Razvoj web aplikacije za učinkovito vremensko upravljanje poslovnim sastancima

Zubčić-Đurđević, Anthea

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Prijediplomski studij računarstva

Završni rad

Razvoj web aplikacije za učinkovito vremensko upravljanje poslovnim sastancima

Rijeka, srpanj 2023.

Anthea Zubčić-Đurđević
00690881609

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Prijediplomski studij računarstva

Završni rad

Razvoj web aplikacije za učinkovito vremensko upravljanje poslovnim sastancima

Mentor: doc. dr. sc. Marko Gulić

Rijeka, srpanj 2023.

Anthea Zubčić-Đurđević
00690881609

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 8. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Anthea Zubčić-Đurđević (0069088160)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **Razvoj web aplikacije za učinkovito vremensko upravljanje poslovnim sastancima / Development of a web application for effective time management of business meetings**

Opis zadatka:

Razviti web aplikaciju za vremensko upravljanje poslovnim sastancima na različitim geografskim lokacijama. Aplikacija omogućuje dodavanje sastanaka u poslovni raspored uz automatsko izračunavanje udaljenosti između lokacije trenutnog sastanka i lokacija sastanka radi efikasnijeg upravljanja radnim vremenom. Za razvoj klijentskog dijela aplikacije treba koristiti React java script radni okvir. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Rustov web radni okvir Rocket uz PostgreSQL sustav za upravljanje bazom podataka. Za određivanje udaljenosti između lokacija dvaju sastanaka, potrebno je aplikaciju povezati s Google Maps API-jem.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 26. ožujka 2023.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad.

Rijeka, srpanj 2023.

Anthea Zubčić-Đurđević
Ime Prezime

Sadržaj

Popis slika

Popis slika

Popis tablica

Poglavlje 1

Uvod

U ovom radu je napravljena web aplikacija koja omogućuje dodavanje sastanaka u poslovni raspored, uz automatsko izračunavanje udaljenosti između lokacija sastanka, radi efikasnijeg upravljanja radnim vremenom.

Aplikacija korisnicima olakšava pronalazak pružatelja usluge i dogovaranje sastanaka. Korisnici mogu uvijek vidjeti popis dostupnih pružatelja usluga kao i slobodne termine te u bilo kojem trenutku dogovoriti sastanak. Također, preko aplikacije mogu komunicirati s pružateljima usluga te plaćati sastanke putem *Google Pay* aplikacijskog programskog sučelja (eng. application programming interface, skraćeno API). Pružatelji usluga imaju detaljan uvid u svaki sastanak i dugovanja korisnika te, kao i korisnici, mogu dodavati sastanke. Također, mogu komunicirati s korisnicima i potvrditi plaćene sastanke.

Za razvoj klijentskog dijela aplikacije korišten je React [1] JavaScript radni okvir te knjižnica Semantic UI [2]. React.js je jedna od najpopularnijih JavaScript knjižnica otvorenog koda, a koristi se za brzu i efikasnu izgradnju interaktivnih korisničkih sučelja [8]. Jednostavna je za korištenje, brza je i efikasna te pogodna za kombiniranje s drugim alatima. Semantic UI je moderni klijentski okvir za izgradnju korisničkih sučelja web aplikacija [9]. Uključuje veliki broj gotovih CSS stilova, JavaScript komponenti i smjernica za upotrebu te razne grafičke dizajne koji se mogu koristiti za brzo kreiranje modernih i vizualno privlačnih sučelja.

Za razvoj poslužiteljskog dijela web aplikacije korišten je Rustov [3] web radni okvir Rocket [4], uz PostgreSQL [5] sustav za upravljanje bazom podataka. Rocket je

jedan od najpopularnijih web razvojnih okvira pisanih u programskom jeziku Rust. Koristi se u razvoju brzih, sigurnih, fleksibilnih, jednostavnih i skalabilnih web aplikacija [10]. PostgreSQL je moćan sustav za upravljanje bazom podataka. Podržava relacijske i nerelacijske modele baze podataka i nudi napredne funkcije.

Za kontrolu verzija koda korišten je Git [6], a kao razvojno okruženje Visual Studio Code [7]. Git je distribuirani sustav za praćenje promjena u kodu i upravljanje verzijama programa. Omogućava efikasnu i istovremenu suradnju na projektima, praćenje promjena, poništavanje grešaka i održavanje povijesti koda. Visual Studio Code je integrirano razvojno okruženje (eng. integrated development environment, skraćeno IDE) unutar kojeg se može pisati i uređivati kod u različitim programskim jezicima [11]. Unutar njega se nalazi sve potrebno za izradu projekta.

Kroz ovaj rad predstavljene su tehnologije i alati korišteni prilikom izrade aplikacije, upute za pokretanje projekta na računalu, shema baze podataka, programski kod i njegov repozitorij. Repozitorij projekta se može pronaći na idućoj poveznici: https://github.com/antheazd/scheduleMe.

Poglavlje 2

Korištene tehnologije i alati

2.1 Programski jezici i knjižnice

2.1.1 Rust

Rust [12] je moderan programski jezik čiji razvoj 2006. godine započinje Graydon Hoare, tada 29-godišnji programer tvrtke Mozzila. Jednog dana, kad se Graydon vraćao u svoj stan u Vancouveru, ustanovio je da dizalo nije u funkciji jer se njegov program srušio [13]. Programi su najčešće bili napisani u jezicima C ili C++, koji omogućuju brzo pisanje izvrživog koda, ali olakšavaju i slučajno uvođenje memorijskih grešaka koje će uzrokovati rušenje. Hoare je tad uočio potrebu za razvojem novog programskog jezika koji bi bio sigurniji, efikasniji i moderniji od postojećih. Daljnji razvoj se nastavio u godinama koje su uslijedile, pri čemu su se mnogi programeri i istraživači iz Mozilla Researcha uključili u projekt. Prva verzija je objavljena u siječnju 2010. godine.

Rust danas koristi više od 2.8 milijuna programera te je, prema anketi Stack Overflowa, posljednjih 8 godina najvoljeniji programski jezik na svijetu [14]. U svojem kodu su ga implementirale neke od najvećih kompanija svijeta: Amazon, Microsoft, Discord, Dropbox, Facebook, Mozilla, Cloudflare, Coursera, Figma, npm i druge [15].

2.1.1.1 Rust sanduk (eng. crate)

Rustov sanduk označava najmanju količina koda koju Rustov *compiler* uzima u obzir odjednom. Postoje dvije vrste sanduka: *binarni sanduk* i *sanduk knjižnice*. *Binarni sanduci* su programi koji se mogu prevesti u izvršnu datoteku, koju se potom može pokrenuti, kao što je program naredbenog retka ili poslužitelj. Svaki mora imati funkciju koja se zove *main* (u ovom projektu *rocket*), koja definira što se događa kada se izvršna datoteka pokrene. *Sanduci knjižnice* nemaju glavnu funkciju i ne prevode se u izvršnu datoteku. Umjesto toga, oni definiraju funkcionalnosti koje se dijele između više projekata.

Sanduci korišteni unutar ovog projekta su:

- **Cargo** je upravitelj paketa. Omogućuje brzu i jednostavnu izradu inicijalnog projekta, deklariranje različitih ovisnosti, pokretanje, testiranje i objavu projekta.
- **Serde** omogućuje naredbu *derive*, koja se koristi za serijalizaciju i deserijalizaciju struktura podataka u različite formate, čime se smanjuje količina koda. Unutar projekta se koristi prilikom dobivanja podataka iz formi.
- **Diesel** zamjenjuje direktno korištenja SQL upita. Služi za *Object-Relational Mapping* (skraćeno ORM) manipulacije u bazi podataka, a u ovom je projektu korišten za stvaranje modela unutar baze podataka.
- **Sqlx** pruža podršku za manipulaciju podacima iz baze podataka pomoću SQL naredbi.

2.1.1.2 Prednosti

Prednost Rusta je velika brzina i memorijska učinkovitost budući da, bez trošenja vremena na izvođenje ili sakupljanje smeća, može pokretati usluge ključne za performanse, raditi na ugradbenim uređajima i lako se integrirati s drugim jezicima [16]. Nadalje, Rustov sustav bogat tipovima podataka i modelima vlasništva jamči sigurnost memorije, pritom omogućujući eliminacije mnogih grešaka tijekom prevođenja. Sigurno je i djeljenje podataka između dretvi te je moguće koristiti Rust za izradu

sigurnih mrežnih servera. Budući da je fokus stavljen na performanse, pogodan je za korištenje u ugradbenim sustavima, strojnom učenju te sistemskim programiranjima niske razine. Također, moguće je pisati i kod visokih performansi koji ne ovisi o vanjskim komponentama ili knjižnicama.

2.1.1.3 Nedostatci

Budući da je Rust "novi" programski jezik, još uvijek se razvija, što znači da kvaliteta možda još ne zadovoljava očekivanja vezana za stabilnost. Nedavno je Rust imao problema sa stabilnošću i povremeno je dolazilo do pada programa prilikom parsiranja velikih količina koda. Nadalje, Rustov *compiler* je dostupan samo za 64-bitne sustave, stoga se ne može koristiti na starijim računalima. Također, dokumentacija za Rust je još uvijek u izradi te se preporučuje korištenje Rusta samo u kombinaciji s online priručnikom *compiler*-a. Još uvijek nema puno podrške za alate automatske generacije koda- generacija koda može se postići ručno, ali nijedan alat ne može automatski pretvoriti projekt visoke razine u izvorni kod Rusta. Naposljetku, i dalje postoje neke funkcionalnosti koje su tek nedavno implementirane ili još nisu dovršene, kao što su biblioteka za interoperabilnost s C-om, podrška za IPv6 i mnoge druge.

2.1.2 Rocket

Razvoj Rocket web frameworka započinje Sergio Benitez 2016. godine kao osobni projekt [17]. Razvijan je dugo vremena i sa puno Rust makrosustava, kako bi se pojednostavio proces razvoja, te se zbog toga stabilni Rust *compiler* nije mogao koristiti do nedavno. 2021. godine *async/await* funkcionalnosti su dodane Rustu te ih je Rocket dodao iste godine. Rocket je jedan od najpopularnijih Rustovih web razvojnih okvira, koji se koristi u razvoju brzih, sigurnih, fleksibilnih, jednostavnih i skalabilnih web aplikacija. To je jednostavan web okvir bez mnogo dodataka, a programeri mogu proširiti mogućnosti Rocketa korištenjem Rust sanduka.

2.1.2.1 Prednosti

Rocket web aplikacije imaju brži odziv od stranica pisanih u apstraktnijim jezicima. Nadalje, ima jednostavnu i intuitivnu sintaksu te podršku za prikaz predložaka, čime se programerima olakšava pisanje kôda i razvijanje aplikacija [18]. Podržava različite vrste web aplikacija, uključujući REST API, GraphQL i druge. Također, ima podršku za izradu ruta, predložaka, struktura, formi, kolačića, tokova, testova, okruženja i drugih te se aplikacije radi poboljšanja performansi mogu skalirati. Naposljetku, otvorenog je koda, dakle dostupan je svim programerima koji ga žele koristiti i prilagođavati svojim potrebama, a glavni repozitorij se nalazi na Githubu.

2.1.2.2 Nedostatci

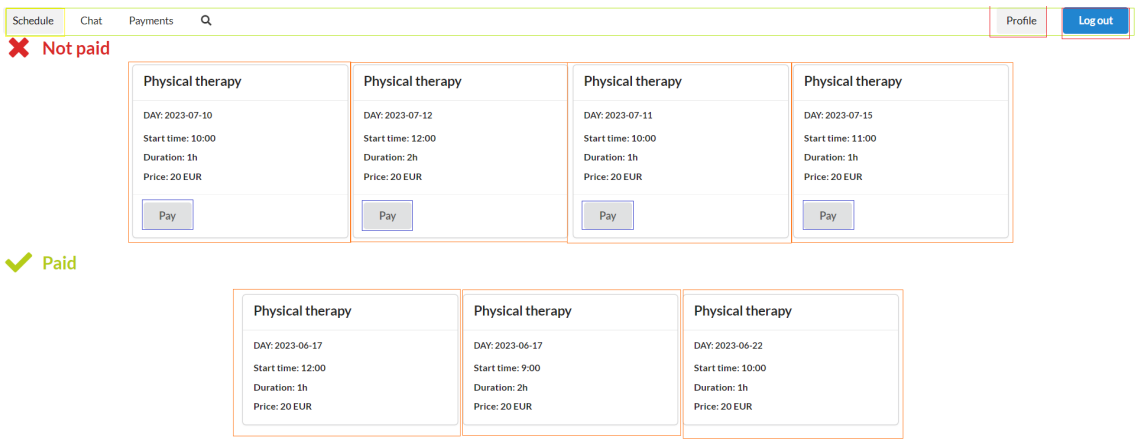
Jedini značajniji nedostatak Rocketa je da radi samo s noćnom (eng. *nightly*) verzijom Rusta [19]. Noćna verzija služi za isprobavanje novih eksperimentalnih značajki Rusta, dok programeri primarno koriste stabilnu verziju.

