
```

Slika 4.18.. Prikaz funckije „processMessageToChatGPT“

Slika 4.19.. Prikaz „Chatify“ sučelja

4.3. Pozadinska aplikacija (engl. Backend)

Pri izradi pozadinske aplikacije važna su 3. koraka, a to su: postavljanje servera, postavljanje samih modela koji će se koristiti u aplikaciji te izrada ruta i njihovih funkcionalnosti.

4.3.1. Postavljanje servera

Kako bi se postavio server potrebno je pozvati određene funkcije i slijediti određene korake kako bi se mogao uspješno koristiti i spremati podatke u bazu. Kao što je prethodno napomenuto za spremanje podatak u bazu se koristi „MongoDB“. Kako bi se povezali na „MongoDB“ server koristi se funkcija *mongose.connect()* kojoj se prosljeđuje poveznica od servera koja se kreira na „MongoDB“ web stranici.

```
const mongoose = require("mongoose");
require("dotenv").config();

// mongoose.set("strictQuery", false);

mongoose.connect(
  `mongodb+srv://${process.env.DB_USER}:${process.env.DB_PW}@cluster0.
  () => {
    console.log("connected to mongodb");
  }
);
```

Slika 4.20. Primjer povezivanja na „MongoDB“ bazu

Nakon što povezivanje na bazu uspješno prođe mora se otvoriti *http* luka (engl. port) i *socket.io* luka (engl. port) za *slušanje* promjena uživo. Primjer otvaranja port-ova može se vidjeti na slici 4.21. Zatim je potrebno reći serveru da će se koristiti rute iz drugih datoteka pomoću funkcije *app.use()*. Naposlijetku se koristi funkcija *server.listen(PORT,()=>{})* koja nam omogućava da baza radi i *sluša* nadolazeće promjene i zahtjeve.

```
const server = require("http").createServer(app);
const PORT = 5001;
const io = require("socket.io")(server, {
  cors: {
    origin: "http://localhost:3000",
    methods: ["GET", "POST"],
  },
});
```

Slika 4.21. Primjer otvaranja port-ova

4.3.2. User model

Kako bi se kreirali korisnici u bazu potrebno je napraviti *user* objekt. Koristimo tzv. *mongoose.Schema()* kako bi kreirali model. Funkciji se predaju svi podatci koje želimo da korisnik ima kao npr. ime, e-mail, lozinka, slika, itd.

```
const UserSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "Can't be blank"],
    },
    email: {
      type: String,
      lowercase: true,
      unique: true,
      required: [true, "Can't be blank"],
      index: true,
      validate: [isEmail, "invalid email"],
    },
    password: {
      type: String,
      required: [true, "Can't be blank"],
    },
    picture: {
      type: String,
    },
    newMessages: {
      type: Object,
      default: {},
    },
    status: {
      type: String,
      default: "online",
    },
  },
  { minimize: false }
);
```

Slika 4.22. Primjer „user“ modela

Vrlo važne funkcionalnosti za *user-a* su *hashiranje* lozinke i pretraga korisnika po e-mail-u. Kada korisnik unese lozinku, radi sigurnosti, lozinku je potrebno *hashirati*. Za *hashiranje* lozinke se koristi *bcrypt* paket, koji dobivenu lozinku *hashira* u 128 bit-ni string, npr. „63fb7f6a644f3bb75689681a-63fb7f6a644f3bb75689681a“. Kada se korisnik prijavljuje potrebno je provjeriti je li uneseni e-mail postoji u bazi. E-mail je unikatni identifikator za korisnike, zato pomoću predanog e-mail-a se mora provjeriti postoji li korisnik u bazi, ako postoji, te ako je lozinka točna korisnik je uspješno logiran.

