RAZVOJ PROGRAMSKOG RJEŠENJA ZA SPAJANJE POTENCIJALNIH KLIJENATA S POLIKLINIKAMA

Soldan, Timon

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

RAZVOJ PROGRAMSKOG RJEŠENJA ZA SPAJANJE POTENCIJALNIH KLIJENATA S POLIKLINIKAMA

Timon Soldan

Zagreb, veljača 2023.

Student vlastoručno potpisuje Završni rad na prvoj stranici ispred Predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 28. veljače 2023

Predgovor

Prvo se želim zahvaliti mentoru Danielu Beleu na nevjerojatnom doprinosu u svojem obrazovanju i dubinskom znanju prenesenom na ovom studiju. Bilo je veliko iskustvo biti pod njegovim mentorstvom na Visokom učilištu Algebra. Njegova otvorenost za pitanja, ideje i davanje nesebičnih savjeta značajno su pridonijeli mojem razvoju kao programera. Njegovo znanje i mogućnost objašnjavanja kompleksnih principa, arhitektura i apstrakcija na jednostavan način omogućile su mi dodatno napredovanje u ovom zanimanju. Želim se zahvaliti i svojoj djevojci, obitelji, prijateljima i kolegama na podršci tijekom godina studiranja i usavršavanja. Bilo mi je zadovoljstvo studirati na Visokom učilištu Algebra i konstantno dobivati i širiti nova znanja iz područja programskog inženjerstva.

A Gradišćanska 24, HR-10000 Zagreb
T (01) 2222 182
E student@algebra.hr
www.algebra.hr

ALGEBRA

Temeljem članka 8. Pravilnika o završnom radu i završnom ispitu na preddiplomskom studiju Visokog učilišta Algebra sačinjena je ova

Potvrda o dodjeli završnog rada

kojom se potvrđuje da student Timon Soldan, JMBAG 0321010035, OIB 11064460593 u šk. godini 2021./2022., studij: Primjenjeno računarstvo - Preddiplomski studij, smjer: Programsko inženjerstvo, od strane povjerenstva za provedbu završnog ispita, dana 23.12.2021. godine, ima odobrenu izradu završnog rada

s temom: **Razvoj programskog rješenja za spajanje potencijalnih klijenata s poliklinikama**

i sažetkom rada: Student će svojim radom prikazati napredno korištenje inovativnih tehnologija u kontekstu razvoja sustava za povezivanja klijenata sa klinikama. Rezultat ovog završnog rada je sustav koji omogućuje interaktvino i napredno koristenje informacijskog sustava u svrhu pomoći pacijentima.

Mentor je: Daniel Bele.

Odobrenjem završnog rada studentu je omogućen upis kolegija "Izrada završnog projekta/Praksa" te je sukladno članku 8. Pravilnika o završnom radu i završnom ispitu dužan najkasnije do početka nastave ljetnog semestra u sljedećoj školskoj godini, uspješno obraniti završni rad uspješnim polaganjem završnog ispita.

U protivnom student može zatražiti novog mentora/icu i temu te ponovo upisati kolegij "Izrada završnog projekta/Praksa" budući da rad koji nije predan i obranjen na završnom ispitu u roku određenom Pravilnikom završnom radu i završnom ispitu prestaje vrijediti. Izrada novog završnog rada se izvodi sukladno rokovima određenima za školsku godinu u kojoj je studentu određen novi mentor/ica i dodijeljen novi završni rad.

Potpis studenta:	Potpis mentora:	Potpis predsjednika povjerenstva:

Ova potvrda izdaje se u 4 (četiri) primjerka od kojih 3 (tri) idu kao prilog završnom radu.

Sažetak

U ovom završnom radu se predstavlja inovativno rješenje za efikasnije upravljanje naručivanjem pacijenata, odnosno povezivanje potencijalnih pacijenata s poliklinikama u Republici Hrvatskoj. Utvrđeni su nedostaci trenutnog načina naručivanja pacijenata i ponuđen je prijedlog kako bi se ti problemi mogli uspješno riješiti. Time bi se olakšao cjelokupan način naručivanja kako za same pacijente, tako i za poliklinike. Arhitektura sustava se sastoji od klijentskog sloja, poslužiteljsko-podatkovnog sloja te objašnjenja korištenih programskih jezika, tehnologija i okruženja za funkcioniranje same aplikacije. Implementirano je programsko rješenje u obliku mobilne aplikacije za pacijente i poliklinike te web aplikacije i REST API.