2.1.3 React

React je jedna od najpopularnijih JavaScript knjižnica otvorenog koda za razvoj korisničkih sučelja [20]. Prvi je put objavljena 2013. godine od strane kompanije Facebook (Meta).

2.1.3.1 Komponente

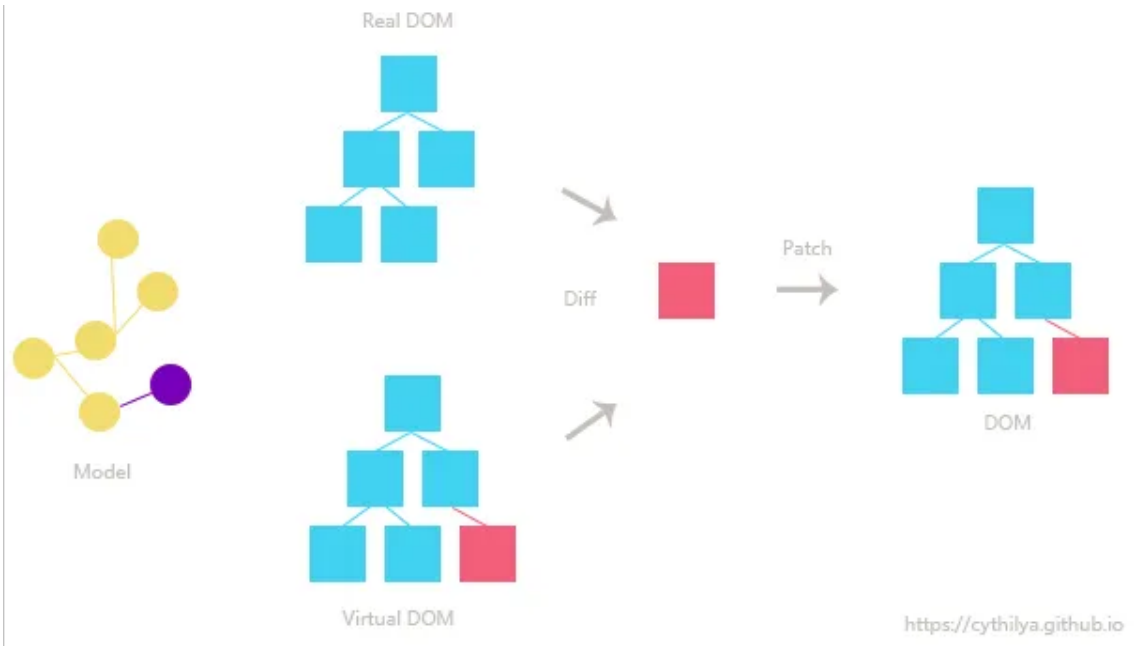
Komponente su ključni koncept u Reactu, a označavaju ponovno upotrebljive građevne blokove čijom se kombinacijom dobivaju konačna korisnička sučelja [21]. Time se omogućava lakše razvijanje i održavanje velikih aplikacija. Svaka komponenta predstavlja dio korisničkog sučelja i može se koristiti više puta u različitim dijelovima aplikacije. Na slici 2.1 prikazana je stranica s označenim komponentama koja sadrži povijesti plaćanja korisnika. U zelenom okviru se nalazi komponenta *Navigacijska traka* (eng. *Navigation bar*, skraćeno NavBar), a unutar nje se nalaze dvije komponente *Dugme* (eng. *Button*) označene crvenim okvirima. U narančastim okvirima se nalaze komponente *Plaćanje* (eng.*Payment*), a u nekima od njih se nalaze komponente *GooglePayDugme* (eng. *GooglePayButton*). U Reactu postoje dvije vrste komponenti: klase i funkcijske komponente, a unutar projekta su korištene klase.



Slika 2.1 *Komponente na stranici*

2.1.3.2 Virtualni DOM

Jedna od glavnih karakteristika Reacta je virtualni objektni model dokumenta (eng. virtual document object model, skraćeno DOM), privremena reprezentacija stvarnog DOM-a koja postoji samo u memoriji i ne prikazuje se na ekranu. DOM je reprezentacija HTML strukture web stranice, gdje je svaki element u HTML dokumentu čvor u DOM stablu. Na slici 2.2 prikazan je proces izmjene DOM-a. Kada se promijeni stanje ili svojstva React komponente, React ažurira virtualni DOM. Zatim provodi proces nazvan *"rekoncilijacija"* kako bi utvrdio razlike između prethodnog virtualnog DOM-a i ažuriranog virtualnog DOM-a. Identificira komponente ili elemente koji su se promijenili, dodali ili uklonili. Na slici je promjena očitovana u obliku nadodane kockice *Diff*. Potom, React ažurira samo ono što je potrebno i to radi vrlo optimizirano, kako bi se minimizirao broj manipulacija DOM-om. Nadodat će kockicu na stvarni DOM i time ga ažurirati.



Slika 2.2 *Virtualni DOM [22]*

Korištenje virtualnog DOM-a omogućava Reactu ažuriranje stvarnog DOM-a na učinkovit način. Umjesto izravnih i potencijalno skupih manipulacija cijelim DOM

stablom, ažuriraju se specifični dijelove koji su se promijenili. To rezultira boljim performansama i responzivnijim korisničkim sučeljem, pogotovo u složenim aplikacijama s čestim ažuriranjima.

2.1.3.3 JSX

JavaScript XML (skraćeno JSX) je jezična ekstenzija JavaScripta, koja programerima omogućava pisanje koda vrlo sličnog HTML-u, odnosno definiranje izgleda korisničkog sučelja, unutar JavaScripta.

Kad se pokrene razvojni poslužitelj putem naredbe *npm start* ili se izgradi React aplikacija putem naredbe *npm run build*, pokreće se proces koji pretvara JSX kod u JavaScript naredbe. Svi JSX kodovi se u konačnici pretvaraju u pozive metode *React.createElement()*. Metoda *createElement()* je ugrađena u React biblioteku. Na slici 2.3 može se vidjeti primjer pretvorbe JSX koda u JavaScript kod. Metoda *createElement()* upućuje React da stvori element, koji je tipa *paragraph*, s unutarnjim sadržajem 'Hello world!'. Taj element *paragraph*-a se najprije stvara unutar virtualnog DOM-a te, nakon što su svi elementi za sve JSX elemente stvoreni, virtualni DOM se pretvara u stvarne naredbe za manipulaciju pravim DOM-om koje izvršava preglednik.

Slika 2.3 *Pretvorba JSX koda*

2.1.3.4 Udice (eng. Hooks)

Udice omogućavaju korištenje stanja i drugih Reactovih funkcija unutar komponenti, čime se povećava jednostavnost i efikasnost [23]. One su funkcije koje omogućuju "spajanje" na React stanje i značajke životnog ciklusa iz funkcijskih komponenti. Udice ne rade unutar klasa, već omogućuju korištenje Reacta bez klasa. Mogu se pozivati samo na najvišoj razini komponente i ne mogu biti uvjetne.

Reactove najpoznatije udice su *State*, *Context*, *Ref* i *Effect*. Mogu se napraviti i vlastite udice.

State omogućuje pamćenje informacija poput korisničkog unosa. Na primjer, komponenta obrasca može koristiti stanje za pohranu ulazne vrijednosti, dok komponenta galerije slika može koristiti stanje za pohranu odabranog indeksa slike.

Effect omogućuje povezivanje i sinkronizaciju s vanjskim sustavima. Na primjer, rad s mrežom, DOM-om preglednika, animacijama i sa drugim kodom koji nije React.

2.1.3.5 React Native

React Native je popularan okvir za razvoj mobilnih aplikacija, za iOS i Android, izrađen na temelju Reacta [24]. Budući da se koristi ista baza kodova, olakšava i ubrzava razvoj i održavanje aplikacija za različite platforme.

Objavio ga je Facebook 2015. godine kao projekt otvorenog koda, a u samo nekoliko godina postao je jedan od najboljih okvira za mobilni razvoj. Danas se koristi za neke od najpopularnijih aplikacija današnjice, kao što su Instagram, Facebook i Skype.

2.1.3.6 Prednosti

React je jednostavan za korištenje te je lako razumljiv, budući da se bazira na JavaScript jeziku i kreiranju komponenti. Nadalje, virtualni DOM omogućava veću brzinu i efikasnost u radu sa velikim i kompleksnim aplikacijama. Ažuriranjem samo izmjenjenih dijelova poboljšava performanse i ubrzava odaziv aplikacije. Također, optimizacija web stranice (Search Engine Optimization) ovisi o nekim čimbenicima

poput brzine odaziva web aplikacije. Stoga će brže React web aplikacije imati veći prioritet prikaza na vrhu tražilice od sporijih stranica pisanih u običnom JavaScriptu. Pogodan je i za kombiniranje sa drugim alatima za razvoj klijentskog dijela aplikacija, kao što su Redux, GraphQL i Next.js, a može se koristiti i za razvoj poslužiteljskog dijela aplikacija sa Node.js-om. Naposljetku, budući da je jedna od najčešće korištenih JavaScript bibilioteka, ima veliku zajednicu programera. Otvorenog je koda i dostupan svim programerima za korištenje i prilagođavanje svojim potrebama.

2.1.3.7 Nedostatci

Reactov okvir nije potpuno opremljen i bavi se samo dijelom prikaza. Neki okviri, poput Angulara, pružaju potpuno funkcionalnu *Model View Controller* (skraćeno MVC) strukturu s kvalitetnim upravljanjem. React se bavi samo dijelom prikaza, a za dodavanje kontrolera i modela potrebne su dodatne biblioteke i alati, što može rezultirati lošijom strukturom koda i obrazaca. Nadalje, React se često mijenja i time prisiljava programere na ažuriranje načina pisanja koda jer do tada korišteni načini pisanja koda postaju zastarjeli. Također, postaje teško održavati postojeću dokumentaciju u zajednici. Nedostatak dobre dokumentacije, pored toga što otežava učenje Reacta, može i usporiti razvoj timova s manje iskusnim programerima.

2.1.4 Semantic UI

Semantic UI je moderan klijentski okvir za izgradnju korisničkih sučelja web aplikacija. Koristi semantičke nazive kako bi napravio kod lakšim za razumijevanje i održavanje, čime se olakšava izgradnja kvalitetnih i intuitivnih korisničkih sučelja [25].

Uključuje veliki broj gotovih CSS stilova, JavaScript komponenti i smjernica za upotrebu, što programerima omogućava brzu i laku izradu korisničkog sučelja bez potrebe za pisanjem koda od nule. Nudi i razne grafičke dizajne, koji se mogu koristiti za brzo kreiranje modernih i vizualno privlačnih sučelja.

2.1.4.1 Prednosti

React omogućava jednostavno korištenje već gotovih i jednostavnih ili kreiranja vlastitih komponenti sa Semantic dizajnom. Podržava korištenje različitih tema, datoteka s CSS i font dizajnom ili JavaScript metoda te se u konačnici dobiva elegantan izgled s interaktivnim sučeljem. Nadalje, podržava responzivni dizajn, stoga će se sučelja automatski prilagođavati veličini ekrana. To znači da će sučelja lijepo izgledati i na većim ekranima računala i manjim ekranima mobilnih uređaja. Naposljetku, otvorenog je koda što ga čini besplatnim za korištenje i izmjenu.

2.1.4.2 Nedostatci

Semantic UI podržava manji broj internet preglednika od njegovog konkurenta Bootstrapa, a kao jedan primjer možemo uzeti Internet Explorer 7 [26]. Nadalje, rijetko se ažurira. Razvoj Semantic UI-ja izgledao je napušten dugo vremena, a između 2018. i 2019. godine nije bilo niti jedne promjene, što je potaknulo zabrinutost zajednice. To je potaknulo zajednicu da stvori nove varijante Semantic UI-ja, kao što je Fomantic-UI. Naposljetku, ima manju zajednicu od Bootstrapa. Dok Bootstrap ima preko 50.000 pratitelja na GitHubu, Semantic UI ima manje od 1.000. Na stranici StackOverflow postoji samo 2.500 pitanja za Semantic UI, od kojih otprilike 25% nije dogovoreno.

2.2 Ostali alati

2.2.1 PostgreSQL

PostgreSQL je moćan objektno-relacijski sustav baze podataka otvorenog koda s više od 35 godina aktivnog razvoja [27]. Podržava relacijske i nerelacijske te nudi napredne *Structured Query Language* (skraćeno SQL) funkcije, uključujući strane ključeve, podupite i okidače.

