Razvoj poslužiteljskih komponenti web aplikacije za stvaranje novih prijateljskih i ljubavnih veza

Kožić, Karlo Dini

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

KARLO DINI KOŽIĆ

RAZVOJ POSLUŽITELJSKIH KOMPONENTI WEB APLIKACIJE ZA STVARANJE NOVIH PRIJATELJSKIH I LJUBAVNIH VEZA

Diplomski rad

Pula, rujan, 2021. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

KARLO DINI KOŽIĆ

RAZVOJ POSLUŽITELJSKIH KOMPONENTI WEB APLIKACIJE ZA STVARANJE NOVIH PRIJATELJSKIH I LJUBAVNIH VEZA

Diplomski rad

JMBAG: 0246065152, redoviti student

Studijski smjer: Sveučilišni diplomski studiji Informatike

Predmet: Napredni algoritmi i strukture podataka

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Pula, rujan, 2021. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Karlo Dini Kožić 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 ikoji 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

U Puli, 19. rujna 2021. godine

IZJAVA

o korištenju autorskog djela

Ja, Karlo Dini Kožić dajem odobrenje Sveučilištu Jurja Dobrile

u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Razvoj poslužiteljski komponenti web aplikacije za stvaranje novih prijateljskih i ljubavnih veza“ 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, 19. rujna 2021. godine

Potpis

_____

Sadržaj

1. Uvod

Upoznavanje novih ljudi je sve teže u današnjem svijetu. Sve više i više ljudi traži nova poznanstva i ljubavi preko interneta te ih pokušava pronaći preko raznih foruma i socijalnih mreža, kao i preko samih aplikacija za stvaranje ljubavnih veza. S time je došla zamisao za ovaj projekt koji bi olakšao upoznavanja novih ljudi u blizini.

Aplikacija je osmišljena tako da se izbjegne površinsko upoznavanje ljudi. Ostale aplikacije omogućavaju „upoznavanje“ mnoštva ljudi što opterećuje krajnjeg korisnika da izabere osobu s kojom želi pričati. Taj fenomen se zove paradoks izbora (engl. The paradox of choice). Pomislilo bi se da kada se osobi omogući veći izbor, da joj je lakše pronaći izbor s kojim je zadovoljna, ali mnoštvo izbora zapravo zahtjeva više uložene energije i vremena za procjenu što je najbolje za tu osobu te je na kraju izbor koji izabere ostavlja s osjećajem nezadovoljstva (Lab, 2021). Kako bi se riješio taj paradoks odabran je sljedeći pristup: svaki korisnik ima određeni broj životnih bodova koji se oduzimaju za bilo koje radnje poput sviđa mi se, ne sviđa mi se i sličnih radnji koje bi trebale rezultirati pronalaskom ili odbijanjem novih odnosno starih korisnika. Tako se pokušava smanjiti izbor ljudi koji su korisniku ponuđeni. Smanjivanje životnih bodova kod „ne sviđa mi se“ funkcionalnosti služi u svrhu promoviranja ostajanja u kontaktu sa starim korisnicima. S time se poboljšava korisničko iskustvo i omogućava stvaranje kvalitetnijih odnosa među korisnicima.

Aplikacija se sastoji od klijentskog dijela i poslužiteljskog dijela. Kolegica Laura Sučić se u svom diplomskom radu bavi izradom klijentskih dijelova web aplikacije (Sučić, 2021) dok se ovaj rad bavi razvojem poslužiteljskih komponenti web aplikacije za stvaranje novih prijateljski i ljubavnih veza s modernim tehnologijama koje je lako održavati, skalirati i ažurirati. Zbog toga je kao rješenje odabran „RESTful“ server podijeljen u tri manja servisa.

Ovaj rad opisuje poslužiteljski dio aplikacije, koje komponente postoje i kako one međusobno komuniciraju i zajedno stvaraju jednu cjelinu, odnosno kako koja komponenta slaže podatke i što ti podatci predstavljaju, te kako se pristupa određenim funkcionalnostima poslužiteljskog dijela aplikacije, kako se pronalaze korisnici koji su slični ili korisnici koji su u blizini određenog korisnika.

2. Pregled tržišta

Današnje moderne tehnologije posreduju gotovo svakim aspektom ljudskoga života. Time su te tehnologije dobile nevjerojatnu moć nad svojim korisnicima te same mogu mijenjati načine ponašanja, želje, osjećaje i misli svojih korisnika. Nekada su se ljudi upoznavali pomoću obitelji, prijatelja ili poznanika kroz takozvane „weak-ties“ krugove, dok je danas Internet zamijenio socijalne interakcije tih skupina ljudi. U današnje se vrijeme više nije potrebno sastajati uživo s prijateljima kako bi se družili s njima, već se pomoću raznih aplikacija to može učiniti virtualno Internetom. To je umanjilo mogućnosti upoznavanja novih ljudi, potencijalnih prijatelja i ljubavnih partnera. Kao što je Internet smanjio potencijal upoznavanja novih ljudi, tako ga je i novim aplikacijama omogućio te dao još jedan odgovor na primarnu ljudsku potrebu druženja i zbližavanja s drugim ljudima. Na taj se način opet smanjila potreba za upoznavanjem novih ljudi uživo. Stigme koje su nekada postojale kod upoznavanja novih ljudi preko interneta su se povukle (Paska, 2020).

Najpoznatije aplikacije koje su odgovorile na ljudsku potrebu upoznavanja novih prijatelja i romantičnih partnera su: Tinder, OKCupid, Tantan, Badoo and Zoosk. Tinder i OKCupid su u vlasništvu američke tvrtke Match Group, Tantan je aplikacija kineskog porijekla, dok je Badoo ruskog. Od 2013. godine, Tinder je implementirao takozvani „swipe“ slika kao način dolaska do novog korisnika. „Swipe“ ulijevo označava da se korisnik ne želi spojiti s prikazanim korisnikom, dok „swipe“ udesno označava da želi. Na isti gore opisani način, Tinder spaja korisnike u vrijeme pisanja ovoga rada (Humphrey, 2021). Mehanika „swipe“ je od tada postala norma kod ostalih aplikacija. Sve gore naveden aplikacije svode se na sljedeće: profilnu stranicu s galerijom slika, korisnikovim interesima i kratkim opisom korisnika, razgovornim dijelom aplikacije gdje korisnik može komunicirati s drugim korisnicima te početnim dijelom aplikacije gdje su korisniku prikazani novih korisnici koje prije nije vidio. Prikazanim korisnicima korisnik može reći da mu se sviđaju („swipe“ udesno) ili ne sviđaju („swipe“ ulijevo). Ako su se međusobno sviđali (neke aplikacije to zovu „matched“), mogu započeti razgovor.

Korisnici aplikacija za upoznavanje novih prijatelja i romantičnih partnera imaju strah da ih drugi korisnici, kao što su prijatelji, obitelj, trenutni ili budući poslodavci, ne

prepoznaju (Solis i Wong, 2017). Taj strah kod korisnika prvenstveno nastaje zbog korištenja aplikacije tako da se pronađe partner s kojim bi imali seks bez obaveza. To upućuje da korisnici ne bi tražili seks bez obaveze s ljudima koje upoznaju preko svojih tzv. „weak-ties“. Korisnici koji traže ozbiljne veze i prijateljstva moraju se nositi s korisnicima koji to ne traže te im je zadovoljstvo korištenja aplikacije uveliko umanjeno. Premda su se stigme kod upoznavanja novih ljudi pomoću Interneta smanjile, nastale su stigme prema određenim aplikacijama zbog velikog broja korisnika koji ih koriste kao sredstvo za postizanje vlastitog trenutnog zadovoljstva. Korištenje aplikacija koje koriste „swipe“ mehaniku je povezano s povišenom razinom stresa, anksioznosti i depresije kod svojih korisnika naspram ljudi koji ne koriste takve aplikacije (Holtzhausen, i dr., 2020). Jedan od mogućih odgovora za lošije mentalno stanje korisnika takvih aplikacije se može pronaći u studiji „Hladne intimnosti: Oblikovanje čustvenega kapitalizma“.

Eva Illouz je 2010. godine provela istraživanje o kvantitativnim procesima korisnika aplikacija za upoznavanje ljubavnih partnera. Rezultati istraživanja pokazali su tendenciju da procjena potencijalnih partnera nalikuje na kupnju dobara u samoposluzi. (Illouz, 2010). To pokazuje kako korisnici ne gledaju na druge korisnike kao na ljude, nego kao na dobro za svoje vlastito zadovoljstvo. Time će korisnici pokušati maksimizirati svoje zadovoljstvo bez obzira na to kakve će osjećaje izazvati njihovi postupci kod drugih korisnika. Ako se pritom uzme u obzir doza anonimnosti i osjećaja manjka odgovornosti koji dolaze s upotrebom takvih aplikacija, lako je zaključiti da će većina odnosa među korisnicima ostati na površnoj razini međusobne razmjene dobara. Takva ponašanja stvaraju osjećaj frustracije i nezadovoljstva među korisnicima (Guo, 2021). Kao odgovor na te probleme, došlo je do ideje za ovaj rad. Rad opisuje razvoj poslužiteljskih komponenti aplikacije za stvaranje trajnih odnosa među korisnicima umanjujući osjećaj razmjene dobara kažnjavanjem korisnikovih pokušaja maksimizacije trenutnog zadovoljstva na dugotrajnu štetu iskustva korištenja aplikacije.