```
UserSchema.pre("save", function (next) {
  const user = this;
  if (!user.isModified("password")) return next();

  bcrypt.genSalt(10, function (err, salt) {
    if (err) return next(err);

    bcrypt.hash(user.password, salt, function (err, hash) {
      if (err) return next(err);

      user.password = hash;
      next();
    });
  });
});
```

Slika 4.23. Primjer „hash-iranja“ lozinke

```
UserSchema.statics.findByCredentials = async function (email, password) {
  const user = await User.findOne({ email });
  if (!user) throw new Error("invalid email or password");

  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) throw new Error("invalid email or password");
  return user;
};
```

Slika 4.24. Primjer pronalaženja korisnika prema unesenom email-u

4.3.3. Korisnikove rute

Registracija i prijava su dvije najosnovnije stvari ako imamo korisnike, zato se moraju omogućiti. Upravljanje rutama vršimo pomoću *express* rutera koji nam omogućuje kreiranje ruta. Registracija korisnika je vrlo jednostavna, podatke koje dobijemo s prednje strane (engl. frontend-a) (ime, e-mail, lozinku i fotografiju) koristimo kako bi kreirali korisnika, ukoliko postoji korisnik s unesenim e-mailom prikazuje se greška da korisnik već postoji. Prijava korisnika je slična registraciji, samo je potreban e-mail i lozinka pomoću kojih pozivamo gore navedenu funkciju *findByCredentials(email,password)* koja će vratiti je li korisnik unio točan e-mail i lozinku. Ukoliko je, korisniku se mijenja status na *online* (prijavljen) i vraća se korisnik na frontend.

```
// creating user
router.post("/", async (req, res) => {
  try {
    const { name, email, password, picture } = req.body;
    const user = await User.create({ name, email, password, picture });
    res.status(201).json(user);
  } catch (e) {
    let msg;
    if (e.code == 11000) {
      msg = "User already exists";
    } else {
      msg = e.message;
    }
    res.status(400).json(msg);
  }
});

// login user
router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findByCredentials(email, password);
    user.status = "online";
    await user.save();
    res.status(200).json(user);
  } catch (e) {
    res.status(400).json(e.message);
  }
});
```

Slika 4.25. Primjer ruta za prijavu i registraciju

4.3.4. Message model

Message model je vrlo jednostavan model u kojem se također koristi *mongoose.Schema()*, te se kreira model koji sadrži u sebi poruku, od kuda je došla poruka, ID sobe, vrijeme i datum te za koga je namijenjena poruka.

```
const mongoose = require("mongoose");

const MessageSchema = new mongoose.Schema({
  content: String,
  from: Object,
  socketid: String,
  time: String,
  date: String,
  to: String,
});

const Message = mongoose.model("Message", MessageSchema);