2.2.1.1 Prednosti

Koristi *Multiversion concurrency control* (skraćeno MVCC) za istovremenu obradu i visoke stope transakcija s malom stopom zastoja. Visoka dostupnost i oporavak od kvara poslužitelja pružaju mu visoku stabilnost. Nadalje, za upravljanje podacima podržava SQL, što ga čini jednostavnim za korištenje, i *JavaScript Object Notation* (skraćeno JSON). Podržava i transakcijska procesiranja, čime se omogućuje konzistentan rad s podacima, čak i u okvirima složenijih i zahtjevnijih transakcijskih procesa. Isto tako, pruža podršku za funkcije poput replikacije, sigurnosti i kontrola verzija te nestrukturirane vrste podataka (na primjer video i slike). Također, enkripcije podataka, SSL certifikati i napredne metode provjere autentičnosti omogućavaju sigurnost. Osim toga, može se koristiti na različitim platformama, uključujući Windows, Linux i macOS operacijske sustave. Naposlijetku, ima veliku zajednicu te je besplatan za korištenje.

2.2.1.2 Nedostatci

PostgreSQL je sporiji od drugih sustava za upravljanje bazama podataka, a brže izvršavanje imaju SQL Server i MySQL. Fokus je postavljen na unaprijeđenje kompatibilnosti, dok poboljšanje brzine izvršavanja zahtijeva dodatni rad. Također, instalacija može biti kompliciran za početnike.

2.2.2 Visual Studio Code

Visual Studio Code je besplatni integrirani razvojni okvir (eng. integrated development environment, IDE) otvorenog koda razvijen od strane Microsofta, koji se koristi za pisanje i uređivanje koda različitih programskih jezika. Omogućava programerima da unutar jednog programa imaju implementirano sve potrebno za izradu projekta.

2.2.2.1 Prednosti

Visual Studio Code ima podršku za mnoštvo jezika poput Jave, Javascripta, TypeScripta, Pythona, C++-a, Rusta i mnogih drugih. Ima i prilagodljiv i modularn dizajn s kojim omogućava programerima dodavanje novih funkcionalnosti i proširenja razvijenih od strane korisničke i razvojne zajednice. Sadrži niz ugrađenih alata, uključujući podršku za pokretanje projekata, integraciju sa Git-om, inteligentno ispravljanje grešaka, terminal i mnoge druge korisne funkcije. Pored toga, dostupan je na različitim platformama, uključujući Linux, macOS i Windows. Također, moguće ga je integrirati sa drugim Microsoft-ovim proizvodima i uslugama, kao što su Azure i Visual Studio Team Services. Naposljetku, besplatan je za korištenje.

2.2.2.2 Nedostatci

Visual Studio Code može biti zahtjevniji po resursima od nekih primitivnijih uređivača i stoga može sporije raditi na starijim računalima [28]. Također, može nekad biti teško otkriti greške kodu, ponajviše početnicima, a može biti i izazovno naviknuti se na veliki skup alata i značajki.

2.2.3 Git

Git je distribuirani sustav za kontrolu verzija koda. Prilikom rada na jezgri Linuxa programeri više nisu mogli koristiti BitKeeper, koji nije bio otvorenog koda, te im ostale zamjene za njega nisu bile odgovarajuće [29]. Kao odgovor na taj problem finski inžinjer Linus Torvalds, autor Linux operativnog sustava, 2005. godine objavljuje Git. Git se koristi se za praćenje promjena u kodu i upravljanje verzijama programa.

Omogućava programerima i timovima efikasnu i istovremenu suradnju na projektima, praćenje promjena, poništavanje grešaka i održavanje povijesti koda. Postao je popularan među programerima zbog svoje jednostavnosti, brzine i mogućnosti rada sa različitim granama koda.

Git se koristi na nivou naredbene linije, a postoje i mnoga grafička sučelja koja olakšavaju njegovo korištenje. Neke od najčešće korištenih naredbi u Gitu su *init*, *clone*, *add*, *commit*, *branch*, *merge*, *push*, *pull*, a postoje i mnoge druge naredbe koje se koriste za naprednije funkcionalnosti [30].

2.2.3.1 Naredbe

Ukoliko postoji potreba za kreiranjem novog Git repozitorija, to se može napraviti koristeći naredbu *git init*, a za lokalno kopiranje nekog vanjskog repozitorija koristi se naredba *git clone*. Prilikom rada na projektu i dodavanja novih značajki, dobra je praksa periodično spremati promjene. Naredbom *git add* označavaju se datoteke čije se promjene žele spremiti, a naredbom *git commit* te se promjene spremaju uz poruku (*commit message*) kojom se opisuju promjene. Nadalje, projekti na kojima radi više osoba ili se dodaju i testiraju nove značajke, puno su pregledniji kada se sastoje od *grana* (eng. *branch*). Svaka grana može odgovarati nekom zadatku te se produkcijski kod može sigurno odvojiti od testnog koda. Grane se kreiraju naredbom *git branch*, a spajaju se pomoću naredbe *git merge*. Dalje, kada se u lokalni direktorij trebaju povući nove promjene s vanjskog repozitorija, to se može napraviti koristeći naredbu *git pull*, dok se promjene na vanjski repozitorij spremaju naredbom *git push*.

Popularne platforme za upravljanje projektima, poput GitHub-a i GitLab-a, koriste Git za upravljanje kodom i suradnju na projektima. Za potrebe ovog projekta, lokalno je korišten Git, dok se glavni repozitorij nalazi na platformi Github.

2.2.3.2 Prednosti

Git Pruža najbolje performanse kada su u pitanju sustavi kontrole verzija s optimiziranijim naredbama od ostalih sustava. Kriptira podatke pomoću metode SHA-1 te njegovi algoritmi sigurno upravljaju verzijama, datotekama i revizijama kako ne

bi došlo do oštećenja podataka. Također, omogućuje i više međusobno neovisnih lokalnih grana, prebacivanje konteksta, kod temeljen na ulogama (grane za testiranje i produkciju) i eksperimentiranje bez gubitka prethodnih verzija. Nadalje, repozitorij ili kompletna baza koda se može zrcaliti na lokalni sustav programera. Naposljetku, ima veliku zajednicu i dobru dokumentacija, a uz to je i besplatan za korištenje.

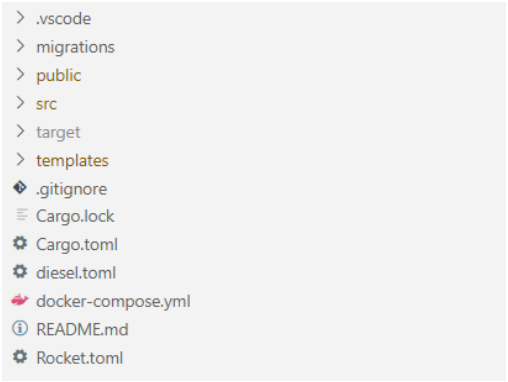
2.2.3.3 Nedostatci

Git nema mogućnost spremanja velikih binarnih datoteka u centralni repozitorij Git-a jer ne komprimira velike binarne datoteke na svom centralnom repozitoriju. Nadalje, početak korištenja Git-a može biti izazovan. Postoji veliki popis naredbi koje treba razumjeti i znati ih primjeniti, što može biti prilično zastrašujuće za početnike, pogotovo one bez iskustva s korištenjem sustava za kontrolu verzija. Zatim, nema ograničenja pristupa centralnom repozitoriju Git-a. Svi koji imaju pristup centralnom repozitoriju imaju pristup svim dostupnim informacijama, a to izlaže Git sigurnosnim prijetnjama i napadima.

Poglavlje 3

Raspored projekta

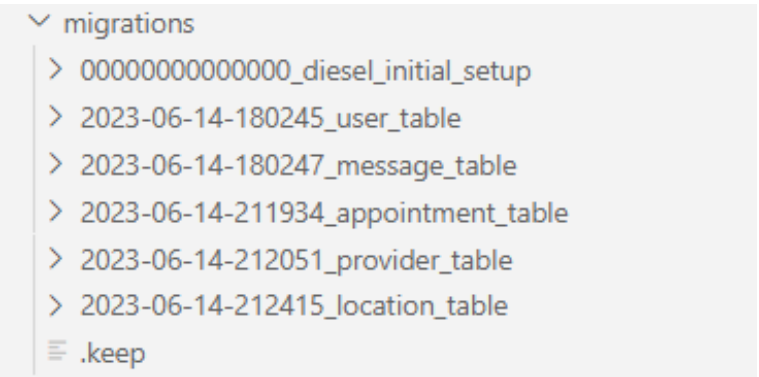
Na slici 3.1 prikazane su datoteke i direktoriji unutar glavnog direktorija projekta.



Slika 3.1 *Glavni direktorij projekta*

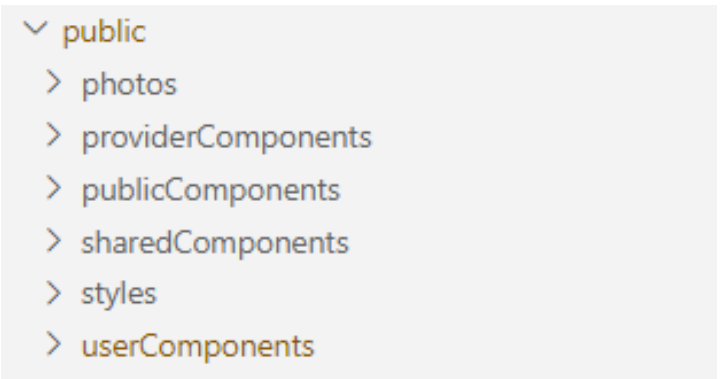
Direktorij *.vscode* automatski stvara Visual Studio Code te se u njemu nalaze konfiguracijske datoteke projekta: *settings.json* i *launch.json*.

Direktorij *migracije* (eng. *migrations*) se stvara prilikom prvog pokretanja projekta. Unutar direktorija se nalaze datoteke sa skriptama migracija koje se izvršavaju na bazi podataka kako bi se stvorila ili ažurirala njena struktura ili podaci. Sadržaj direktorija je prikazan na slici 3.2.



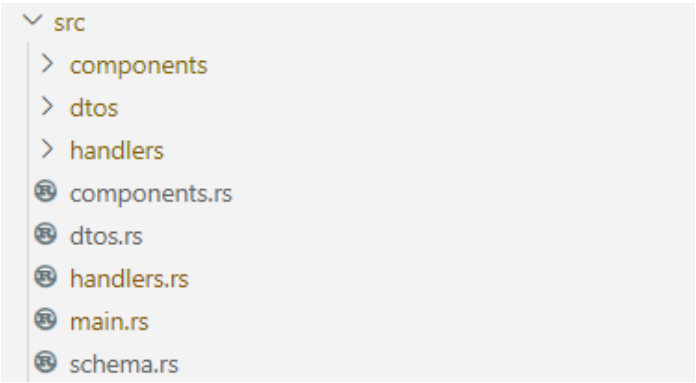
Slika 3.2 *Direktorij migrations*

Direktorij *public*, čiji je sadržaj vidljiv na slici 3.3, sadrži direktorije s React komponentama (eng. *adminComponents, userComponents, sharedComponents*), stilovima dizajna (eng. *styles*) i fotografijama (eng. *photos*) koje se prikazuju na korisničkom sučelju.



Slika 3.3 *Direktorij public*

U direktoriju *src*, čija je struktura vidljiva na slici 3.4, nalazi se izvorni kod projekta.



Slika 3.4 *Direktorij src*

U datoteci *main.rs* se nalazi glavna funkcija *rocket* s kompletnom logikom programa.

Datoteka *schema.rs* sadrži definicije tablica i njihovih struktura u bazi podataka.

Kako bi u Rustu kod bio organiziran u hijerarhijsku strukturu, koriste se moduli. Svaki modul ima datoteku s ekstenzijom .rs i direktorij jednakog imena. To je vidljivo na primjeru modula *komponente* (*components*)- postoji datoteka *components.rs*, u kojoj su definirani podmoduli (*provider*, *appointment* i drugi) i direktorij *components* u kojem se nalaze definicije svakog od podmodula.
Modul komponente sadrži strukture koje se nalaze u bazi podataka (*pružatelje usluga, korisnike, sastanke, poruke, lokacije*), enumeraciju *reponse* kojom će se kasnije definirati izlazi iz funkcija te zajedničke funkcije za upravljanje kolačićima (*shared_functions*).

Modul *dtos* služi za slanje izmjenjenih struktura podataka na klijentski dio aplikacije. Na primjer, prilikom prikaza sastanka pružatelju usluga se želi prikazati i ime korisnika koji je sastanak naručio, ali struktura sastanak sadrži samo identifikator korisnika. Iz baze podataka se vraćaju podaci sastanka kojeg će se proširiti s podacima o korisniku, to se sprema u dto strukturu *user_appointment* i prikazuje pružatelju.

Modul *rukovatelji* (*handlers*) se sastoji od dva podmodula: *funkcije nad bazom* (*sql_functions*) i *prikazi predložaka* (*templating*). *Funkcije nad bazom* služe za dobivanje podataka iz baze u JSON obliku i spremanje podataka u bazu. *Prikazi predložaka* za dane rute vraćaju ime predloška kojeg se prikazuje korisniku.