3. Korištene tehnologije

Tehnologija izabrana za izgradnju poslužiteljskog dijela aplikacije je Node.js run-time programsko okruženje. Node.js je izabran, jer omogućava izgradnju web servisa i mrežnih alata koristeći jedan od najpopularnijih programski jezika: JavaScript (Overflow, 2021).

3.1 Node.js

Node.js koristi V8 *„JavaScript engine“* izrađen od „Chromium Projekt“-a. Prva verzija V8 je izašla 2.9.2008. dok je Node.js napravio Ryan Dahl 2009. godine. Službeno je podržan na Linux, MacOs, Windows i dr. operacijskim sustavima.

Node.js je modularnog tipa pa svaki modul omogućava zasebne funkcionalnosti. Osnovni moduli koji dolaze s Node.js su modul za *„file system I/O“*, mrežne protokole (DNS, HTTP, TCP, TLS/SSL ili UDP), međuspremnik za binarne podatke, modul s kriptografskim funkcijama, podatkovnim tokovima i dr.. Moduli su napravljeni tako da koriste API dizajn kako bi smanjili kompleksnost pisanja serverske aplikacije. Krajnji korisnici mogu napraviti svoje vlastite module ili koristiti tuđe javno dostupne module pri izgradnji svoje aplikacije. Najpoznatiji repozitorij za Node.js module je „npm package manager“ koji omogućava jednostavno instaliranje i korištenje tuđih modula. Node.js koristi *„event - driven programming“* što omogućuje izgradnju brzih web servera u JavaScriptu. Tim pristupom omogućena je izgradnja skalabilnih servera bez korištenja više dretvi, korištenjem pojednostavljenog modela *„event-driven programming“* koji koristi „callbacks“ kao signale o završetku zadatka. Node.js spaja jednostavno korištenje skriptnih jezika (JavaScript) sa snagom „Unix“ mrežnog programiranja.

3.2 Node.js biblioteke

Tijekom ovog projekta korišteno je mnogo različitih modula preko „yarn package manager“-a kao što su Express.js, Socket.io, MongoDB, mongoose, itd.

Express.js je poslužiteljski okvir za izgradnju web aplikacija i API-ja. On je de facto standard poslužiteljski okvir za izradu Node.js aplikacija (Serby, 2021). Koristi minimalistički pristup s mnogo dodataka koji nisu u osnovnom paketu, a koji proširuju njegove funkcionalnosti prema potrebi. Express.js je izabran jer u sebi već ima način

za obradu REST metoda, omogućuje postavljenje takozvanog među softvera (*engl. Middleware*) kako bi se dodatno proširila obrada HTTP zahtjeva te definira tablicu ruta kojima se izvršavaju potrebne radnje navedene preko URL-a i HTTP metoda.

Socket.io je JavaScript biblioteka koja omogućava dvosmjernu komunikaciju između klijenta i poslužitelja u pravom (realnom) vremenu. Sastoji se od dva dijela: klijentskog dijela koji se pokreče pomoću web preglednika i poslužiteljskog dijela za Node.js aplikaciju. Oba dijela imaju gotovo identične API-je. Kao i Node.js, baziran je na „event-driven“ principu. Socket.io omogućava izgradnju analitike u pravome (realnom) vremenu, instantno slanje poruka i kolaboraciju u izgradnji dokumenata. U ovom projektu je korišten u svrhu izgradnje slanja poruka između dva korisnika u pravom vremenu.

3.3 Ostale korištene tehnologije

MongoDB je nerelacijska baza podataka koja koristi dokumente kako bi spremila podatke. Koristi dokumente koji sliče na JSON format. U projektu je korištena biblioteka mongoose ODM kako bi se olakšalo kontroliranje MongoDB baze podatka. Mongoose ODM koristi sheme kako bi modelirao podatke. Sadrži automatsku validaciju tipova, izgradnju upita i. dr.

JWT (JSON Web Token**)** je predloženi internetski standard za kreiranje kriptiranih podataka tipa JSON. Tokeni se mogu kriptirati privatnim ili javnim ključem.

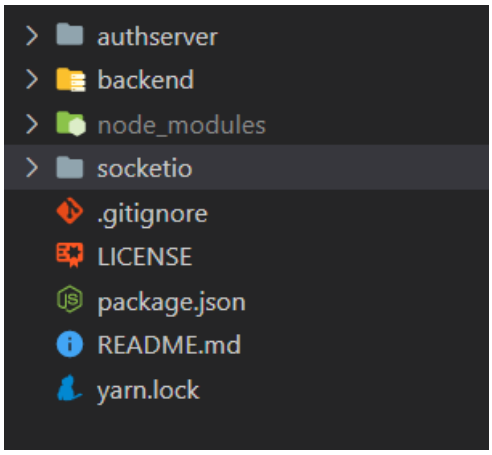
GeoJSON je format za zapisivanje različitih geografskih podatkovnih struktura. Trenutna formalna specifikacija GeoJSON formata izašla je 2016. od strane IETF (the Internet Engineering Task Force) organizacije (GEOJSON, 2021).

4. Arhitektura aplikacije

Poslužiteljski dio aplikacije je podijeljen u tri dijela:

- sustav za autentifikaciju
- sustav za upravljanjem korisničkim podatcima
- sustav za sinkronu komunikaciju.

Svaki od njih je zadužen za dio zahtjeva sustava. Sva tri sustava se pokreću kao zasebni procesi u trenutku pokretanja poslužitelja (servera). Na sljedećoj slici prikazanoj pod rednim brojem 1, vidljiva je podjela sustava u tri dijela.



Slika 1. Pregled triju sustava

Klijentski dio aplikacije je opisan u radu kolegice Laure Sučić (Sučić, 2021). Klijentski i poslužiteljski dijelovi aplikacije su samostalni i ne ovise jedan o drugome, već klijentski dio radi zahtjeve na poslužiteljski dio koji onda obrađuje te zahtjeve te na njih odgovara. Ako iz nekog razloga poslužiteljski dio aplikacije prestane raditi, klijentski dio se neće srušiti, ali neće biti funkcionalan. To omogućava ponovno pokretanje poslužiteljskog dijela aplikacije bez potrebe ponovnog pokretanja klijentskog dijela ili obratno.

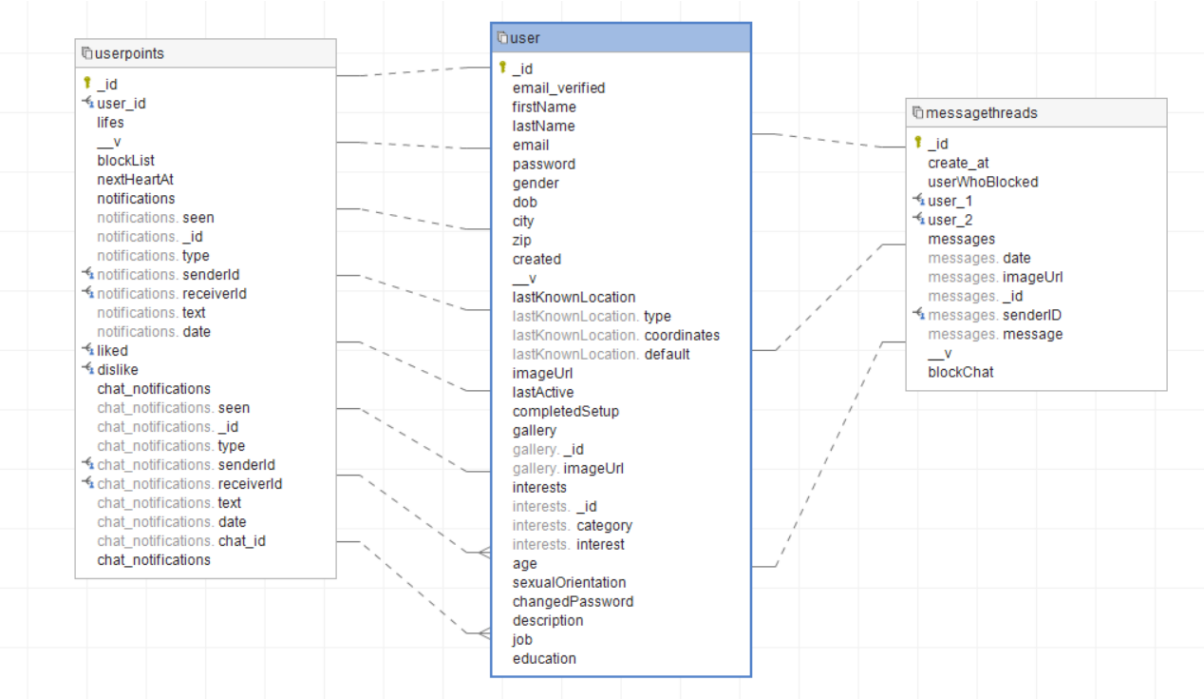
4.1 Cijelokupan pregled aplikacije

Jedini način na koji korisnik može upravljati ovom aplikacijom je preko klijentskih komponenti koje je u svrhu ovog rada napravila kolegica Laura Sučić (Sučić, 2021). Kada klijentske komponente žele obaviti radnje npr. prijave, registracije, dohvata korisničkih podataka, itd. moraju poslati HTTP zahtjev na poslužiteljski server.