module.exports = Message;
```

Slika 4.26. Prikaz modela za poruke (message modela)

4.3.5. Socket funkcionalnosti

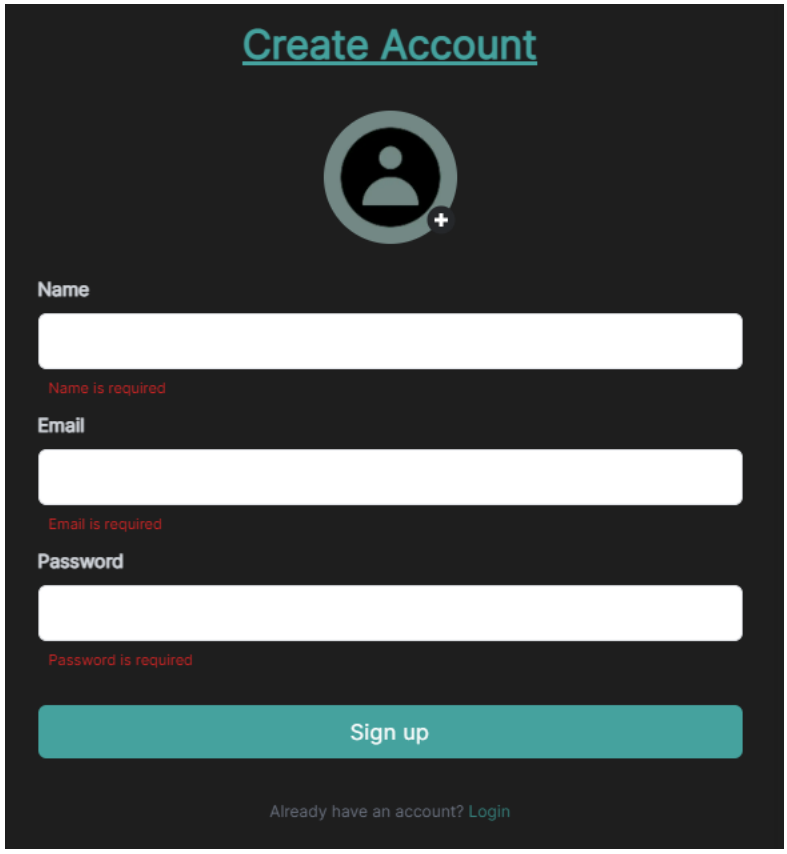
Već prethodno napomenuto u sekciji korisničkog sučelja, postoje *new-user*, *join-room*, *message-room* događaji koji omogućuju praćenje promjena uživo, kako su se pratile promjene potrebno je na frontend-u *slušati* određene događaje. *new-user* događaj omogućuje kada se u bazi registrira novi korisnik da se vidi ta promjena, vrlo sličan događaj *new-group* omogućuje kada se u bazi stvori nova grupa da se također vidi ta promjena. *join-room* događaj nam omogućuje povezivanje iz jedne sobe u drugu sobu, tj. drugi razgovor. Vrlo je bitno napustiti prethodnu sobu kako se server ne bi pretrpao korisnicima. Kada korisnik prijeđe u drugu sobu ovaj događaj mu vraća sve sortirane poruke po vremenu iz nove sobe. Posljednji događaj *message-room* omogućuje slanje novih poruka i primanje obavijesti. Kada se događaj pozove, kreira se nova poruka za određenu sobu, te se dohvaćaju sve poruke iz te sobe i vraćaju nazad na *room-messages* događaj. Poslije toga pomoću *socket.broadcast-emit()* funkcije koja samo obavijesti slušatelje da je stigla nova poruka.

```
socket.on("new-user", async () => {
  const members = await User.find();
  io.emit("new-user", members);
});

socket.on("join-room", async (newRoom, previousRoom) => {
  socket.join(newRoom);
  socket.leave(previousRoom);
  let roomMessages = await getLastMessagesFromRoom(newRoom);
  roomMessages = sortRoomMessagesByDate(roomMessages);
  socket.emit("room-messages", roomMessages);
});