U direktoriju *templates* se nalaze Tera predlošci koji se prikazuju na klijentskom dijelu aplikacije. Tera predlošci su dizajnirani da pružaju zaštitu od potencijalnih sigurnosnih problema (na primjer *Cross-Site Scripting* napada), omogućavaju ponovnu uporabu koda (*extend* i *include* naredbe), imaju dobre performanse i omogućavaju korištenje petlji.

Gitignore je datoteka koju koristi Git kontrola toka, a služi za ignoriranje datoteka i direktorija prilikom praćenja promjena i dodavanja u Git repozitorij.

Cargo.toml je konfiguracijska datoteka u kojoj se definiraju zavisnosti (eng. dependencies) i postavke projekta.

Kada se *Cargo.toml* kompilira, nastaje *Cargo.lock* datoteka. Prilikom ponovnog kompiliranja, Cargo će koristiti već kompiliranu verziju unutar *Cargo.lock* datoteke te će time ubrzati proces kompiliranja.

U *README.md* datoteci se nalaze upute za pokretanje projekta, u *Markdown* sintaksi.

Rocket.toml datoteka sadrži podatke za konfiguraciju Rocket projekta. To može biti *adresa*, *vrata* (eng. *port*), *broj radnika* (eng. *workers*), *zapisi* (eng. *log*) i drugi.

U *diesel.toml* datoteci se nalaze konfiguracijske informacije koje se koriste za definiranje veze sa bazom podataka, direktorija za migracije, funkcionalnosti u Diesel biblioteci i generiranje SQL shema baze podataka na osnovu Diesel modela.

Datoteka *docker-compose.yml* se koristi za definiranje više Docker spremnika i njihovih konfiguracija, kako bi se pokrenula kompleksna aplikacija ili servisno okruženje.

Poglavlje 4

Sintaksa koda

4.1 Rocket

Rocket sintaksa je dizajnirana da bude intuitivna i jednostavna za korištenje i snalaženje. Unutar projekta se razlikuju iduće komponente:

- Izvršna funkcija: Nalazi se unutar datoteke *main.rs*, obavezna točka izvršavanja Rust programa.
- Struktura: Struktura i logika korištenih podataka, nastala korištenjem tehnika objektno-orijentiranog programiranja.
- Rukovatelj: Korisničko sučelje aplikacije koje, sukladno zahtjevima, procesira i vraća korisniku podatke. Razlikuju se:
  - Funkcije nad bazom (sql_functions)
  - Prikazi predložaka (templating)

4.1.1 Izvršna funkcija (*main.rs*)

Prilikom pokretanja programa, Rust će automatski pozvati funkciju *rocket* i izvršiti njen kod prikazan na slici 4.1 te nastaviti prema ostalim funkcijama koje poziva iz funkcije *rocket* ili iz drugih dijelova programa.

```
#[get("/schedule_appointments/<provider_id>", format = "json")]
pub async fn schedule_appointments(
    mut db: Connection<Logs>,
    cookies: &CookieJar<'_>,
    provider_id: i64,
) -> Json<Vec<Appointment>> {
    let mut appointments: Vec<Appointment> = Vec::new();
    let user_id = get_cookie_id(cookies);

    let query = sqlx::query(r#"SELECT *, CAST(appointments.day AS VARCHAR) AS appointment_day FROM
        appointments WHERE provider_id = $1;"#)
                .bind(provider_id)
                .fetch_all(&mut *db)
                .await
                .unwrap();

    for row in query {
        let appointment = Appointment::new(
            row.get("id"),
            Some(0),
            row.get("provider_id"),
            row.get("appointment_day"),
            row.get("start_hour"),
            row.get("start_minute"),
            row.get("duration"),
            String::new(),
            Some(0.0),
            false,
            row.get("alt"),
            row.get("lng"),
        );

        appointments.push(appointment);
    }
    return Json(appointments);
}
```

Slika 4.1 *Funkcija rocket*

Iduće naredbe se koriste za kontrolu upozorenja. Naredba *#[allow(dead_code)]* isključuje upozorenja za funkcije i komponente, a *#[allow(unused)]* upozorenja za

varijable, argumente funkcija ili druge entitete koje se ne koristi. Naredba *#[macro_use]* omogućuje korištenje naredbe *#derive* unutar struktura, o kojima će se pričati malo kasnije.

Nadalje, definiraju se moduli, paketi i funkcionalnosti koje će biti korištene unutar *rocket* funkcije. Naredba *extern crate* se odnosi na vanjski paket definiran unutar *Cargo.toml* datoteke, dok *mod* definira module iz projekta. Naredba *use* precizira koje će se funkcionalnosti iz prethodno definiranih paketa i modula koristiti.

Unutar funkcije se najprije koristi naredba *rocket::build()*, kojom se pokreću aplikacija i server.

Metoda *.attach()* služi za dodavanje *Fairing*-a u web aplikaciju. *Fairing* omogućava prilagodbu i modifikaciju aplikacije u različitim fazama njenog izvršavanja- poput inicijalizacije globalnog stanja ili postavljanja veze prema bazi podataka. U ovom slučaju, *Logs::init()* je veza prema PostgreSQL bazi podataka.

Metoda *mount* povezuje rute i rukovatelje unutar aplikacije. Kada se pozove *.mount()*, prosljeđuje mu se putanja ("/") i lista ruta (routes![..]) koje će biti mapirane na tu putanju. Svaka ruta ima definiranu *Hypertext Transfer Protocol* (skraćeno HTTP) metodu i odgovarajuću funkciju rukovatelja koja će se izvršiti.

U idućem primjeru se na ruti *"/static"* nalazi *FileServer* koji će posluživati statične datoteke iz direktorija *"public"*. Datotekama se sada može pristupati koristeći poveznicu nalik ovoj: *"static/scheduleMe.png"*.

Ukoliko ne postoji rukovatelj za rutu na kojoj se korisnik nalazi, izvršit će se rukovatelj *not_found*.

Pozivanjem metode *.attach(Template::fairing())* dodaje se Fairing sa podrškom za prikazivanje *HyperText Markup Language* (skraćeno HTML) predložaka u aplikaciji.

4.1.2 Struktura (Struct)

Unutar ovog projekta, strukture se nalaze u direktoriju: src/components. Kao primjer jedne stukture može se uzeti *sastanak* (*Appointment*) sa slike 4.2. Unutar ove strukture je sadržano sve potrebno za razlikovanje različitih sastanaka unutar baze podataka. Struktura je *javna* (eng. *public*), što omogućuje vidljivost iz svih dije-

```rust
#[derive(FromForm, Debug, Serialize, Deserialize, Clone)]
pub struct Appointment {
    id: Option<i64>,
    user_id: Option<i64>,
    provider_id: Option<i64>,
    day: String,
    start_hour: i32,
    start_minute: i32,
    duration: String,
    kind: String,
    price: Option<f32>,
    paid: bool,
    alt: Option<f64>,
    lng: Option<f64>
}
```

Slika 4.2 *Struktura Appointment*

lova projektnog koda. Omotač *Option* omogućava kreiranje obrazaca bez određenog atributa. Na primjer, prilikom registracije sastanka, korisnik ne upisuje jedinstveni *identifikator sastanka* (eng. *id*), već se on kreira naknadno, prilikom umetanja novog sastanka u bazu podataka. *Korisnikov identifikator* (eng. *user_id*) služi za povezivanje sastanka s korisnikom koji ga je zatražio. *Dan* (eng. *day*), *početni sat* (eng. *start_hour*) i *početna minuta* (eng. *start_minute*) označavaju vrijeme početka sastanka, a *trajanje* (eng. *duration*) sastanka može biti 45 minuta, 1 sat ili 2 sata. *Vrsta* (eng. *kind*) označava vrstu sastanka te su trenutno dostupne opcije: masaža, fizikalna terapija ili trening. *Cijena* (eng. *price*) je izražena u eurima i označava naknadu koja se naplaćuje korisniku putem *Google Pay*-a, a zapis o plaćanju sastanka nalazi se u atributu *plaćen* (eng. *paid*). Lokacija sastanka sadržana je u atributima *visina* (eng. *latitude*, skraćeno alt) i *širina* (eng. *longitude*, skraćeno lng), koji će služiti za izračun udaljenosti lokacija.

4.1.3 Rukovatelj (Handler)

Rukovatelji su definirani u direktoriju: handlers

4.1.3.1 Prikazi predložaka (templating)

Na slici 4.3 nalazi se prikaz predloška *schedule_get()*, koji se poziva na ruti "/schedule" korištenjem GET metode. U slučaju da se želi dohvatiti podatke *POST*

```
#[get("/schedule")]
pub async fn schedule_get(cookies: &CookieJar<'_>) -> Result
    <Template, Redirect> {
    match check_cookies(cookies) {
        true => {
            return Ok(Template::render("schedule", context! {}
                ));
        }
        _ => {
            return Err(Redirect::to(uri!(login_get())));
        }
    }
```

Slika 4.3 *Prikaz predloška raspored*

metodom, pritom šaljući podatke unutar objekta imena *appointment*, prva će linija izgledati ovako:

```
#[post("/schedule", data = "<appointment>")]
```

Postojanje podataka unutar korisnikovih kolačića se provjerava u posebnoj funkciji *check_cookies* iz koje se vraća odgovor u obliku *bool* tipa podataka. Ovisno o odgovoru, rukovatelj vraća *Result*, koji može biti uspješan ili neuspješan. Ukoliko kolačići postoje, rezultat biti vraćen u omotaču *Ok()* te će se korisniku vratiti predložak rasporeda, a ukoliko kolačići ne postoje, rezultat će se vratiti unutar omotača *Err()* te će korisnik biti preusmjeren na drugu stranicu.

4.1.3.2 Funkcije nad bazom (sql_functions)

Pomoću funkcije na slici 4.4, iz baze podataka se dobiva popis svih sastanaka odabranog pružatelja usluge.

U prvoj liniji se uz vrstu metode definira i izlazni tip podatka iz funkcije- JSON.

Funkcija je *asinkrona* (eng. *async*) jer se unutar funkcije koristi *čekanje* (eng. *await*). Asinkrona funkcija ne blokira izvršavanje ostatka koda dok se ne završi neki

```rust
#[get("/schedule_appointments/<provider_id>", format = "json")]
pub async fn schedule_appointments(
    mut db: Connection<Logs>,
    cookies: &CookieJar<'_>,
    provider_id: i64,
) -> Json<Vec<Appointment>> {
    let mut appointments: Vec<Appointment> = Vec::new();
    let user_id = get_cookie_id(cookies);

    let query = sqlx::query(r#"SELECT *, CAST(appointments.day AS VARCHAR) AS appointment_day FROM
        appointments WHERE provider_id = $1;"#)
                .bind(provider_id)
                .fetch_all(&mut *db)
                .await
                .unwrap();

    for row in query {
        let appointment = Appointment::new(
            row.get("id"),
            Some(0),
            row.get("provider_id"),
            row.get("appointment_day"),
            row.get("start_hour"),
            row.get("start_minute"),
            row.get("duration"),
            String::new(),
            Some(0.0),
            false,
            row.get("alt"),
            row.get("lng"),
        );

        appointments.push(appointment);
    }
    return Json(appointments);
}
```

Slika 4.4 *Prikaz sastanaka iz baze*

njen zadatak, već kreira objekt koji predstavlja budući rezultat operacije te nastavlja s izvršavanjem ostalog koda dok se taj zadatak ne završi. U funkciju se šalju kolačići i veza na bazu podataka (*db*). Korisniku se vraća JSON u kojem se nalazi vektor struktura *sastanak*.

Vektori pružaju dinamičnost i fleksibilnost u radu sa strukturama podataka. Sadrže različite operacije, metode pristupa i manipulacije elementima te mehanizme

za efikasno upravljanje memorijom. Vektor imena *appointments* će biti vraćen iz funkcije.

Prilikom poziva baze se prvo definira funkcija *sqlx::query* kojoj se šalje upit (eng. query): vrati sve sastanke iz baze gdje je identifikator pružatelja usluge jednak *provider_id*-u. Bind služi da se oznaka *$1* u upitu zamjeni varijablom *provider_id*. Naredbu *fetch_all* se koristi kada se očekuje da će rezultat biti više redova u tablici te joj se šalje pokazivač na tablicu. *Await*-om se označava ranije spomenuto čekanje na povrat informacija, a *unwrap* će rezultat akcije umotati u omotač *Rezultat*.

Podaci iz svih vraćenih redaka se pretvaraju u strukture *sastanak* i dodaju u vektor.

Naposlijetku, vektor se pretvara u Json oblik te isti vraćamo iz funkcije.