Server će odgovoriti na te zahtjeve ili pozitivno, s traženim podatcima, ili negativno, s odgovarajućom porukom zašto je zahtjev odbijen. Kod razgovora između korisnika potrebno je ostaviti dvosmjerni kanal komunikacije između dva korisnika koja razgovaraju što je postignuto uz pomoć biblioteke „Socket.io“. Klijentski dio aplikacije pošalje event da se korisnik spojio na razgovor te se otvara kanal za dvosmjernu komunikaciju. Kada se potom drugi korisnik spoji na isti razgovor isto se šalje event pomoću „Socket.io“ biblioteke te su zajedno smješteni u isti kanal. Kada jedan korisnik pošalje drugome poruku, klijentski dio aplikacije pomoću biblioteke „Socket.io“ šalje event sustavu za sinkronu komunikaciju koja obrađuje taj event te označava da je poruka došla od jednog korisnika i da treba na klijentski dio „Socket.io“ biblioteke poslati vrijednost te poruke drugome korisniku. Kada korisnik npr. započne razgovor s drugim korisnikom, onda taj korisnik koji započinje razgovor gubi tzv. životne bodove. Na poslužiteljski server se šalje zahtjev da se od tog korisnika životni bodovi trebaju smanjiti, te se kao odgovor šalje podatak koliko korisnik ima životnih bodova i kada će mu se oni opet povećati. Kada dođe vrijeme za povećanje životnih bodova, server će ih povećati i spremiti promjene, a klijentski dio aplikacije će ponovo tražiti podatke o životnim bodovima.

4.2 Struktura podatka kroz sustave

Slijedećom slikom pod rednim brojem 2 je prikazan ER dijagramom koji prikazuje međuodnosi odnos tablica podataka. Te tablice su korištene za spremanje korisničkih podataka.



Slika 2. ER dijagram baze podataka

User tablica sadrži elemente poput imena (firstName), prezimena (lastName), lozinke (password), i dr. Element email mora biti jedinstven, dok se dob (age) korisnika računa tijekom kreacije novog korisnika od njegovog datuma rođenja (dob - date of brith). Element „lastKnownLocation“ je objekt koji sadrži zadnje poznate geografske širine i dužine. Prilikom kreacije uzima se lokacija njegove adrese kao početna geografska širina i dužina. Koristi format GeoJSON kako bi definirao poziciju korisnika. Element „ImageUrl“ označava URL do putanje korisnikove trenutne profilne slike. Element „Gallery“ je polje URL-ova slika koje je korisnik stavio u svoju galeriju. Element „Interests“ je polje objekata koji se sastoje od kategorije i interesa.

Model „MessageThread“ predstavlja razgovor između dva korisnika (user) zbog toga ima elemente user_1 i user_2 koji su vanjski ključevi na ID-ove korisnika. Korisnički ID-ovi moraju biti različiti te postoji samo jedan dokument tipa „MessageThread“ s određenom kombinacijom korisničkih ID-ova. Element

„Messages“ je polje objekta poruka. Svaki objekt poruka sadrži datum poruke, ID korisnika koji je poruku poslao te tekst ili URL slike poruke.

Model „UserPoints“ se koristi za dodate informacije o korisniku i služi za kontroliranje korisnikoivh životnih bodova (lifes). Bodovi se gube nakon što korisnik izvrši neku radnju kao što je pokretanje novog razgovora ili kada označi da mu se neki korisnik svidio. Životni bodovi se automatski dopunjavaju nakon određenog vremena te se spremaju u element „nextHeartAt“ što predstavlja vrijeme kada će novi životni bod biti dostupan. Ako korisnik ima maksimalan broj životnih bodova, postavlja se na prazno (null). Sljedećom slikom pod rednim brojem 3 prikazan je kod funkcije „addUserPointsAfterTime“.

```
export async function addUserPointAfterTime(user_id){

  timeOut=null;
  const uPoints = await UserPoints.find({user_id:user_id})
  let lifes = uPoints[0].lifes;
  lifes++
  console.log(lifes-1+"=>"+lifes)
  uPoints[0].lifes = lifes
  uPoints[0].nextHeartAt =  Date.now()+LIFE_REFILL_TIME


  if(lifes<5){
  timeOut= setTimeout(async()=>{await addUserPointAfterTime(user_id)},LIFE_REFILL_TIME)
  }else{
    uPoints[0].nextHeartAt =null
  }
  await uPoints[0].save()
}
```

Slika 3. Funkcija addUserPointsAfterTime

Polja „notificatiosn“ i „chatNotifications“ sadrže informacije o notifikacijama korisnika. Tip notifikacije označava o kakvom se tipu notifikacije radi. Razgovorne notifikacije su razdvojene od običnih notifikacije kako bi se olakšalo slanje notifikacija krajnjem korisniku. Kada osoba dobije notifikaciju da se nekome svidjela, to joj se mora odmah prikazati, ali ako osoba dobije poruku od korisnika pa zatim u kraćem vremenu još jednu, a na nju nije odgovorila, nepotrebno joj je opet slati notifikaciju o tome da joj je poslana nova poruka. Polja „like“ i „dislike“ sadrže ID -ove osoba koji se sviđaju i ne sviđaju korisniku. Korisnik s osobama koje su mu se svidjele može započeti razgovor ako se i on njim sviđa. Osobe kojima se ID-ovi nalaze u polju

„dislike“ se ne prikazuju korisniku na mapi i one ne mogu s korisnikom započeti razgovor.

4.3 Sustav za autentifikaciju

Sustav za autentifikaciju je namijenjen autentifikaciji i autorizaciji korisnika. Koristi RESTful način upravljanja podatcima. Ima tri glavne rute koje se nalaze na „/api/v1/auth “ mapi. Sve funkcije se nalaze u kontroler modulu authController.mjs kako bi se lakše odvojile zadaće svake funkcionalnosti. Sljedećom slikom pod rednim brojem 4 prikazane su rute sustava za autentifikaciju.

```
export const router = Router();

router.post('/login',loginUser);
router.get('/getAccess',checkForToken,getAccess);
router.get('/checkToken',checkForToken,checkAccessToken);
```

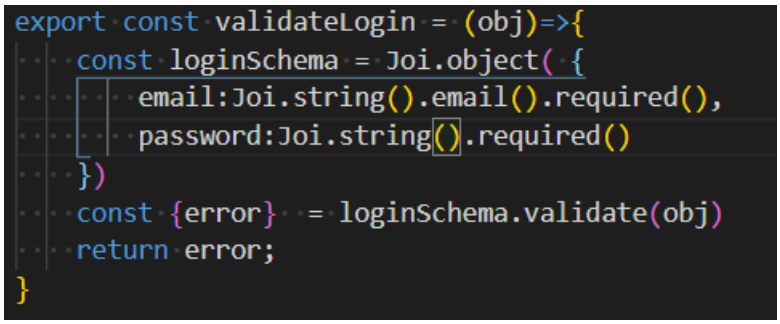
Slika 4. Rute sustava za autentifikaciju

Kada se napravi određeni HTTP zahtjev, onda se na određenoj ruti poziva određena funkcija. Pomoću „app.mjs“ datoteke se povezuju s MongoDB bazom podataka. Konfiguracijski podatci se nalaze u „db.mjs“ i config.mjs“ datotekama.

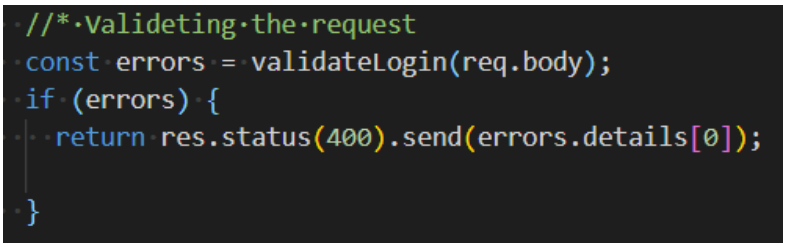
4.3.1 Ruta login

Kada se napravi POST HTTP zahtjev na rutu /login, izvršava se funkcija „loginUser“ koja najprije validira tijelo zahtjeva pomoću validator funkcije koja se nalazi u datoteci „authValidators.mjs“. Ako validacija ne prođe, vraća „error“ kao odgovor na zahtjev. U tijelo zahtjeva potrebno je poslati email i lozinku. Pomoću „Joi.js“ biblioteke

generiraju se automatske poruke koje govore zašto je tijelo zahtjeva neispravno kao što je vidljivo iz sljedećih prikazanih slika pod rednim brojevima 5 i 6.