socket.on("message-room", async (room, content, sender, time, date) => {
  await Message.create({
    content,
    from: sender,
    time,
    date,
    to: room,
  });
  let roomMessages = await getLastMessagesFromRoom(room);
  roomMessages = sortRoomMessagesByDate(roomMessages);
  // sending message to room
  io.to(room).emit("room-messages", roomMessages);
  socket.broadcast.emit("notifications", room);
});
```

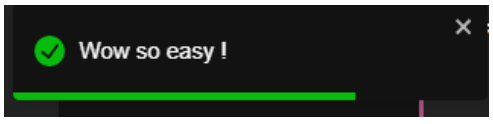
Slika 4.27. Prikaz funkcionalnosti socket-a

5. RAD S APLIKACIJOM

U radu s aplikacijom, cilj je pružiti korisniku što jednostavnije sučelje za koristiti, te mu dati dovoljno informacija kao što su tzv. *toastovi* i greške prikazane jasnom crvenom bojom gdje koje korisniku pokazuju što treba ispraviti prilikom ispunjavanja formi. Prikaz *toastova* i grešaka se mogu vidjeti na slikama 5.1. i 5.2.



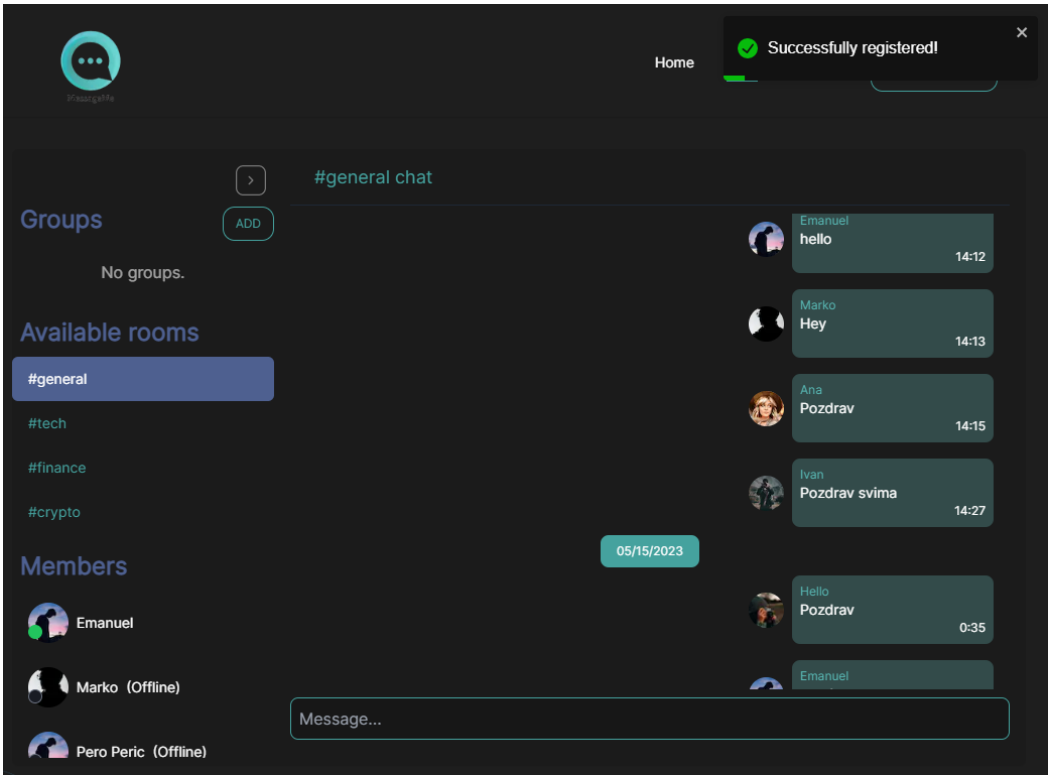
Slika 5.1. Prikaz grešaka



Slika 5.2. Prikaz toastova

Kako bi korisnik koristio pristupio aplikaciji mora se prvo registrirati ili prijaviti. Kada se korisnik prijavi ili registrira aplikacija ga redirecta na *chat* stranicu. Chat stranica je minimalistički izrađena, sastoji se od lijevog bočnog prozora u kojem se mogu vidjeti generalne grupe, grupe u

kojima je korisnik te sve korisnike koji su pristupili aplikaciji, ako su korisnici trenutno prijavljeni bit će prikazan zeleni krug s korisnikovom profilnom slikom. Ukoliko korisnik primi novu poruku prikazat će se notifikacija pokraj imena korisnika koji je poslao poruku. te glavnog dijela u kojem se prikazuju poruke među korisnicima. Primjer prikaza sučelja za razgovor se može vidjeti na slici 5.3.



Slika 5.2. Prikaz sučelja za razgovor

Ukoliko korisnik želi razgovarati s umjetnom inteligencijom, klikom na *chatify* u zaglavlju može navigirati na *chatify* stranicu. *Chatify* stranica se sastoji od jednostavnog korisničkog sučelja gdje se mogu vidjeti poruke od korisnika i *bota* za razgovor, te jednostavne input forme pomoću koje može poslati poruku.

6. ZAKLJUČAK

U ovom završnom radu uspješno je razvijena web aplikaciju za razmjenu poruka kojoj koja se sastoji od jednostavnog korisničkog i aplikacijskog sučelja koji se povezuju pomoću *web-socketa*. Aplikacija ima slične funkcionalnosti poput WhatsApp-a ili Facebook Messengera. Korištenjem modernih tehnologija i naprednog programskog jezika, omogućeno je korisnicima brzo i jednostavno sučelje za razmjenjivanje poruka koje sadržava visoku razinu sigurnosti. Ova aplikacija također nudi dodatne značajke poput grupnih razgovora što poboljšava korisničko iskustvo. U ovom radu je bilo mnogo tehničkih problema u kojima je bio zadatak ih riješiti, te pomoću njih naučiti kako izraditi sučelje u programskom jeziku „NextJs“, te backend u programskom jeziku „NodeJs“. Jedna od najvažnijih funkcionalnosti koje smo naučili u ovom radu je kako pravljati *web-socketima* i kako omogućiti pravovremeno u komunikaciju između korisnika. U zaključku, moja aplikacija za razmjenu poruka je uspješno razvijena i testirana, te sam uvjeren da će pružiti korisnicima jednostavno, sigurno i ugodno iskustvo razmjena poruka, te vjerujem da će moj rad pružiti korisne smjernice drugim programerima za daljnji razvoj ovakvih aplikacija.

SAŽETAK