4.2 Tera

Predložak predstavlja sloj koji se bavi prezentacijom podataka korisniku. To je tekstualna datoteka s logikom prikaza podataka, koja sadrži izgled web stranice i način popunjavanja iste podacima. U prethodnom je primjeru bio zahtjev za prikazom predloška *raspored* (eng. *schedule*) na izlazu iz funkcije. Kada se zatražio prikaz predloška *raspored*, program je dohvatio datoteku odgovarajućeg imena (schedule.html.tera) iz direktorija templates. Sadržaj schedule.html.tera datoteke nalazi se na slici 4.5:

Budući da se obrasci za dizajn i navigacijsku traku ponavljaju u nekoliko predložaka, taj je dio koda odvojen u zasebnu datoteku base.html.tera te se time povećala ponovna iskoristivost koda. Korištenjem naredbe *extend "base"* se, brzo i jednostavno, dohvaćaju CSS stilovi, React skripte i ikona aplikacije te se ugrađuje navigacijska traka.

Budući da je za izradu ovog projekta korišten React, skripte koje se nalaze na dnu će se pozvati i unutar prethodnih *div* elemenata umetnuti komponente. Komponente podatke dobivaju na temelju informacija o sastancima. Detaljnije će se o tome pričati u nastavku rada.

```
{% extends "base" %}
{% block content %}
<script async
    src="https://maps.googleapis.com/maps/api/js?key
        =AIzaSyBL1NQunxEmWEwwsdPkxpZY9A9gqD_csl8&callback=initMap">
</script>

<div class="full_page">
    <div id="table">
    <div id="schedule"></div>
    </div>
</div>

<script src="/static/sharedComponents/loading.js" type="text/babel"></script>
<script src="/static/userComponents/addAppointment.js" type="text/babel"></script>
<script src="/static/sharedComponents/table.js" type="text/babel"></script>
<script src="/static/userComponents/beforeSeparator.js" type="text/babel"></script>
<script src="/static/userComponents/afterSeparator.js" type="text/babel"></script>
<script src="/static/userComponents/appointment.js" type="text/babel"></script>
<script src="/static/userComponents/schedule.js" type="text/babel"></script>
{% endblock content %}
```

Slika 4.5 *Prikaz schedule predloška*

4.3 React

U nastavku je objašnjen dio koda komponente *Sastanak* (*Appointment*) iz korisničkog dijela aplikacije. Cijeli kod se nalazi na idućoj poveznici. Radi osiguranja "strože" interpretacije koda i primjene određenih pravila, dobivanja veće kontrole nad kodom, otkrivanja greški u ranijoj fazi i poboljšanja čitljivost koda, u svim JavaScript datotekama je korištena naredba:

```
'use strict';
```

Nasljeđivanjem komponente *React.Component*, prikazanim na slici 4.6, klasa *Sastanak* nasljeđuje njene karakteristike i funkcionalnosti, kao što su stanje i funkcije životnih ciklusa komponente (*componentDidMount*, *componentDidUpdate*, *componentWillUnmount*). Konstruktorom su definirana stanja komponente. Pomoću naredbe *super(props)* poziva se konstruktor roditeljske klase (*React.Component*). Nadalje,

inicijaliziraju se lokalna stanja komponente. Stanja (eng. *state*) služe za čuvanje i upravljanje dinamičkim podacima komponenti. Omogućavaju prilagodbu na promjene podataka i automatsko ažuriranje komponenti nakon promjene stanja. Na dnu konstruktura navode se imena funkcija u kojima se mijenja stanje komponenti. Na slici 4.7 prikazane su funkcije životnih ciklusa komponente Sastanak. Funkcija

```
class Appointment extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      duration: 0,
      before_separator_display: false,
      after_separator_display: false,
      appointment_display: false,
      extra_weeks: 0,
      location: {}
    };
    this.callback = this.callback.bind(this);
    this.updateDistance = this.updateDistance.bind(this);
  }
```

Slika 4.6 *Konstruktor komponente sastanak*

componentDidMount će se izvršiti prilikom prvog postavljanja komponente, dok će se funkcija componentDidUpdate izvršiti prilikom ažuriranja komponente. Kako bi se spriječio beskonačan broj izvršavanja funkcija koje poziva componentDidUpdate, unutar nje se provjerava ukoliko je došlo do izmjene stanja *dodatni_tjedni* (*extra_weeks*) ili *lokacija* (*locations*).

Na slici 4.8 se nalazi funkcija *updateDistance()*, pomoću koje se poziv Google-ov API. Kako bi se smanjio broj poziva na Google-ov API, na početku fukcije *updateDistance* provjerava se potreba za prikazom određenog sastanka i njegovog *odvajača* (*separator*). Neće se izračunavati udaljenost ako se sastanak nalazi u drugom tjednu, lokacija nije ispravna, lokacija prethodnog sastanka i dogovorenog se podudaraju ili početak ili kraj sastanka odgovara granici radnog vremena pružatelja usluge. Nadalje, kreiraju se objekti *odredište* (*destination*) i *početna lokacija* (*origin*), koje se šalje Google API-ju. Po uzoru na dokumentaciju Google Maps API-ja, kreira se novi objekt *DistanceMatrixService*, uz pomoć kojeg će se izračunavati udaljenost. Nada-

```
componentDidUpdate() {
  if (this.props.extra_weeks != this.state.extra_weeks) {
    this.setState({ extra_weeks: this.props.extra_weeks });
    this.check_visibility();
  }
  if (this.props.location != this.state.location) {
    this.setState({ location: this.props.location });
    this.check_visibility();
    this.updateDistance();
  }
}
```

Slika 4.7 *Metode životnih ciklusa komponente*

lje, uz prethodno definirane objekte, definiraju se zahtjevi za izračun. Vremenske udaljenosti će biti izračunane u slučaju vožnje osobnim vozilom, uzimajući u obzir promet, neće se izbjegavati cestarine te će se izvršavati na principu metričkog sustava. Budući da Google-ov API ne vraća odmah odgovor, već se odgovor mora pričekati, koristi se *povratni poziv* (*callback*), koji će nakon primljenog odgovora, ažurirati stanje komponente.

```
updateDistance() {
  this.setState({ extra_weeks: this.props.extra_weeks });
  this.check_visibility();
  if (isNaN(this.props.location.alt) || isNaN(this.props.location.lng) || isNaN(this.props.alt) || isNaN(this
      .props.lng)) return;

  const destination = new google.maps.LatLng(Number(this.props.location.alt), Number(this.props.location.lng
      ));
  const origin = new google.maps.LatLng(Number(this.props.alt), Number(this.props.lng));
  ...
    return;
  }

  var service = new window.google.maps.DistanceMatrixService();

  service.getDistanceMatrix(
    {
      origins: [origin],
      destinations: [destination],
      travelMode: window.google.maps.TravelMode.DRIVING,
      avoidHighways: false,
      avoidTolls: false,
      unitSystem: google.maps.UnitSystem.METRIC
    },
    this.callback
  );
}

callback(response, status) {
  if (status === 'OK') {
    this.setState({ duration: response.rows[0].elements[0].duration.value });
    this.props.add_appointment(this.props.day, this.props.start_hour, this.props.start_minute, this.props
        .duration, this.state.duration);
  } else {
    console.error('Error:', status);
  }
```

Slika 4.8 *Metoda updateDistance()*

Metoda *render()*, prikazana na slici 4.9, u React komponenti je odgovorna za generiranje sadržaja korisničkog sučelja. Ažurira se prilikom prvog postavljanja te se ponovno ažurira nakon promjena stanja. Unutar metode se nalaze podkomponente sastanka, *odvajači* (*BeforeSeparator* i *AfterSeparator*), kojima se šalju podaci unutar *pomagača* (*props*). Kao primjer se mogu uzeti *BeforeSeparator*, kojem se unutar pomagača *trajanje* (*duration*) šalje vrijeme trajanja odvajača, iz kojeg će se naknadno izračunati njegova visinu. Prilikom promjene stanja roditeljske komponente, ažuriraju se i njene podkomponente.

```
render() {
  return (
    <div>
      {this.state.appointment_display ?
        <div className="ui visible message" style={{ height: this.appointment_height(), right: this.right(), top
            : this.top() }}><span className="appointment_time">{this.start_time()} - {this.end_time(this.props
            .duration, this.props.start_hour, this.props.start_minute)}</span></div>
        : null}

      {this.state.before_separator_display ?
        <BeforeSeparator right={this.right()} extra_days={this.extra_days} duration={this.state.duration}
            separator_height={this.separator_height()}
          end_time={this.start_time()} appointment_start={this.top()}
        />
        : null}

      {this.state.after_separator_display ?
        <AfterSeparator right={this.right()} extra_days={this.extra_days} duration={this.state.duration}
            separator_height={this.separator_height()}
          start_time={this.end_time()} appointment_start={this.top()} appointment_height={this
              .appointment_height()}
        />
        : null}

    </div>
  );
}
```

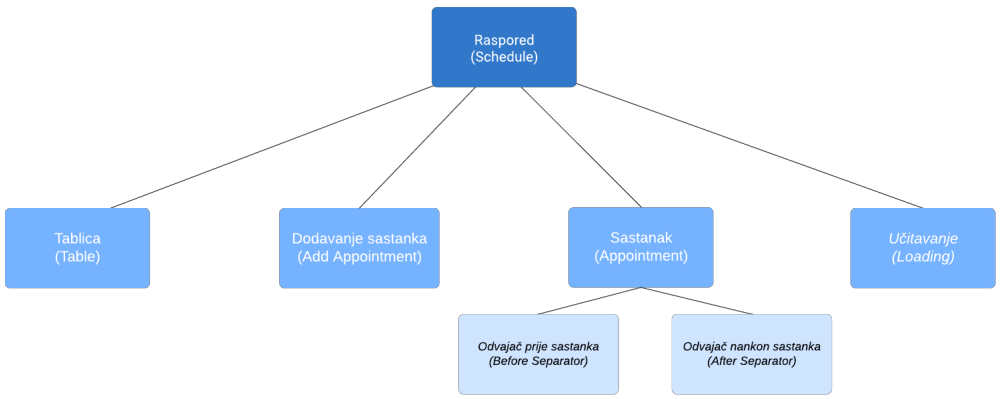
Slika 4.9 *Metoda render()*

Sastanak je u sklopu projekta uvijek korišten kao podkomponenta *Rasporeda*. Kada bi se *Sastanak* trebao prikazati kao samostalna komponenta unutar HTML elementa *div* s identifikatorom *"appointment"*, to bi se moglo učiniti kako je prikazano na slici 4.10.

```
let appointment = document.getElementById('appointment');
const root = ReactDOM.createRoot(appointment);
root.render(<Appointment />);
```

Slika 4.10 *Dodavanje komponente u DOM*

Na slici 4.11 prikazana je hijerarhija nasljeđivanja komponente raspored:



Slika 4.11 *Komponenta raspored i njene podkomponente*

Poglavlje 5

Struktura baze podataka

Na slici 5.1 se nalazi shema baze podataka. Svaki *korisnik* (model *user*) ima svoj jedinstveni *identifikator* (*id*), *ime* (*name*), *prezime* (*surname*), *email adresu*, *lozinku* (*password*), *broj telefona* (*phone*) i *opis* (*description*). Identifikator korisnika je *bigint* tip podatka i označava brojeve u rasponu od -2^{63} do $2^{63}-1$. Prvotna ideja projekta je bila koristiti samo pozitivne brojeve za pohranu identifikatora, no PostgreSQL nema tipa podataka sa samo pozitivnim brojevima. Kao odgovor na taj problem, u aplikaciji bi se mogla uvesti inkrementacija identifikatora za broj 2^{63}, ali za potrebe ovog projekta to nije bilo neophodno. Ime i prezime korisnika su tipa *character varying*, a u aplikaciji pobliže opisuju svakog korisnika. Korisnik se u aplikaciji autorizira unosom emaila i passworda, koji su oboje tipa *character varying*. Broj telefona, tipa *character varying*, do sad nije aktivno korišten, ali daljnjim proširenjem aplikacije bio bi uveden kao kanal za autorizaciju novog korisnika. *Opis* korisnika, tipa *character varying*, nije do sad korišten, ali ukoliko dođe do proširenja aplikacije, omogućila bi se pružateljima usluga mogućnost kratkog opisa korisnika i njegovih zahtjeva.

Korisnik prilikom registracije mora unijeti adresu na kojoj želi dogovarati sastanke te se ta adresa pohranjuje u model *lokacije* (eng. *locations*). Svaka lokacija ima svoj jedinstveni *identifikator*, *identifikator korisnika* (eng. *user_ id*), *visinu* (eng. *latitude*, skraćeno alt), *dužinu* (eng. *longitude*, skraćeno lng) i opis(eng. *description*). *Visina* i *dužina* su parametri tipa *double precision*, koje Google Maps koristi prilikom vremenskih izračuna. Opis je tipa *character varying* i predstavlja tekst koji se prika-

zuje na korisničkom sučelju, a njegovom pohranom smanjujemo broj poziva prema Google API-ju. Podaci o lokaciji se mogu mjenjati na korisnikovom profilu.