Slika 5. Funkcija za validaciju tijela zahtjeva za prijavu



Slika 6. Poziv funkcije za validaciju

Poslije se pomoću mongoose.js sheme „User“ provjerava da li postoji korisnik s email adresom navedenom u tijelu zahtjeva. Ako email postoji provjerava se da li spremljeni „hash“ lozinke odgovara lozinci unesenoj u tijelo zahtjeva pomoću „bcrypt“ biblioteke. Ako odgovara, vraćaju se dva JWT tokena. Ako lozinka ili email nisu valjani,

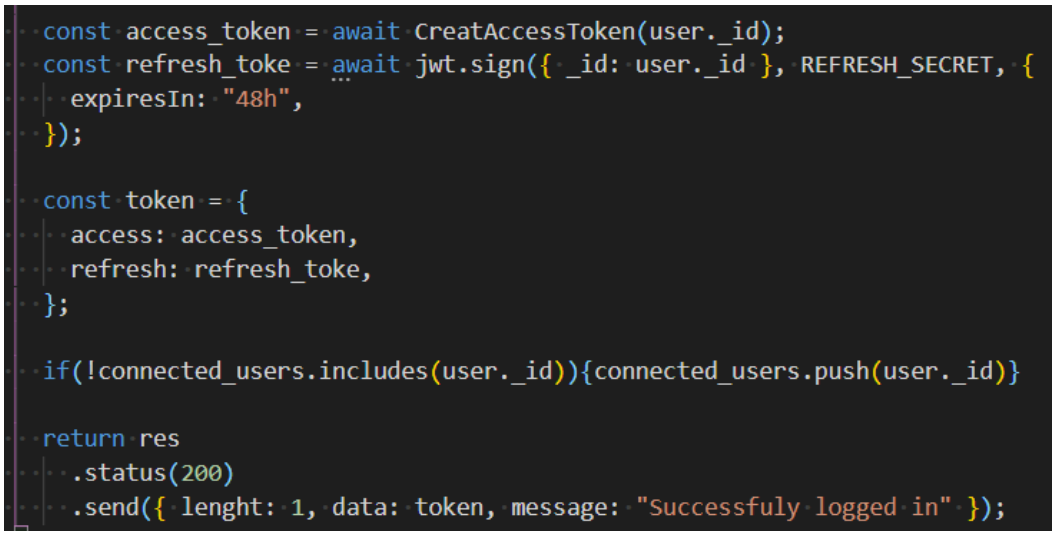
vraćaju se odgovarajuće poruke. U sljedećoj slici pod rednim brojem 7 prikazan je kod kojim su omogućene gore navedene funkcionalnosti.

```
//* Check if user exists
const { email, password } = req.body;
const user = await User.findOne({ email });

if (user == null) {
    return res.status(400).send({message:`User with email: ${email} doesn't exits`})

}

//* Check Password
const checkPassword = await bcrpyt.compare(password, user.password);

if (!checkPassword) {
    return res.status(401).send({message:`Invalid password for User with ${email}`})
}
```

Slika 7. Provjera email-a i lozinke na zahtjev za prijavu

Jedan JWT token predstavlja dopuštenje za mijenjanje i dohvat podataka i prestaje vrijediti nakon kraćeg vremena (15 min). Drugi je takozvani „refresh token“ koji ima puno dulje vrijeme postojanja (48h) i pomoću kojega se može ponovno dohvatiti takozvani „access token“ s kojim se mogu mijenjati podatci u bazi. U oba slučajeva se „hash“-ira ID korisnika kao vrijednost za hash, ali koriste se različiti tajni ključevi za enkripciju. Potom se „refresh token“ sprema u polje trenutnih aktivnih

korisnika i vraćaju se ta dva tokena kao odgovor na zahtjev. U sljedećoj slici pod rednim brojem 8 se pokazuje kako se izvode gore navedene funkcionalnosti.

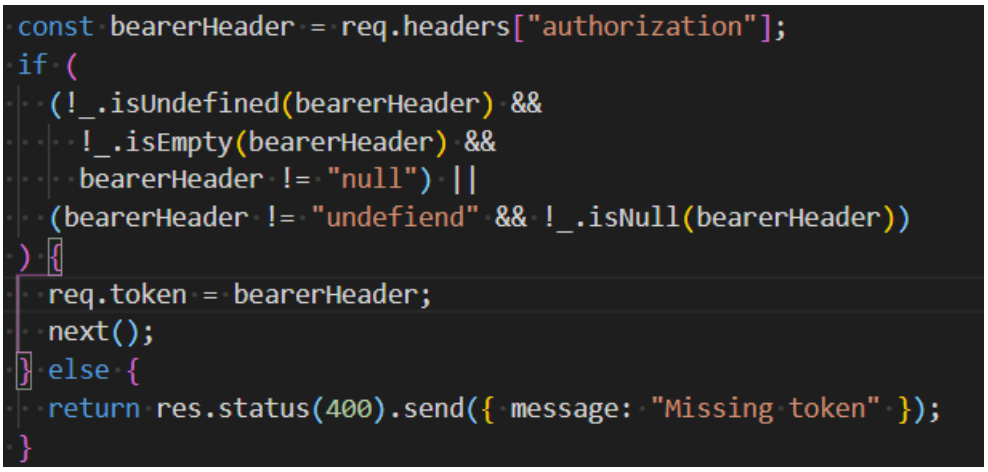


```
const access_token = await CreatAccessToken(user._id);
const refresh_toke = await jwt.sign({ _id: user._id }, REFRESH_SECRET, {
  expiresIn: "48h",
});

const token = {
  access: access_token,
  refresh: refresh_toke,
};

if(!connected_users.includes(user._id)){connected_users.push(user._id)}

return res
  .status(200)
  .send({ lenght: 1, data: token, message: "Successfuly logged in" });
```

Slika 8. Izrada Access i refresh tokena kao odgovor na zahtjev za prijavu

4.3.2 Ruta getAccess

Ova ruta zahtjeva u zaglavlju traži „authorization“ element koji predstavlja „refresh token“. Ako je token validan i ako je korisnikov ID trenutno spremljen u polje „connected_user“ vraća se novi „access token“. Pomoću međusoftvera „checkForToken“ provjerava se da li zaglavlje ima „authorization“. Ako zahtjev nije pravilan ili token je neispravan, vraća se odgovarajuća error poruka na zahtjev. Sljedeća slika pod rednim brojem 9 prikazuje kako se dohvaća i provjerava „authorization“ element zaglavalja.



```
const bearerHeader = req.headers["authorization"];
if (
  (!_.isUndefined(bearerHeader) &&
    !_.isEmpty(bearerHeader) &&
    bearerHeader != "null") ||
  (bearerHeader != "undefiend" && !_.isNull(bearerHeader))
) {
  req.token = bearerHeader;
  next();
} else {
  return res.status(400).send({ message: "Missing token" });
}
```

Slika 9. Pregled provjere zaglavlja zahtjeva

4.3.3 Ruta checkAccessToken

Ova ruta zahtjeva u svojem zaglavlju „authorization“ element koji predstavlja „access token“. Ako je token validan i ako je ID korisnika u „connected_users“ kao odgovor se vraća ID korisnika. Ova ruta također koristi checkForToken među softver.

4.4 Sustav za upravljanje korisničkim podatcima

Ovaj sustav je namijenjen za upravljanje svim korisničkim podatcima pomoću mongoose shemi. Ovo je glavni dio poslužiteljskog sustava te on izvršava najviše usluga na klijentske zahtjeve. Koristiti sustav za autentifikaciju kako bi autentificirao dolazeće zahtjeve klijenta. Ima nekoliko ruta koje su sve povezane preko app.mjs datoteke tako da ona prosljeđuje zahtjeve do odgovarajuće rute. Svaka ruta ima svoje kontrolere. Sljedeća slika pod rednim brojem 10 prikazuje rutere sustava za upravljanje korisničkim podatcima.

```
//* ROUTES
app.use('/api/v1',authRouter) //Authentification
app.use('/api/v1',userRouter) //User CRUD
app.use('/api/v1',mapRouter) //Map router
app.use('/api/v1',chatRouter)// chat router
app.use('/api/v1',userPointsRouter)//points
```

Slika 10. Ruteri sustava za upravljanje korisničkim podatcima.

4.4.1 Ruter za autentifikaciju korisnika

„AuthRouter“ se sastoji od dvije rute koje omogućavaju prijavu i token za pristup podatcima („access token“). Funkcije koje se izvršavaju na određeni zahtjev nalaze se u „authController.mjs“ datoteci. Sljedeća slika pod rednim brojem 11 prikazuje rute „rutera“ za autentifikaciju korisnika.

```
export const router = Router();

router.post('/login',loginResponse);
router.get('/getAccess',accessResponse);
```

Slika 11. Rute za autentifikaciju

- HTTP POST zahtjev „/login“ traži da u tijelu zahtjeva postoji email i lozinka te zahtjev prosljeđuje sustavu za autentifikaciju. Sljedeća slika pod rednim brojem 12 prikazuje kod funkciju „loginResponse“.

```
export const loginResponse = async (req,res,next)=>{
    try {
        const newUrl = findServerUrl(req,'login','auth/login')
        const responce = await axios.post(newUrl,req.body,{
            headers:req.headers,
        })

        return res.status(200).send(responce.data)

    } catch (error) {
        return res.status(error.response.status).send(error.response.data)
    }

}
```

Slika 12. LoginResponse funkcija

- HTTP GET zahtjev „/getAccess“ traži da u zaglavlju zahtjeva postoji „authorization“ ključ koji sadrži token za ponovno dobivanje tokena za pristup podatcima („refresh token“). Zahtjev se prosljeđuje sustavu za

autentifikaciju. Sljedeća slika pod rednim brojem 13 prikazuje funkciju „accessResponse.

```
export const accessResponse = async(req,res,next)=>{
  try {

    const newUrl = findServerUrl(req,'getAccess','auth/getAccess')
    const response = await axios.get(newUrl,{
      headers:{
        'authorization':req.headers['authorization']
      }
    })
    return res.status(200).send(response.data)
  } catch (error) {
    return res.status(error.response.status).send(error.response.data)

  }
}
```

Slika 13. AccessResponce funkcija

Ako u oba zahtjeva dođe do ikakvih pogreški, kod zahtjeva se vraćaju odgovarajuće „error“ poruke od sustava za autentifikaciju koje se kasnije prosljeđuju kao odgovor na zahtjev.