U ovom završnom radu opisuje se proces izrade aplikacije za razmjenu poruka. Cilj projekta bio je razviti aplikaciju koja će omogućiti korisnicima da međusobno komuniciraju putem poruka. Korištena je tehnologija NextJs za izradu korisničkog sučelja i NodeJs za izradu pozadinske aplikacije. Rad opisuje korake izrade aplikacije, počevši od izrade i dizajniranja korisničkog sučelja i funkcionalnosti, te do implementacije pozadinske aplikacije. Opisani su ključni izazovi s kojima smo se susreli tijekom izrade aplikacije. Konačni rezultat je aplikacija koja omogućava korisnicima da se jednostavno povežu putem poruka. Aplikacija ima intuitivno korisničko sučelje i nudi nekoliko funkcionalnosti kao što su stvaranje grupa za razmjenu poruka i razgovor s umjetnom inteligencijom. Ovaj rad daje uvid u proces izrade web aplikacije za razmjenu poruka.

Ključne riječi:
NextJs,
NodeJs,
razgovori,
pravovremeno,
poruka,
Socket.

ABSTRACT

Title: MessageMe – a messaging website.

This final paper describes the process of developing a messaging application. The project's goal was to create an application that allows users to communicate with each other via messages. NextJs technology was used to create the user interface and NodeJs for developing the backend application. The final paper shows steps for developing the application, starting from designing the user interface and functionality, to implementing the backend application. The key challenges encountered during the application development process are described. The final result is an application that enables users to easily connect via messages. The application has an intuitive user interface and offers several features such as creating groups for message exchange and conversing with artificial intelligence. This paper provides insight into the process of creating a web-based messaging application.

Keywords:

chats,

message,

NextJs,

NodeJs,

real-time,

Socket.

LITERATURA

[1] NextJS, https://nextjs.org, pristup: 21.03.2023.

[2] Tailwind CSS, https://tailwindcss.com, pristup: 21.03.2023.

[3] Redux Toolkit, https://redux.js.org/redux-toolkit/overview pristup: 21.03.2023.

[4] Redux Hook Form, https://react-hook-form.com pristup: 21.03.2023.

[5] NodeJS, https://nodejs.org.en pristup: 21.03.2023.

[6] ExpressJS, https://expressjs.com pristup: 21.03.2023.

[7] MongoDB, https://www.mongodb.com pristup: 21.03.2023.

[8] Socket.io, https://socket.io pristup: 21.03.2023.

[9] OpenAI, https://openai.com pristup: 06.04.2023.

[10] WhatsApp, https://whatsapp.com pristup: 19.04.2023

[11] FacebookMessenger, https:// messenger.com pristup: 19.04.2023

POPIS OZNAKA I KRATICA

HTML - hipertekstualni označni jezik (engl. HyperText Markup Language)

CSS - kaskadni listovi stilova (engl. Cascading Style Sheets)

HTTP - protokol prijenosa hiperteksta (engl. Hyper Text Transfer protocol)

API - sučelje za programiranje aplikacija (engl. Application Programming Interface)

ŽIVOTOPIS

Emanuel Maričić rođen je u Osijeku, 24.12.1999. godine. 2006. godine upisuje osnovnu školu Ivana Gorana Kovačića u Đakovu koju završava 2014. godine te potom upisuje srednju strukovnu školu Antuna Horvata u Đakovu, smjer tehničar za mehatroniku. Nakon završetka srednjoškolskog obrazovanja 2018. godine posvećuje se radnim obvezama da bi u 2019. godini upisao stručni studij Računarstva na fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

X