Slika 5.1 *Shema baze podataka*

Korisnik može imati više dogovorenih *sastanaka* (model *appointments*). Svaki sastanak ima svoj jedinstveni *identifikator*, *identifikator korisnika*, *identifikator pružatelja usluge* (*provider_id*), *dan*(*day*), *početni sat* (*start_hour*), *početnu minutu* (*start_minute*), *trajanje* (*duration*), *tip sastanka* (*kind*), *cijenu* (*price*), oznaku je li *plaćen* (*paid*) i *podatke o lokaciji sastanka* (*alt* i *lng*). U *dan*, koji je tipa podataka *date*, se sprema točan datum kada je dogovoren sastanak, a u *integer*-e početni sat i početna minuta se unosi vrijeme kad je dogovoren sastanak. Trajanje sastanka je tipa character varying, a može poprimiti vrijednost 1 sat, 2 sata ili 45 minuta. *Kind*, tipa *character varying*, opisuje tip sastanka koji je dogovoren te za sad postoje tri opcije: masaža, trening ili fizikalna terapija, dok je *cijena* svakog tretmana trenutno 20 eura. Proširenjem aplikacije, uz svakog pružatelja usluga bi se vezali tipovi tretmana koje pruža, trajanja i cjenik. Podatci o lokaciji opisuju *lokaciju* na kojoj je dogovoren sastanak, a prilikom dodavanja sastanka automatski se dohvaća lokacija koju korisnik ima pohranjenu, dok *boolean plaćen* označava je li korisnik do sada platio sastanak.

Svaki *pružatelj usluge* (model *providers*) ima svoj *identifikator*, *ime*, *prezime*, *email*, *lozinku* i *broj mobitela*. Tipovi podataka i uporabe su jednaki kao i kod modela korisnik.

Korisnici mogu s pružateljima usluga izmjenjivati *poruke* (model *messages*). Svaka *poruka* ima jedinstveni *identifikator*, *identifikator korisnika*, *identifikator pružatelja usluge* (*provider_id*), *vrijeme* (*created*) kad je kreirana u sekundama, *sadržaj* (*content*) i *boolean je li* pošiljatelj *pružatelj usluge* (*is_provider*).

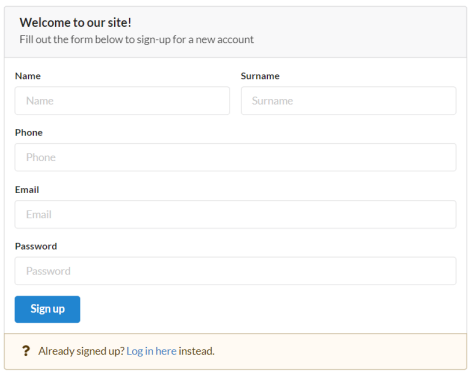
Poglavlje 6

Sučelja

6.1 Korisničko sučelje

6.1.1 Registracija korisnika (Sign up)

Prilikom odlaska na stranicu *za registraciju korisnika*, korisnicima se otvara prozor s obrascem za registraciju, kao na slici 6.1.

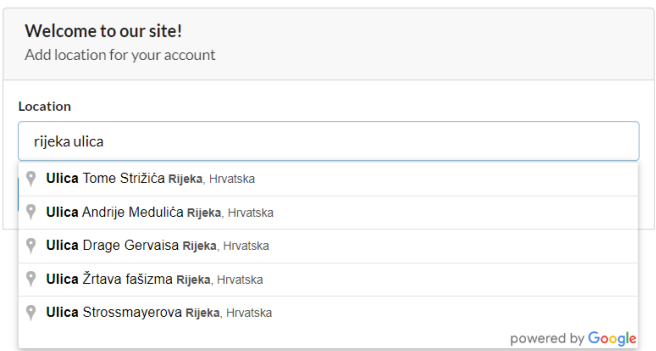


Slika 6.1 *Izgled signup sučelja*

Prilikom unosa podataka provjerava se sintaksa emaila te ukoliko je email krivo napisan, korisnik dobiva obavijest i ne dopušta se potvrda. Prilikom unosa točne email adrese, provjerava se postoji li uneseni email već u bazi podataka. Ukoliko

postoji, i unio je netočnu lozinku, korisniku se neće napraviti novi profil, već će biti preusmjeren na stranicu za *prijavu*. Također, ukoliko je postojeći korisnik u obrascu za registraciju unio ispravne podatke za prijavu, kako bi se smanjio broj zahtjeva prema bazi podataka, korisnika se prosljeđuje dalje na stranicu *pružatelji usluga* i u njegove kolačiće pohranjuje kriptirane podatke (*identifikator*, *ime*, *prezime* i *email*).

Nakon uspješnog unosa podataka novog korisnika, u kolačiće se pohranjuju kriptirani podaci i korisnika se preusmjerava na stranicu za *unos lokacije* na kojoj želi dogovarati sastanke. Korisniku se prilikom unosa lokacije generiraju odabiri mogućih adresa pomoću Google Place Autocomplete-a, koji predlaže adrese na hrvatskom jeziku, te vraća koordinate i opis korisnikovog odabira, kao na slici 6.2.

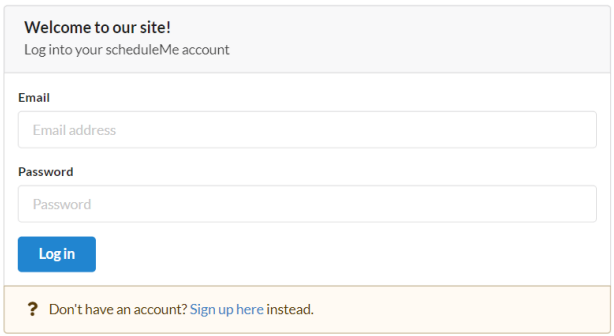


Slika 6.2 *Unos lokacije uz pomoć Google Autocomplete-a*

Nakon uspješnog unosa lokacije, korisnik se preusmjerava na stranicu *pružatelji usluga* (*providers*), gdje može odabrati svog prvog pružatelja usluga.

6.1.2 Prijava korisnika (Log in)

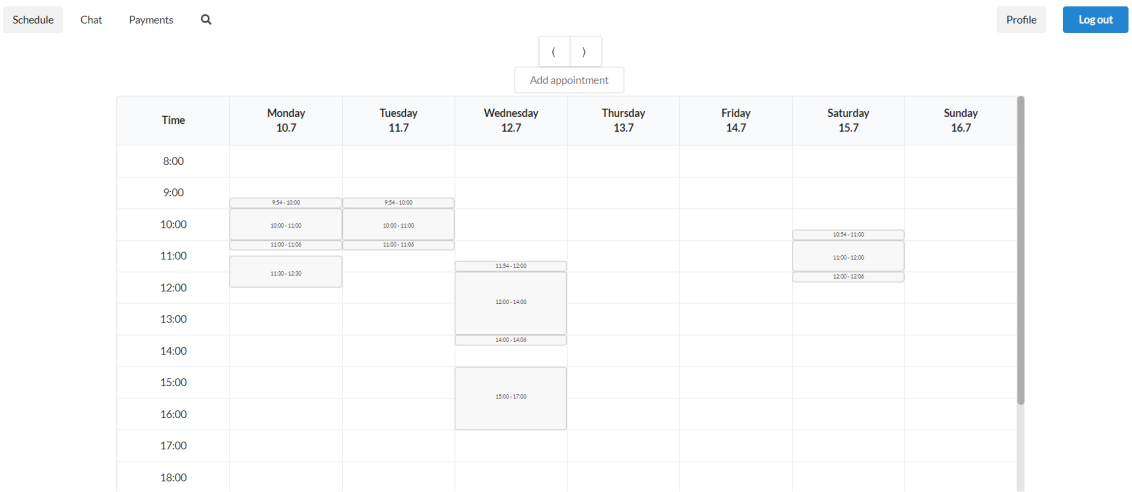
Postojeći se korisnik u aplikaciju prijavljuje putem *obrasca za prijavu* (*login form*) sa slike 6.3. U prvom polju unosi email, a u drugom polju svoju lozinku. Ukoliko su uneseni podaci ispravni, u njegove kolačiće se pohranjuju podaci, a korisnik se preusmjeri na stranicu sa popisom svih *pružatelja usluga*. Ukoliko korisnik s unesenim emailom ne postoji, preusmjerava se na stranicu za registraciju.



Slika 6.3 *Izgled login sučelja*

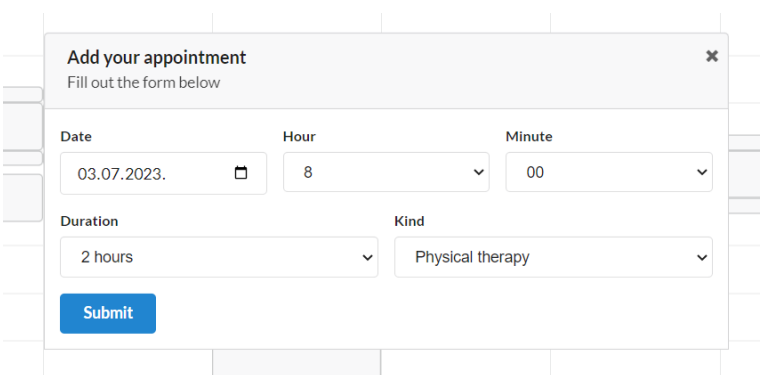
6.1.3 Glavni raspored (Schedule)

Na ovoj se stranici nalazi *raspored* s već dogovorenim sastancima, vidljiv na slici 6.4. Visina i pozicija sastanka ovise o danu i trajanju određenog sastanka. Prilikom prikaza sastanaka na rasporedu, izračunava se udaljenost adrese trenutno prijavljenog korisnika i adrese na kojoj je dogovoren sastanak te se generira *odvajač* (*separator*). Termin koji je postavljen za ponedjeljak (10.7.) u 11:30 sati nema odvajač te se iz toga može zaključiti da je taj sastanak naručio trenutni korisnik ili netko tko živi na istoj adresi kao on. Stoga, ukoliko korisnik želi više sastanaka na istoj lokaciji, jedan za drugim, to može rezervirati. Od lokacije sastanka rezerviranog u ponedjeljak (10.7.) u 10:00 sati, korisnik koji naručuje sastanak je udaljen 6 minuta. Odvajači se izračunavaju na temelju vožnje autom, bez izbjegavanja cestarina te koriste Google Distance Matrix za procjenu vremenske udaljenosti, kako bi se izračunala optimalna vrijednost trajanja vožnje. Kako bi korisnik mogao pregledavati i rasporede drugih tjedana, iznad rasporeda se nalaze gumbi sa strelicama za promjenu po tjednima.



Slika 6.4 *Izgled glavnog rasporeda*

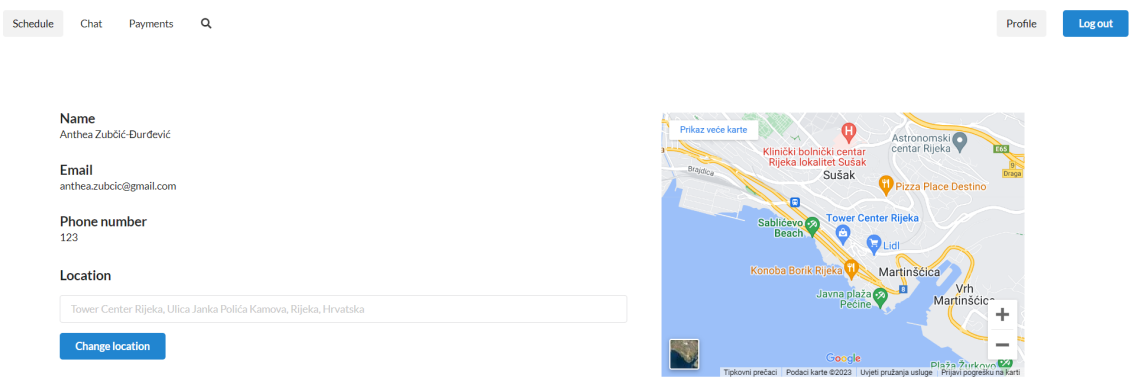
Na slici 6.5 je prikazan obrazac u kojem korisnici registriraju novi sastanak i njegove detalje. Obrazac se može otvoriti klikom na gumb *Add appointment* iznad rasporeda. Ukoliko korisnik unese termin koji se preklapa s postojećim sastankom ili odvajačem, korisniku potvrda istog neće biti omogućena.