4.4.2 Ruter za korisničke Podatke

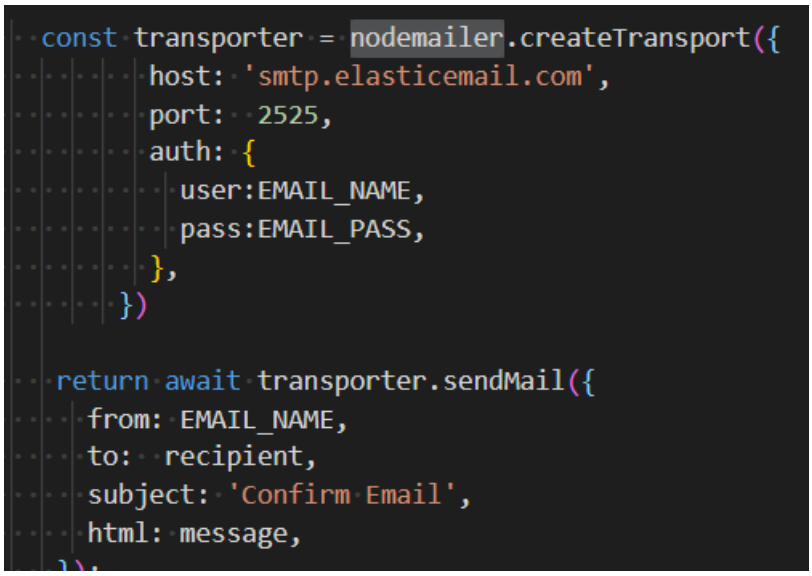
Ovaj ruter obavlja sve potrebne funkcije kod stvaranja, promjene, brisanja i dohvaćanja korisničkih podataka. Pritom ima i dodatne funkcionalnosti kao što su verifikacija unesene email adrese, promjena zaporke te mogućnost zaboravljanja zaporke. Sve funkcije potrebne za izvršavanje funkcionalnosti se nalaze u „userController.mjs“ datoteci. Sljedeća slika pod rednim brojem 14 prikazuje rute „rutera“ za korisničke podatke.

```
router.post('/user',createUser)

//READ
router.get('/user/:id',checkIDParams,userUrl,checkAccess,readUser)

//UPDATE
router.patch('/user',userUrl,checkAccess,paramEQuserId,setActive,updateUser)

//DELETE
router.delete('/user/:id',checkIDParams,userUrl,checkAccess,deleteUser)


//Personal Page Data
router.get('/user',userUrl,checkAccess,paramEQuserId,setActive,readUser)

//Set user location
router.patch('/userLocation',userLocationUrl,checkAccess,setUserLocation)

//Verfiy Email
router.get('/user/verifyEmail/:token',verifyUserEmail)


//FOROGT PASSWORD
router.post('/user/forgotPassword',userFogotPasswordUrl,forgotPassword)

//UPDATE PASSWORD
router.patch('/user/changePassword',userChangePasswordUrl,checkAccess,changePassword)
```

Slika 14. Rute unutar userRouter-a

- HTTP POST zahtjev „/user“ funkcionira tako da se provjeri da li tijelo zahtjeva ima potrebne podatke. Potom se provjerava da li je unesena email adresa već bila u upotrebi. Nakon toga se „hash“-ira zaporka pomoću biblioteke bcrypt, takva se stavlja kao lozinka korisnika. To je mjera opreza kojom se osigurava da se u slučaju krađe podataka iz baze podataka ne može doći do lozinke korisnika. Na kraju se šalje email poruka na adresu novog korisnika kako bi se provjerilo da ta email adresa postoji i da joj novi

korisnik ima pristup. Email se šalje pomoću „nodemailer“ biblioteke. Dolje prikazana slika pod rednim brojem 15 prikazuje postupak slanja email-a.



Slika 15. Izrada i slanje email pomoću nodemailer biblioteke

- HTTP PATCH zahtjev „/user“ “ funkcionira na sličan način kao i POST zahtjev samo što se na drugačiji način provjerava tijelo zahtjeva (nisu svi elementi u tijelu potrebni). Kako bi se moglo izvršiti ažuriranje korisničkih podataka, potrebno je provjeriti prije svega da li zahtjev ima potrebnu autorizaciju. To se radi pomoću sustava za autentifikaciju koji u zaglavlju zahtjeva traži „access“ token pod ključem „authorization“. Nakon toga se zahtjev prosljeđuje sljedećoj funkciji u kojoj se u zaglavlje zahtjeva stavlja

korisnički ID. Sljedeća slika pod rednim brojem 16 prikazuje gore navedeni proces u funkciji „checkAccess“ .

```
const access = req.headers["authorization"];

if (
  !access ||
  _.isUndefined(access) ||
  _.isNull(access) ||
  access == "null" ||
  access == "undefiend"
) {
  return res.status(401).send({ message: "Missing User Token" });
}

const response = await axios.get(req.newUrl, {
  headers: {
    authorization: access,
  },
});

req.user_id = response.data;

next();
} catch (error) {
return res.status(error.response.status).send(error.response.data);
}
```

Slika 16. checkAccess funkcija

Pomoću korisničkog ID iz zaglavlja pronalazi se korisnik kojemu se podatci ažuriraju s onima iz tijela zahtjeva. Na ovaj način se ne može promijeniti email adresa i lozinka korisnika. Kao odgovor na zahtjev šalje se novi korisnički objekt s ažuriranim podatcima. Sljedeća slika pod rednim brojem 17 prikazuje ažuriranja korisničkih podataka.

```
const user = await User.findOneAndUpdate({ _id: id }, update_data, {
  new: true,
});
```

Slika 17. Ažuriranje korisničkih podataka

- HTTP GET „/user/:id“ zahtjev vraća javno dostupne podatke o korisniku s ID navedenom u ruti zahtjeva.
- HTTP DELETE „/user/:id“ zahtjev traži „access“ token u zaglavlju kao i PATCH zahtjev te briše korisnika iz baze podatka ako ID korisnika iz „access“ tokena odgovora ID iz rute zahtjeva. Vraća odgovor o uspješnosti izvršavanja brisanja korisnika.
- HTTP GET „/user“ zahtjev vraća privatne informacije o korisniku. Iz zaglavlja se uzima ID korisnika pomoću „checkAccess“ funkcije (ona provjerava da li je „access“ token validan i stavlja ID iz njega u zaglavlje zahtjeva).
- HTTP GET „/user/verifyEmail/:token“ zahtjev uzima token iz rute te potvrđuje da je korisnik verificirao svoj email. Kada se dekriptira token, dobiva se ID korisnika tako se zna o kojem se korisniku radi. U email-u koji je korisnik dobio pri registraciji nalazi se hiperlink s ovom rutom te na klik ga vodi na tu rutu. Kada zahtjev prođe korisnik se preusmjeri na početnu stranicu web stranice. Sljedeća slika pod rednim brojem 18 prikazuje stvaranja novih životnih bodova korisnika i preusmjeravanja korisnika na lokaciju klijentskog dijela aplikacije.

```
    if(user.email_verified){
        return res.redirect(FRONT_LOCATION);
    }

    user.email_verified = true;
    await user.save();

    //CREATING USER POINTS
    const uPoints = await new UserPoints({ user_id: _id });
    await uPoints.save();

    return res.redirect(FRONT_LOCATION);
} catch (error) {
    res.status(400).send(error.message);
}
```

Slika 18. Prikaz preusmjeravnja validateEmail funkcije

- HTTP POST zahtjev '/user/forgotPassword' traži u tijelu zahtjeva email na koji se korisniku šalje nova generirana lozinka.
- HTTP PATCH zahtjev '/user/changePassword' traži u tijelu novu i staru lozinku te pomoću „checkAccess“ funkcije provjerava ID korisnika. Ako stare „hash“ lozinke odgovaraju „hash“- u lozinke u bazi podataka onda se „hash“-ira nova lozinka i stavlja na mjesto lozinke u bazi podataka.

4.4.3 Ruter za Korisničke Bodove

Ovaj ruter obavlja se potrebne funkcije za sviđa mi se, ne sviđa mi se, korisničke bodove, provjeru međusobno sviđanja i slanje notifikacija korisniku. Sve funkcije potrebne za izvršavanje funkcionalnosti se nalaze u „userPointsController.mjs“ datoteci. Sljedeća slika pdo rednim brojem 19 prikazuje ruti „rutera„ UserPoints“.

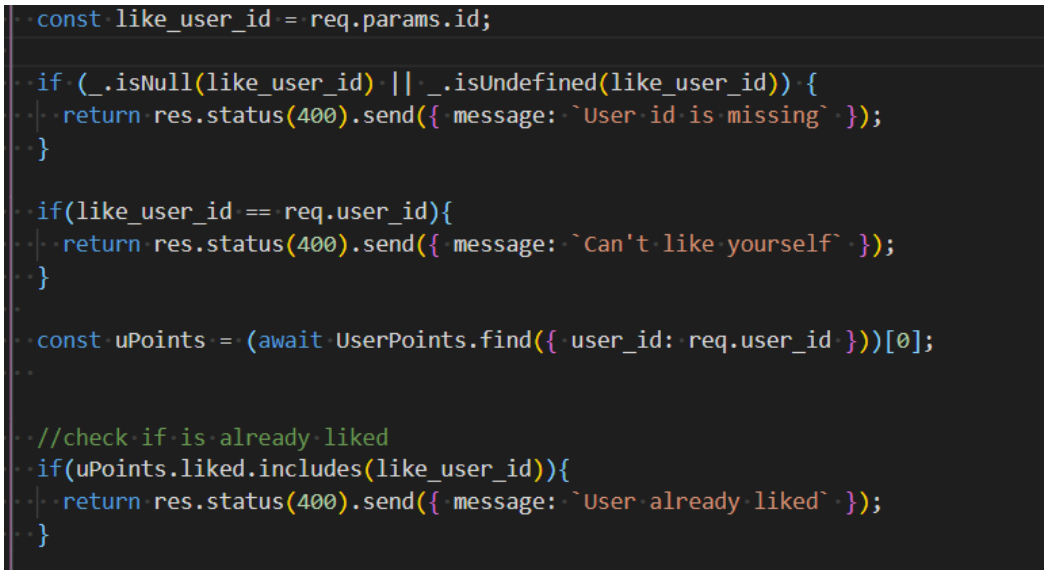
```
router.get('/userPoints',userPointsUrl,checkAccess,getUserPoints)

router.patch('/userPoints/decrease',userPointsDecreaseUrl,checkAccess,decreaseUserPoints)
router.patch('/userPoints/notification',userPointsaddNotificationUrl,checkAccess,addNotifications)

router.get('/userPoints/notification/:id',userPointsaddNotificationUrl,checkAccess,notificationSeen)
router.get('/userPoints/chatNotification/:id',userPointsaddChatNotificationUrl,checkAccess,chatNotificationSeen)

router.get('/userPoints/like/:id',userPointsLikeUrl,checkAccess,likeUser)
router.get('/userPoints/dislike/:id',userPointsDislikeUrl,checkAccess,dislikeUser)


router.get('/userPoints/match/:id',userPointsMatchUrl,checkAccess,matchedUsers)

router.delete('/userPoints/notification/:id',userPointsaddNotificationUrl,checkAccess,deleteNotification)
router.delete('/userPoints/chatNotification/:id',userPointsaddChatNotificationUrl,checkAccess,deleteChatNotification)
```

Slika 19. Rute userPoints routera

- HTTP GET „/userPoints“ zahtjev vraća podatke definirane u userPoints shemi. Pomoću „checkAccess“ funkcije se pronalazi ID korisnika čiji se podatci trebaju vratiti.
- HTTP PATCH „/userPoints/notification“ zahtjev dodaje novu notifikaciju u polje notifikacija. U tijelu zahtjeva nalaze se podatci potrebni za stvaranje nove notifikacije. Polje notifikacije se provjerava da već prije ne postoji ista notifikacija. Kao odgovor vraća ažurirane korisničke podatke tipa userPoints.

- HTTP GET „/userPoints/notification/:id“ zahtjev predstavlja da je korisnik vidio tu notifikaciju. Ovim načinom se može provjeriti da li korisnik ima potreban „access“ token i da li se ta notifikacija nalazi u njegovom polju notifikacija. Ako se nalazi, onda se toj notifikaciji ažurira podatak seen (viđeno na hrvatskom) na istinitu vrijednost (true) te je time označeno da je ta notifikacija viđena. Kao odgovor vraća se potvrda o uspješnosti ažuriranja modela userPoints.
- HTTP GET „/userPoints/ chatNotification/:id“ zahtjev predstavlja da je korisnik vidio notifikaciju u novo zaprimljenoj poruci od drugog korisnika. Izvršava se na vrlo sličan način kao i prijašnji zahtjev HTTP GET „/userPoints/notification/:id“.
- HTTP GET „/userPoints/like/:id“ zahtjev predstavlja da se korisniku svidio drugi korisnik. Preko checkAccess funkcije se provjerava ID korisnika koji radi zahtjev te s ID-em rute utvrđuje se koji korisnik mu se svidio. Utvrđuje se da li se ID osobe nalazi u „like“ polju ili je li ID osobe koje mu se sviđa isti njegovom ID-u te se vraća povratna poruka o toj pogrešci. Ako je korisnik prije napravio zahtjev da mu se ta osoba ne sviđa, ID te osobe se premješta iz polja korisnika koji se ne sviđaju korisniku te se umeće u polje osoba koji se sviđaju korisniku. Potom se korisniku smanjuju životni bodovi za jedan, a osobi koja se sviđa korisniku se pronalazi njegov userPoints model te mu se dodaje nova notifikacija o tome da se tom korisniku svidio. Ako korisnik nema dovoljno životnih bodova, zahtjev mu se odbija s porukom o tome da nema dovoljno životnih bodova. Kao odgovor na zahtjev korisniku se

vraćaju novi ažurirani podatci iz njegovog userPoints modela. Sljedeće slike pod rednim brojem 20 i 21 prikazuje provjeru validnosti zahtjeva.



```
const like_user_id = req.params.id;

if (_.isNull(like_user_id) || _.isUndefined(like_user_id)) {
    return res.status(400).send({ message: `User id is missing` });
}

if(like_user_id == req.user_id){
    return res.status(400).send({ message: `Can't like yourself` });
}

const uPoints = (await UserPoints.find({ user_id: req.user_id }))[0];


//check if is already liked
if(uPoints.liked.includes(like_user_id)){
    return res.status(400).send({ message: `User already liked` });
}
```

Slika 20. Provjere validnosti zahtjeva /userPoints/like/:id

```
if(uPoints.liked.includes(like_user_id)){
    return res.status(400).send({ message: `User already liked` });
}

//check if is in disliked
if(uPoints.dislike.includes(like_user_id)){
    uPoints.dislike = uPoints.dislike.filter(i=> i!=like_user_id)
}

uPoints.liked.push(like_user_id)
await uPoints.save();

//push notification
const userthatLiked = await User.findById(req.user_id)

const notif={
    type:'1',
    senderId:req.user_id,
    receiverId:like_user_id,
    text: userthatLiked.firstName +" "+ userthatLiked.lastName + " has liked you"
}

const reciverUPoints = (await UserPoints.find({ user_id: like_user_id }))[0];

const new_notif = [...reciverUPoints.notifications, notif];
reciverUPoints.notifications = new_notif;


//Decrease User Points
const { error, lifes } = await decresePoints(res, req.user_id);
if (error) {
    return res.status(400).send("Not enough points");
}
```

Slika 21. Ažuriranje polja „like“ i stvaranje nove notifikacije zahtjeva /userPoints/like/:id

- HTTP GET „/userPoints/dislike/:id“ zahtjev je gotovo identičan prijašnjem HTTP GET „/userPoints/like/:id“. Razlika je u tome da se u korisnikovo polje „dislike“ stavlja ID korisnika iz rute te ako je taj ID u korisnikovo polju „like“ onda se pobriše. Pri tom se ne šalje notifikacija korisniku koji se ne sviđa pošiljatelju zahtjev.
- HTTP GET „/userPoints/match/:id“ zahtjev provjerava da li se korisnik s ID-em iz rute i korisnik koji šalje zahtjev međusobno sviđaju. Zahtjev vraća istinitost te provjere (true ili false).

- HTTP DELETE „/userPoints/notification/:id“ i HTTP DELETE „/userPoints/chatNotification/:id“ zahtjevi brišu iz userPoints modela korisnika notifikacije s ID iz rute zahtjeva.

4.4.4 Ruter za prikaz korisnika na Mapi

Ova ruta je zadužena za vraćanje polja korisnika u određenom radijusu od korisnika koji šalje zahtjev. Kako bi to se to moglo izračunati koristi se podatak „userLocation“ iz „User“ modela. Funkcije zadužene za izvršavanje tih zadataka se nalaze u datoteci „mapController.mjs“. Sljedeća slika pod rednim brojem 22 prikazuje rute rutera „mapRouter“.

```
router.get("/map",mapUrl,checkAccess,usersOnMap)
router.post("/map",mapUrl,checkAccess,filterUsersOnMap)
router.get("/findSomeone",findSomeoneUrl,checkAccess,findSomeOne)
```

Slika 22. Rute mapRouter-a

- HTTP GET „/map“ zahtjev koristi query upite na rutu kako bi uzeo radijus u kojem je potrebno pronaći korisnike do korisnika koji šalje zahtjev. U query zahtjev je potrebno definirati parametar range koji predstavlja radijus. Nakon toga se definira udaljenost od svih korisnika u bazi od korisnika koji šalje zahtjev te se vraćaju samo oni koji nisu u korisnikovom polju „dislike“ ili im je udaljenost veća od parametra range. Pomoću korisnikovog „userPoints“ modela se pronalazi se korisnikovo polje „dislike“. Sljedeće slike pod rednim brojevima 23 i 24 prikazuju gore navedene funkcionalnosti.

```
    const { range } = req.query;
    const query = User.find(); // ADD FILLTERS

    const user = await User.findById({ _id: req.user_id });

    //TODO FIX RANGE


    const users = await User.find({}) //all
```

Slika 23. Prikaz uzimanje range parametra iz URL-a

```
const uPoints = (await UserPoints.find({user_id:user._id}))[0]


const results_with_distance = calculateDistance(
  user.lastKnownLocation.coordinates,
  users
).filter(i=> i.distance <= range).filter(el => ! uPoints.dislike.includes(el._id));
```