Slika 6.5 *Dodavanje novog sastanka*

6.1.4 Korisnički profil (User profile)

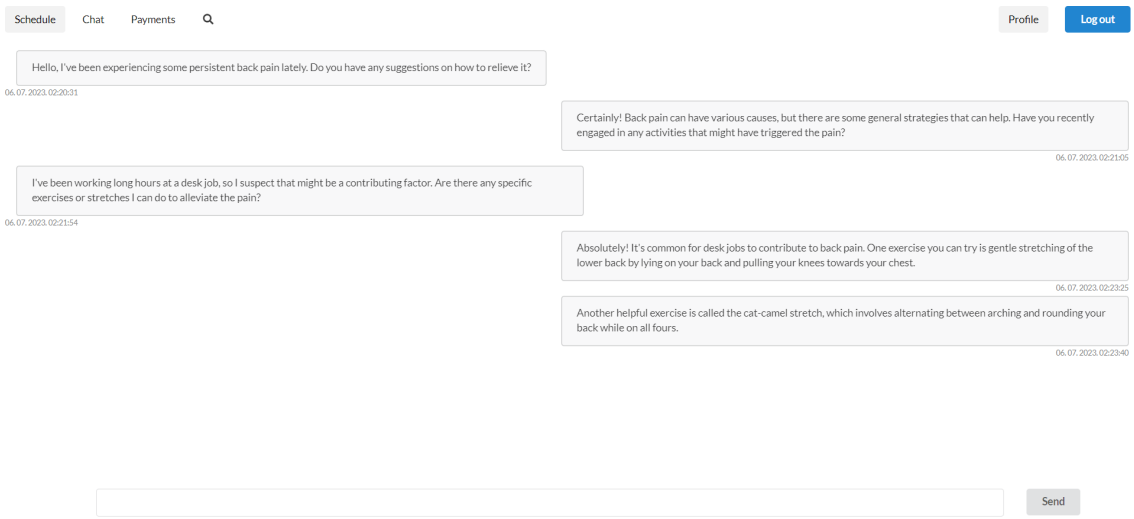
Na stranici *korisničkog profila*, prikazanog na slici 6.6, nalaze se podaci o korisniku i polje za ažuriranje korisnikove adrese. Prilikom unosa adrese, kao i prilikom registracije, koristi se Google Place Autocomplete.



Slika 6.6 *Korisnički profil (User profile)*

6.1.5 Razgovor (Chat)

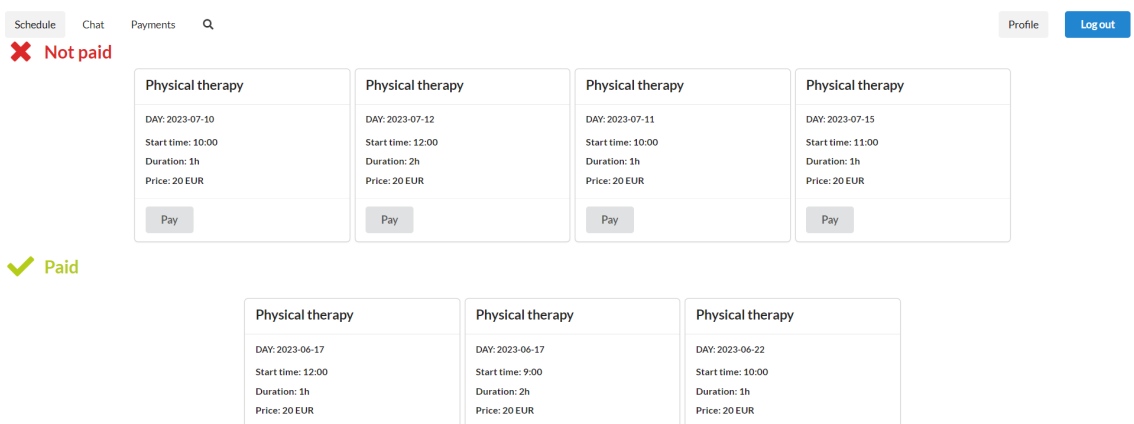
Kada korisnik uđe na stranicu *razgovor*, otvara mu se razgovor s pružateljem usluge kao na slici 6.7. Poruke su poredane po vremenu slanja te se ispod svake poruke prikazuju datum i vrijeme kad je poruka poslana, u lokalnoj vremenskoj zoni. S desne strane se nalaze poruke koje je poslao korisnik, a s lijeve poruke koje je korisniku poslao pružatelj usluge.



Slika 6.7 *Razgovor*

6.1.6 Plaćanja (Payments)

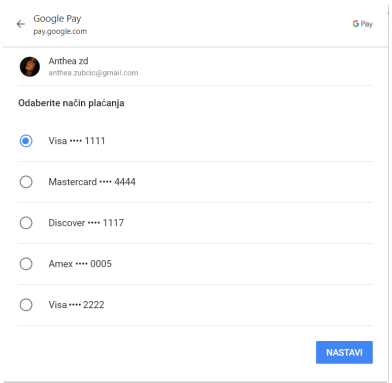
Na stranici *plaćanja* se korisnicima prikazuje popis njihovih neplaćenih i plaćenih sastanaka, kao na slici 6.8.



Slika 6.8 *Plaćanja*

Klikom na gumb *Pay*, korisniku se otvara prozor Google Pay-a u kojem može izvršiti plaćanje. Korisnik odabire karticu kojom želi izvršiti plaćanje te prati upute za završetak plaćanja.

Na slici 6.9 se nalaze testne kartice koje Google omogućuje programerima za potrebe testiranja aplikacija. Ukoliko se plaćanje uspješno izvrši, sastanak se označava plaćenim, a korisniku se sastanak pomiče u odjeljak plaćenih sastanaka.



Slika 6.9 *Odabir kartice za Google Pay plaćanje*

6.2 Sučelje pružatelja usluga

6.2.1 Registracija pružatelja usluga (Provider sign up)

Prilikom odlaska na stranicu *registracija pružatelja usluga*, otvara se prozor s obrascem za registraciju, dizajnom jednak obrascu za registraciju korisnika. Prilikom unosa podataka provjerava se sintaksa emaila te, ukoliko je email krivo napisan, pružatelj usluga dobiva obavijest i ne dopušta se potvrda. Nakon unosa email adrese ispravne sintakse, provjerava se postoji li uneseni email već u bazi podataka pružatelja usluga. Ukoliko postoji i pružatelj usluga je unio netočnu lozinku, neće se kreirati novi profil, već će biti preusmjeren na stranicu za *prijavu pružatelja usluga*. Također, ukoliko je postojeći pružatelj usluga u obrascu za registraciju unio ispravne podatke za prijavu, kako bi se smanjio broj zahtjeva prema bazi podataka, prosljeđuje ga

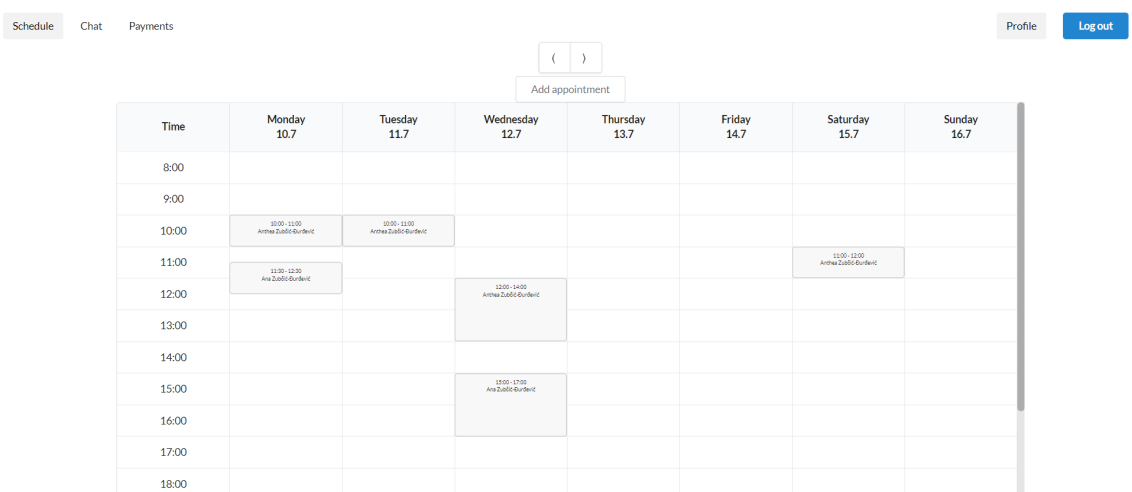
se dalje na *raspored pružatelja usluga* i u njegove se kolačiće pohranjuju kriptirani podaci (*identifikator*, *ime*, *prezime* i *email*).

6.2.2 Prijava pružatelja usluga (Provider login)

Stranica *prijave pružatelja usluga* je dizajnom jednaka stranici za *prijavu korisnika*. Provjerava se unos lozinke i email adrese te se, ukoliko su uneseni podaci točni, pružatelju usluga otvara stranica s njegovim *rasporedom*.

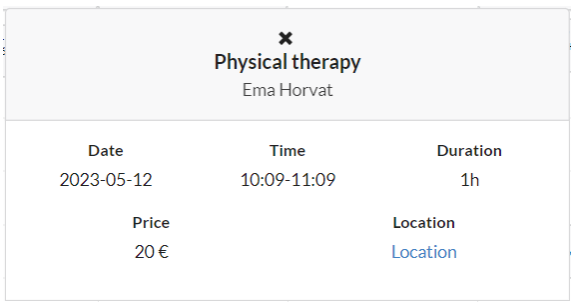
6.2.3 Raspored pružatelja usluga (Provider schedule)

Raspored pružatelja usluga dizajnom je jednak rasporedu koji je vidljiv korisnicima. Ovaj raspored ne sadrži odvajače, ali sadrži detaljne podatke o svakom sastanku, koje treba samo pružatelj usluga vidjeti. Primjer jednog rasporeda se nalazi na slici 6.10.



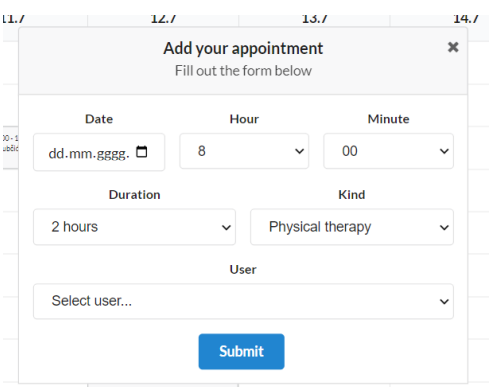
Slika 6.10 *Raspored pružatelja usluga*

Klikom na sastanak koji je prikazan na rasporedu, otvara se prozor u kojem su navedene detaljne informacije o sastanku, kao na slici 6.11. Klikom na *link s lokacijom* (*location*), pružatelju usluga se unutar servisa Google Maps otvara adresa sastanka.



Slika 6.11 *Informacije o sastanku*

Kao i kod korisnika, klikom na gumb za dodavanje sastanka otvara se prozor za registraciju novog sastanka, prikazan na slici 6.12. Klikom na zadnje polje *Korisnik*, otvara se padajući izbornik s postojećim korisnicima te pružatelj usluge odabire korisnika s kojim želi dodati novi sastanak.



Slika 6.12 *Dodavanje sastanka*

6.2.4 Razgovori pružatelja usluga (Provider chats)

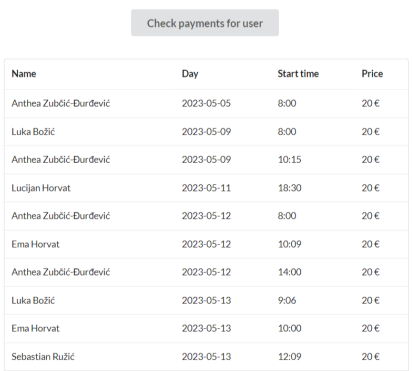
U *razgovorima pružatelja usluga* prikazan je popis korisnika s kojima je pružatelj usluga već razgovarao te gumb *Novi razgovor* (*New chat*). Klikom na gumb otvara se lista svih korisnika s kojima pružatelj usluga još nikad nije razgovarao. Ovdje pružatelj usluge ima mogućnost prvi pokrenuti razgovor s određenim korisnikom.

6.2.5 Razgovor pružatelja usluga s korisnikom (Provider chat)

Razgovor pružatelja usluga s korisnikom je izgledom i funkcionalnošću jednak razgovoru kojeg vidi korisnik. Poruke koje je poslao pružatelj usluge nalaze se na desnoj, a poruke koje je poslao korisnik na lijevoj strani.