Slika 24. Prikaz pronalaženja korisnika

- HTTP POST „/map“ zahtjev filtrira korisnike po tijelu zahtjeva. U tijelu zahtjeva se nalazi radijus od korisnika u kojem se traže korisnici te njihove osobine poput dob, spol i/ili interesi. Po tim karakteristikama se filtriraju korisnici i vraćaju samo oni koji prolaze uvijete. Filtriranje po interesima funkcionira tako da se provjerava je li korisnik ima barem jedan interes u polju traženih interesa. Sljedeće slike pod rednim brojevima 25, 26 i 27 prikazuju funkciju za bodovanje korisnika.

```
const err = validateFilterBody(req.body);
if (err) {
  return res.status(400).send({ message: `Invalid request ${err}` });
}
//helper

let helper = [];
if (!_.isUndefined(req.body.age)) {
  helper.push({ age: { $gte: req.body.age.min, $lte: req.body.age.max } });
}

if (!_.isUndefined(req.body.sex)) {
  let or = [];
  if (req.body.sex == 0) {
    or.push({ gender: "male" });
  } else if (req.body.sex == 1) {
    or.push({ gender: "female" });
  } else {
    or.push({ gender: "male" });
    or.push({ gender: "female" });
    or.push({ gender: "other" });
  }
  helper.push({ $or: or });
}
```

Slika 25. Funkcija za bodovanje korisnika 1. dio

```
    if (!_.isUndefined(req.body.interests)) {
      let arr = req.body.interests.map((i) => i.interest);

      helper.push({ "interests.interest": { $in: [...arr] } });
    }

    const query = User.find({
      $and: helper,
    });
    query.setOptions({ lean: true });
    query.collection(User.collection);
    const user = await User.findById({ _id: req.user_id });

    const { range } = req.body;

    const result = await query.exec() //all
```

Slika 26. Funkcija za bodovanje korisnika 2. dio

```
  const uPoints = (await UserPoints.find({user_id:user._id}))[0]

  const results_with_distance = calculateDistance(
    user.lastKnownLocation.coordinates,
    result,true
  ).filter(i=> i.distance <= range) .filter(el => ! uPoints.dislike.includes(el._id));
  return res
    .status(200)
    .send(results_with_distance.filter((i) => i._id != req.user_id));
} catch (error) {
  console.log(error);
  res.status(400).send(error);
}
```

Slika 27. Funkcija za bodovanje korisnika 3. dio

- HTTP GET „/findSomeone“ zahtjev pronalazi korisnike koji najbolje odgovaraju korisniku. To radi tako da pomoću parametara: dob, spol, seksualne orijentacije i polja interesa boduje sličnosti tih parametara

korisnika koji šalje zahtjev i ostalih korisnika. Vraća prvih 10 rezultata sortiranih od najvećih prema najmanjim. Zanemaruje sve rezultate gdje je bodovna sličnost ispod pet posto i sve korisnike koje je korisnik koji šalje zahtjev stavio u svoje polje „dislike“. Boduje se tako da se za godine gleda što je udaljenost godina veća to su manji bodovi. Tako na primjer, ako je razlika 0 onda se dodjeljuju maksimalni bodovi, dok ako je razlika preko 15, ne dodjeljuju se bodovi. Za spol i seksualnu orijentaciju provjerava se da li su obje osobe heteroseksualne, homoseksualne ili biseksualne te dodjeljuje bodove po kompatibilnosti orijentacije i spola. Ako je npr. osoba heteroseksualna i žena, drugi korisnik muškarac i biseksualan dobit će se dio bodova, ali homoseksualna osoba koja ima spol „other“ i druga osoba homoseksualna koja isto ima spol „other“ dobit će sve bodove. Za interese se boduje tako da se gledaju kategorije interesa i da li osobe imaju preklapanja u interesima, te ako oboje osobe imaju interese istih kategorija, ali nemaju preklapanja, dodaju se djelomični bodovi po broju interesa u istim kategorijama osoba. Na kraju se sve to skalira na jedan odnosno na sto posto. Radijus pretraživanja je 1000km.

4.4.5 Ruter razgovora između korisnika

Ovaj ruter služi za spremanje poruka između korisnika te slanje notifikacija o novim poruka korisnicima koji su zaprimili poruku te blokiranje neželjenih razgovora. Funkcije zadužene za izvršavanje ovih zadaća se nalaze u datoteci „chatController.mjs“ Sljedeća slika pod rednim brojem 28 prikazuje rute rutera „chatRouter“.

```
router.get("/chat",chatUrl,checkAccess,checkChatHeader,getChatThread)
router.post("/chat",chatUrl,checkAccess,createThread)
router.patch("/chat",chatUrl,checkAccess,saveMessage)
router.patch("/blockChat",blockChatUrl,checkAccess,blockChat)
router.get("/userChats",userChatUrl,checkAccess,getUserChats)
```

Slika 28. Rute routera chatRouter

- HTTP GET „/chat“ zahtjev vraća prema ID iz zaglavlja razgovor tipa ChatThread kao odgovor na zahtjev. Vraćaju se samo zadnjih 25 poruka.
- HTTP POST „/chat“ zahtjev stvara novi razgovor ako se ustvrdi razgovor ne postoji. Ako razgovor postoji, vraća se na njega kao dogovor na zahtjev, u

protivnom stvara se novi. Kako bi se zahtjev pozitivno izvršio, potrebno je u tijelu poslati ID korisnika s kojim se stvara razgovor. Kako bi razgovor mogao započeti, potrebno je međusobno sviđanje korisnika. Ovaj zahtjev smanjuje broj životnih bodova korisnika koji šalje zahtjev, te ako korisnik nema dovoljno bodova, zahtjev se odbija i vraća se odgovarajuća poruka. Šalje se obavijest u bazu korisniku s kojim je započet razgovor. Razgovor se ne može započeti sa samim sobom. Kao odgovor na zahtjev šalje se broj životnih bodova, vrijeme kada će dobiti još životnih bodova te ID razgovora.

- HTTP PATCH „/chat“ služi za spremanje novih poruka u bazu. Kako bi se zahtjev mogao izvršiti, potrebno je u tijelo zahtjeva poslati ID razgovora, ID osobe koja šalje poruku te sadržaj poruke. Ako je bilo koja osoba blokirala razgovor, zahtjev se odbija s odgovarajućom porukom. U polje poruka dodaje se nova poruka s vremenom kada je poruka poslana, te se šalje obavijest o dolasku nove poruke korisniku koji je dobio poruku, ako je vrijeme od zadnje dobivene poruke u ovom razgovoru manje od jednog sata.
- HTTP PATCH „/blockChat“ zahtjev blokira ili deblokira razgovor prema vrijednostima poslanim u tijelu zahtjeva. U tijelu zahtjeva treba se nalaziti ID razgovora te „blockChat“ vrijednost koja može biti istinita ili lažna. Prema tome se odlučuje da li se razgovor treba deblokirati ili blokirati. Ako se razgovor opet želi blokirati ili deblokirati, zahtjev se odbija i šalje se odgovarajuća poruka. Ovaj zahtjev oduzima životne bodove korisniku za jedan i ako ih nema dovoljno, zahtjev se odbija s odgovarajućom porukom.
- HTTP GET „/userChats“ zahtjev vraća polje ID-ova svih razgovora gdje se korisnikov ID koristi. Pritom se dodaje ime i prezime svakom korisniku s kojim korisnik ima započet razgovor u to polje.

4.5 Sustav za sinkronu komunikaciju

Sustav za sinkronu komunikaciju omogućava trenutnu komunikaciju između korisnika. Služi za slanje poruka aktivnim korisnicima te notifikacije o porukama ili događajima poput sviđanja korisnika. Sustav se u cijelosti izvršava u datoteci „app.mjs“ pomoću biblioteke „socket.io“.

Kada se korisnik prijavi i omogući mu se pristup glavnoj stranici onda se pomoću „socket.io“ šalje event „connection“ koji označava da se korisnik spoji na „socket“ . Unutar „connection“ funkcije nalaze se ostali „eventhandler“-i (funkcije koje se izvršavaju kada se određeni događaj dogodi) „addUser“, “sendMessage“, „sendNoficiation“ i „disconnect“.

4.5.1 AddUser event

Kada se dogodi event „addUser“, tada se poziva njegova anonimna funkcija koja dodaje novog korisnika u polje trenutno aktivnih korisnika pomoću „addUser“ funkcije. Anonimna funkcija očekuje argument ID korisnika koji se spojio. „addUser“ funkcije provjerava da li je taj korisnik već spremljen u polje trenutno aktivnih korisnika „users“. U polje se sprema objekt s ID korisnika i ID kanala (socket) na kojem se nalazi taj korisnik. Tada se emitira event „getUsers“ u polje trenutno aktivnih korisnika natrag na klijentski dio strance. Sljedeće slike pod rednim brojevima 29 i 30 prikazuju gore navedene funkcionalnosti.

```
//take userId and socketId from user
socket.on("addUser", (userId) => {
  addUser(userId, socket.id);
  io.emit("getUsers", users);
});
```

Slika 29. Prikaz eventa addUser

```
let users = [];
const addUser = (userId, socketId) => {
  !users.some((user) => user.userId === userId) &&
    users.push({ userId, socketId });
};
```

Slika 30. addUser funkcija

4.5.2 SendMessage event

Ovaj event označava da je korisniku pristigla nova poruka. Kada se okine „sendMessage“ event on kao argumente očekuje ID korisnika koji šalje poruku, ID korisniku kojem se šalje poruka, tekst poruke, ID razgovora u kojem se dogodila poruka, te ako je poruka slika, onda putanje do te slike. Pronalazi se korisnik kojemu se šalje poruka iz polja spojenih korisnika. Ako ne postoji, šalje mu se odgovarajuća error poruka. Tom korisniku se na njegov kanal (socket) emitira „getMessage“ event i