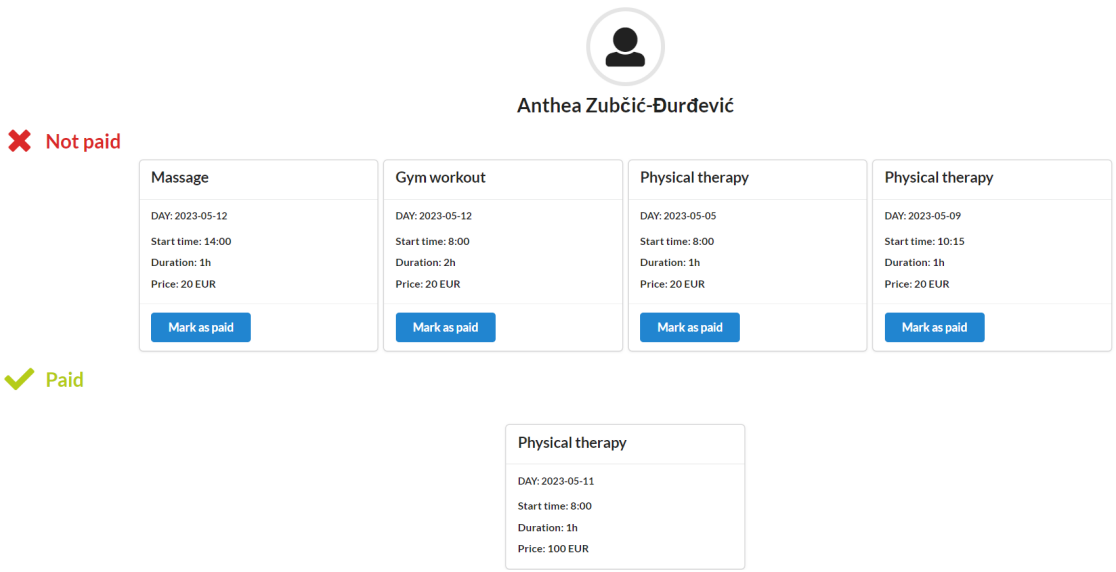
6.2.6 Plaćanja pružatelja usluga (Provider payments)

Na stranici *plaćanja pružatelja usluga* prikazan je popis sastanaka koji još uvijek nisu plaćeni (slika 6.13). Klikom na gumb *Provjeri plaćanja korisnika* (*Check payments for user*), pružatelj usluga može provjeriti povijest plaćanja za određenog korisnika.



Slika 6.13 *Neplaćeni sastanci*

Unutar prikaza povijesti plaćanja (slika 6.14), po korisniku pružatelj usluga može, ukoliko je korisnik sastanak platio nekim drugim putem, označiti sastanak plaćenim klikom na gumb *Označi plaćenim* (*Mark as paid*).



Slika 6.14 *Sastanci odabranog korisnika*

6.2.7 Profil pružatelja usluga (Provider profile)

Profil pružatelja usluga sadrži samo informacije o pružatelju usluga: *ime*, *prezime*, *email* i *broj telefona*.

Poglavlje 7

Upute za pokretanje aplikacije

- nakon instalacije svih potrebnih programa iz poglavlja 2.2 Ostali alati, potrebno je konfigurirati verzije navedene u sljedećoj tablici:

Tablica 7.1 *Korištene verzije*

	verzija
rustc	1.25.2
rustup	1.25.2
cargo	cargo 1.70.0-nightly

- otvara se *Git Bash* i nakon odabira lokalne mape u koju se želi klonirati projekt s Githuba, unosi se naredba:

```
git clone https://github.com/antheazd/rocket.git
```

- pokrenu se programe PgAdmin i Docker
- u datoteke *docker-compose.yml*, *Cargo.toml* i *Rocket.toml* unese se korisničko ime i lozinka za program PgAdmin
- otvori se Windows Command Prompt i pokreće naredbe:

```
cargo clean
docker-compose up -d
cargo install diesel\_cli --no-default-features --features "postgres"
```

- u idućoj naredbi unosi se korisničko ime i lozinka postavljena unutar programa PgAdmin te proizvoljno ime baze:

```
set DATABASE\_URL=postgres://[korisnickoime]:[lozinka]@localhost:5435/[ime
    baze]
diesel setup
diesel migration generate user_table
diesel migration generate provider_table
diesel migration generate location_table
diesel migration generate appointment_table
diesel migration generate message_table
```

  Nakon što se pokrenula *migration generate* naredba, Diesel unutar projekta stvara direktorij naziva *migrations*.

- otvari se direktorij migrations te kopira sql skripte u istoimene datoteke unutar lokalnog direktorija *migrations*.

- unutar Windows Command Prompta pokreću se naredbe:

```
diesel migration run
set ROCKET\_DATABASES={webappdb {url="postgres://postgres:rocket@localhost
    :5435/webappdb"}}
cargo run
```

Poslužiteljska strana aplikacije bi sada trebala biti pokrenuta, a aplikaciji se može pristupiti unutar internet preglednika na adresi: http://localhost:8000

U sljedećoj tablici se nalaze korištene verzije sanduka, definiranih u datoteci Cargo.toml.

Tablica 7.2 *Korištene verzije sanduka*

	verzija	značajka
rocket	0.5.0-rc.2	secrets, json
Serde	1.0	derive
Diesel	2.0.2	postgres
Sqlx	0.6.2	runtime-tokio-rustls, postgres, chrono
rocket_dyn_templates	0.1.0-rc.2	tera

Zaključak

Aplikacija scheduleMe je aplikacija namjenjena pružateljima usluga unutar doma, a kao primjer takvih pružatelja može se uzeti fizioterapija u kući. Svakim radni dan, fizioterapeut mora složiti raspored i korisnike usluge poslagati optimalno, uzimajući u obzir vrijeme prijevoza između dvije lokacije. Također, dok fizioterapeut ima sastanak s nekim od korisnika, ne može dogovarati buduće sastanke s drugim korisnicima, što ponekad rezultira odustajanjem korisnika od sastanka. Aplikacija bi omogućila korisnicima da u svakom trenutku mogu vidjeti plan fizioterapeuta i dogovoriti svoj sastanak u željenom terminu. Također, s fizioterapeutom se mogu dogovoriti i putem razgovora na platformi te fizioterapeut može unijeti sastanak umjesto njih. Platiti mogu putem aplikacije u par jednostavnih koraka. Fizioterapeut bi, prikazom njegove usluge na aplikaciji, potencijalno mogao dobiti nove klijente. Također, sam proces automatiziranog dogovaranja sastanaka bi uštedio vrijeme fizioterapeutu.

Rocket i React su bili odličan temelj za stvaranje projekta. Brzina odziva Rocketove poslužiteljske strane te brzina ažuriranja React komponenti doprinijeli su boljim performansama aplikacije. Također, podjela na React komponente i jednostavnost korištenja Semantic UI-a su omogućili brže pisanje koda. Git je pomogao pratiti promjene u kodu, dok je Visual Studio Code svojim alatima za označavanje koda te upozorenjima uvelike olakšao traženje grešaka.

Aplikacija ima prostora za dodatna poboljšanja. Poruke unutar razgovora se ažuriraju samo ponovnim učitavanjem stranice, prilikom registracije korisnika ne postoji verifikacija mobilnog broja i emaila te bi se trebala napraviti dodatna potvrda prilikom označavanja plaćenih sastanaka (kod pružatelja usluga).

Bibliografija

[1] React.js. adresa: https://github.com/facebook/react/(pogledano 6. 7. 2023.)

[2] Semantic UI adresa: https://github.com/Semantic-Org/Semantic-UI(pogledano 6. 7. 2023.)

[3] Rust adresa: https://github.com/rust-lang/rust(pogledano 6. 7. 2023.)

[4] Rocket adresa: https://github.com/SergioBenitez/Rocket(pogledano 6. 7. 2023.)

[5] The PostgreSQL Global Development Group: "What is PostgreSQL?", s Interneta, https://www.postgresql.org/about/, 2023.

[6] Chacon, S.; Straub, B.J.: "Pro Git book", 2.izdanje, Apress, 9. studenog 2014.

[7] Microsoft: "Visual Studio Code", s Interneta, https://visualstudio.microsoft.com, 2023.

[8] Jordana, A.: "What Is React & How Does It Actually Work?", s Interneta, https://www.hostinger.com/tutorials/what-is-react, 25. svibnja 2023.

[9] Pahuja, S.: "Introduction to Semantic UI React - Building Blocks of React", s Interneta, https://www.knowledgehut.com/blog/web-development/semantic-ui-react, 15. lipnja 2023.

[10] Benitez, S.: "The Rocket Programming Guide", s Interneta, https://rocket.rs/v0.5-rc/guide/, 2022.

[11] Microsoft: "Visual Studio Code Docs", s Interneta, https://code.visualstudio.com/docs, 2023.

[12] Klabnik, S.; Nichols C.; Rust zajednica: "The Rust Programming Language", s Interneta, https://doc.rust-lang.org/book/, 9. veljače 2023.

[13] Thompson, C.: "How Rust went from a side project to the world's most-loved programming language", s Interneta, https://www.technologyreview.com/2023/02/14/1067869/rust-worlds-fastest-growing-programming-language/, 14. veljače 2023.

[14] Stack Overflow: "2023 Developer Survey", s Interneta, https://survey.stackoverflow.co/2023/#technology, 2023.

[15] Odedeyi Feyisayo, A.: "Top 10 Big Companies Using Rust", s Interneta, https://careerkarma.com/blog/companies-that-use-rust/, 6. veljače 2022.

[16] Pandey, A.: "What is Rust and its pros and cons?", s Interneta, https://quicklearncomputer.com/what-is-rust/, 30. siječnja 2023.

[17] Murti, K.: "Rust Web Development with Rocket", Packt Publishing, Birmingham, lipanj 2022.

[18] Shakeel, S:: "Actix or Rocket? Comparing Two Powerful Rust Web Frameworks", s Interneta, https://levelup.gitconnected.com/actix-or-rocket-comparing-two-powerful-rust-web-frameworks-114a3540f0b3, 22. lipnja 2020.

[19] Maidan, M.; Isakova, D.: "Rust web frameworks: development in 2023.", s Interneta, https://yalantis.com/blog/rust-web-frameworks/, 2023.

[20] Deshpande, C.: "The Best Guide to Know What Is React", s Interneta, https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs, 7. veljače 2023.

[21] Schwarzmüller, M.: "React key concepts", Packt Publishing, 26. prosinca 2022.

[22] Sheth, R.: *Slika virtualnog DOM-a*, s Interneta, https://miro.medium.com/v2/resize:fit:750/format:webp/0*d4t42wjxErZFGqHC.png, 7.7.2023.

[23] Meta: "Built-in React Hooks", s Interneta, https://react.dev/reference/react#performance-hooks, 2023.

[24] Budziński, M.: "What Is React Native? Complex Guide for 2023", s Interneta, https://www.netguru.com/glossary/react-native, 2023.

[25] MIT: "Semantic UI", s Interneta, https://semantic-ui.com/, 2023.

[26] New Technology Magazine: "Semantic UI or Bootstrap? Discover the pros and cons of each framework", s Interneta, http://newtechmag.net/2020/03/29/semantic-ui-or-bootstrap-discover-the-pros-and-cons-of-each-framework/, 29. ožujka 2020.

[27] Google Cloud: "PostgreSQL vs SQL Server: What are the key differences?", s Interneta, https://cloud.google.com/learn/postgresql-vs-sql, 2023.

[28] Von Harz, T.: "VS Code vs Sublime Text – Which One Should You Choose?", s Interneta, https://tylerthetech.com/vs-code-vs-sublime-text-which-one-should-you-choose/, 25. svibnja 2023.

[29] The Linux Foundation: "10 Years of Git: An Interview with Git Creator Linus Torvalds", s Interneta, https://www.linuxfoundation.org/blog/blog/10-years-of-git-an-interview-with-git-creator-linus-torvalds, 6. travnja 2015.

[30] Jelenković, L.: "Osnovno o Git-u", Laboratorijske vježbe, Fakultet elektrotehnike i računarstva, Zagreb, http://www.zemris.fer.hr/ leonardo/oszur/lab/git-osnovno.html, 28. ožujka 2018.

Sažetak

U ovom radu cilj je bio izraditi web aplikaciju koja omogućuje dodavanje sastanaka u poslovni raspored, uz automatsko izračunavanje udaljenosti između lokacija sastanka. Tokom izrade ovog projekta, unutar VS Code IDE-a, korišteni su: Rocket razvojni okvir, React biblioteka, Semantic UI knjižnica, PostgreSQL baza podataka i Git kontrola toka. Na web aplikaciji korisnici mogu nadodavati sastanke, dopisivati se s pružateljima usluga, plaćati i pregledavati plaćene sastanke. Pružatelji usluga mogu vidjeti povijest plaćanja korisnika, dodavati sastanke i označavati plaćene sastanke. Web aplikacija je zbog svih ovih funkcionalnosti jako korisna jer omogućuje efikasnije upravljanje radnim vremenom pružatelja usluga i jednostavnije dogovaranje usluga od strane korisnika.

Ključne riječi: Web aplikacija, Rust, Rocket, React, Google API

Abstract

In this work, the goal was to create a web application that allows adding meetings to the business schedule, with automatic calculation of the distance between locations of meetings. During the creation of this project, within the VS Code IDE, the following were used: Rocket development framework, React library, Semantic UI library, PostgreSQL database and Git flow control. On the web application, participants can add meetings, correspond with service providers, pay and view paid meetings. Service providers can view a user's payment history, add appointments and mark paid appointments. The web application is useful due to all these functionalities, as it enables more efficient management of the service provider's working hours and simpler arrangement of services by the user.

Keywords: Web application, Rust, Rocket, React, Google API