„getNotification“ event koji će obraditi klijentski dio aplikacije. Unutar „getMessage“ eventa kao argumente potrebno je poslati ID korisnika koji šalje poruku, tekst poruke, te ako postoji URL slike. „getNotification“ event kao argumente uzima ID korisnika kojemu se šalje poruka, tekst poruke, ID razgovora u kojem se poruka dogodila, te tip notifikacije. Sljedeća slika pod rednim brojem 31 prikazuje event „sendMessage“.

```
socket.on(
  "sendMessage",
  ({ senderId, receiverId, text, chatId, imageUrl }) => {

    const user = getUser(receiverId);

    if (user == undefined) {
      const reportError = getUser(senderId);
      io.to(reportError.socketId).emit("Error", {
        msg: "User disconected",
      });
      return;
    }

    io.to(user.socketId).emit("getMessage", {
      senderId,
      text,
      imageUrl,
    });

    io.to(user.socketId).emit("getNotification", {
      senderId,
      text,
      chatId,
      type: 0,
    });
  }
);
```

Slika 31. SendMessage event

4.5.3 sendNoficiation event

Ovaj event označava da je korisnik dobio novu notifikaciju. „sendNoficiation“ event kao argumente očekuje ID korisnika koji je poslao notifikaciju, ID korisnika koji je dobio notifikaciju, tekst notifikacije te tip notifikacije. Pomoću ID korisnika koji je dobio notifikaciju pronalazi se kanal (socket) na kojem je on spojen. Ako ne postoji korisnik s tim ID znači da nije trenutno aktivan te se šalje odgovarajuća error poruka pomoću emitiranja „Error“ eventa koji obrađuje klijentski dio aplikacije. Kanalu (socket - u) na kojem se nalazi korisnik koji je zaprimio notifikaciju emitira se event „getNotification“ s

potrebnim parametrima . Sljedeća slika pod rednim brojem 32 prikazuje event „sendNotification“.

```
socket.on("sendNoficiation", ({ senderId, receiverId, text, type }) => {
  const user = getUser(receiverId);
  if (user == undefined) {
    const reportError = getUser(senderId);
    io.to(reportError.socketId).emit("Error", {
      msg: "Korisnik nije na vezi",
    });
    return;
  }
  io.to(user.socketId).emit("getNotification", {
    senderId,
    text,
    type,
  });
});
```

Slika 32. SendNotification event

4.5.4 disconnect event

Ovaj event označava da korisnik nije trenutno aktivan na klijentskoj strani aplikacije te ga miče iz polja trenutno aktivnih korisnika. Nakon što se njegov kanal (socket) zatvorio, klijentskom dijelu stranice šalje se event „getUsers“ s novim poljem trenutno aktivnih korisnika. Sljedeća slika pod rednim brojem 33 prikazuje „disconnect“ event.

```
//when disconnect
socket.on("disconnect", () => {
  console.log("a user disconnected!");
  removeUser(socket.id);
  io.emit("getUsers", users);
});
```

Slika 33. disconnect event

5. Zaključak

U ovom radu predstavljena je aplikacija koja rješava problem ljudske bliskosti i stvaranja novih prijateljstava u vremenu sve veće ne zainteresiranosti za bližnje. U već gotovim rješenjima iskrenost i trajnost odnosa između korisnika je naknadna ideja koje se ni ne pokušava realizirati, već se pokušava korisnika zavarati na trošenje monetarnih dobara kako bi pronašao osobu s kojom će se zbližiti. Nažalost, takvim aplikacijama se koristi da korisnici ne pronađu osobu koja ima odgovara, jer korisnici će prestati koristiti tu aplikaciju u vremenu kada se s nekim zbliže. Ako uzmemo u obzir koliko korisnika ostaje sa veoma negativnim iskustvom poslije korištenja takvih aplikacija, vidimo da postoji potreba za aplikacijama opisanim u ovom radu. Zbog toga je kroz ovu aplikaciju stavljen fokus na prijateljske i ljubavne odnose.

Aplikacija za stvaranje novih prijateljskih i ljubavnih veza omogućava korisnicama fokusirani pristup na korisnika s kojim pričaju. Također, s mogućnosti pronalazaka novih ljudi u blizini, stvaranje novih prijateljstava i ljubavnih veza je lakše nego ikada s time da postoje posljedice (gubljenje života), ako se korisnik odluči za blokiranje ili ne sviđanje određenog korisnika. S time se postigla određena kazna za osobe koje samo kao na traci upoznaju nove ljude s kojima gotovo sigurno neće (ne žele) stvoriti istinsku prijateljsku ili ljubavnu vezu.

Korištenjem tehnologija poput Node.js s Express bibliotekom, razvoj samih REST servisa je bilo lak zbog brojne literature i već postojećih gotovih rješenja. Buduću skalabilnost uveliko olakšava korištenje mongoDB kao primarne baze podataka s ODM - om mongoose. Podjelom poslužiteljskim komponenti na tri dijela omogućena je nadogradnja svakog dijela neovisno o drugome te se olakšalo održavanje cjelokupnog sustava.

Kao sljedeći korak nadogradnje ove aplikacije bilo bi sakupljanje podataka o uspješnosti funkcije „findSomeone“. Korištenjem umjetne inteligencije bi se mogao napraviti bolji model za pronalaženje nasumičnih osoba zanimljivih korisniku. Moguće je proširenje sustava razgovora na omogućavanje slanja datoteka i glasovnih poruka.

Literatura

GEOJSON. (2021). *GeoJSON*. Dohvaćeno iz https://geojson.org/: https://geojson.org/ [19. rujna 2021.]

Guo, J. (26. 7 2016). *The Washington Post*. Dohvaćeno iz Why everyone is miserable on Tinder: https://www.washingtonpost.com/news/wonk/wp/2016/07/26/why-everyone-is-miserable-on-tinder/ [19. rujna 2021.]

Herron, D. (2020). *Node.js Web Development - Fifth.*

Holtzhausen, N., K. F., Thakur, I., Ashley, J., Rolfe, M., & Pit, S. W. (2020). *Swipe-based dating applications use and its association with mental health outcomes: a cross-sectional study.*

Humphrey, K. (15. 5 2013). *https://www.startribune.com/*. Dohvaćeno iz Lust at First Photo: https://www.startribune.com/lust-at-first-photo-tinder-heats-up-dating-app-scene/207573481/ [19. rujna 2021.]

Illouz, E. (2010). *Hladne intimnosti: Oblikovanje čustvenega kapitalizma.* Ljubljana.

Lab, T. D. (2021). *The Decision Lab*. Dohvaćeno iz The Paradox of Choice - The Decision Lab: https://thedecisionlab.com/reference-guide/economics/the-paradox-of-choice/ [19. rujna 2021.]

Overflow, S. (2021). *Stack Overflow*. Dohvaćeno iz Stack Overflow Developer Survery 2021: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof [19. rujna 2021.]

Paska, I. (2020.). *Fast Choices and Emancipatory Spaces: Complex Reality of Online dating Apps.* Zagreb,Hrvatska.

Serby, P. (7. Siječanj 2012). *Case study: How & why to build a consumer app with Node.js*. Dohvaćeno iz venturebeat.com: https://venturebeat.com/2012/01/07/building-consumer-apps-with-node/ [19. rujna 2021.]

Solis, R. J., & Wong, K. Y. (2017). *To meet or not to meet? Measuring motivations and risks as predictors of outcomes in the use of mobile dating applications in China.*

Sučić, L. (2021). *Razvoj klijentskih komponenti web aplikacije za stvaranje novih prijateljskih i ljubavnih veza.* Pula: Sveučilište Jurja Dobrile.

Popis slika

SAŽETAK

Ovaj rad opisuje proces izgradnje poslužiteljskih komponenti web aplikacije za stvaranje novih prijateljskih i ljubavnih veza. Opisuje početnu problematiku modernih rješenja za upoznavanje novih prijatelja i sklapanje novih ljubavnih veza, potom opisuje korištene tehnologije i objašnjava kako će biti korištene u unutar samoga projekta. Zatim se radi pregled modela baze podataka i objašnjava što svaki element predstavlja i kako će se koristiti unutar projekta. U arhitekturi opisuje podjelu na tri manja servisa i za što je koji zaslužan. Detaljno opisuje kako se koji od zahtjeva na određeni servis izvršava i što je sve potrebno da bi se taj zahtjev uspješno izvršio, te objašnjava koncepte potrebne za razumijevanja odnosa među servisima i zahtjevima korisnika. Na kraju daje se osvrt na izvršenje ovog rada i moguća buduća proširenja.

Ključne riječi: web aplikacija, Node.js, Express.js, RESTful, JWT, MongoDB, Mongoose, poslužiteljske komponente, Socket.io

ABSTRACT

The contents of this paper cover the process of developing server-side components of a web application for forming new friendships and love relationships. Firstly, it describes the problems of modern existing applications for forming new friendships and love relationships, it then lists the technology that will be used and how it will be used in this project. It further presents models used to describe the data and what each element represents as well as how it's used in the work. Architecture section describes the division of components in three smaller services and the purpose of each service. It explains in detail how is each request to the server handled and all prerequisites that are required to successfully handle the request. It further explains the needed concepts to understand interworking between services and user requests. In the end it gives the summery of what was done and how the application could be upgraded.

Key words: web application, Node.js, Express.js, RESTful, JWT, MongoDB, Mongoose, backend, Socket.io