Ključne riječi: mobilna aplikacija, web aplikacija, naručivanje pacijenata, poliklinika, programski jezik, programsko rješenje

This final thesis presents an innovative solution for efficient patient scheduling by connecting potential patients with clinics in the Republic of Croatia. The shortcomings of the current patient scheduling method have been identified, and a proposal has been made to effectively solve these problems and ease the entire scheduling process for both patients and clinics. The system architecture consists of a client layer, a server-data layer, and explanations of the programming languages, technologies, and environments used for the application's functioning. A software solution has been implemented in the form of a mobile app for patients and clinics, a web application and a REST API.

Keywords: mobile application, web application, patient scheduling, clinic, programming language, software solution

Sadržaj

1. Uvod

U današnjem svijetu tehnologija igra sve važniju ulogu u svakodnevnom životu, a jedno od područja koje može profitirati od tehnološkog napretka je upravo zdravstvo. Naručivanje pacijenata za preglede u poliklinikama je proces koji može biti zahtjevan i zbunjujući za pacijente, a također i za poliklinike koje moraju organizirati svoje resurse i što efikasnije poslovati. U ovom završnom radu predstavlja se inovativno rješenje za efikasnije upravljanje naručivanjem pacijenata u Hrvatskoj. Prijedlog se temelji na spajanju potencijalnih pacijenata s poliklinikama te smanjivanju nedostataka trenutnog dugotrajnog i kompleksnog načina naručivanja. Ova aplikacija će mnogo značiti za sve uključene strane, jer će olakšati cjelokupan proces naručivanja za pacijente i poliklinike. Cilj je omogućiti pacijentima da se lakše naruče na svoje preglede bez dugih čekanja na telefonske pozive i zbunjujućih procesa. Pronalazak poliklinike s traženom uslugom bit će lakši zbog usporedbe te poliklinike s drugima koje nude sličan ili isti tretman, odnosno pregled. Poliklinike će, s druge strane, imati bolju organizaciju i uvid u svoje raspoložive i popunjene termine za preglede te će moći efikasnije upravljati svojim resursima. Uz sve navedeno, poliklinika također dobiva pojednostavljeno interno vođenje termina klijenata na osnovu uzetog datuma i vremena termina i mogućnost daljnje komunikacije unutar same aplikacije. Programsko rješenje je efikasno zbog svog modularnog pristupa i korištenja naprednih tehnologija. Arhitektura sustava se sastoji od klijentskog sloja, poslužiteljsko-podatkovnog sloja te objašnjenja korištenih programskih jezika, tehnologija i okruženja za funkcioniranje same aplikacije. Implementirano je programsko rješenje u obliku mobilne aplikacije za pacijente i poliklinike te web aplikacije za administratora i REST API. Praktični doprinos rada je izrada jednog tehnološkog rješenja koje se sastoji od 3 komponente: REST API i Firebase, web aplikacija i mobilna aplikacija.

2. Nedostatci trenutnog načina naručivanja pacijenata i prijedlog rješenja

U sljedećim potpoglavljima detaljno će se opisati problem naručivanja pacijenata na preglede i termine u poliklinikama koji je potaknuo nastanak ovog rada. Također će biti detaljno opisana postojeća rješenja prethodno navedenog problema kao i njihova učinkovitost. S obzirom na brojne nedostatke postojećeg sustava, autor ovoga rada će predstaviti vlastiti prijedlog rješenja navedenog problema. Nakon toga, predstaviti će se vlastiti prijedlog rješenja problema od strane autora ovog rada, radi se o učinkovitijem i lakšem naručivanju pacijenata u poliklinike te lakšem funkcioniranju samih poliklinika u organizaciji rada s pacijentima. Bit će opsežno objašnjeno kako se ovaj prijedlog razlikuje od postojećih rješenja. Također će biti opisano koje su koristi i prednosti prijedloga ovoga rješenja. U konačnici, cilj ovog rada je detaljno analizirati problem i ponuditi kvalitetno i adekvatno rješenje za isti.

2.1. Problemi trenutnog naručivanja

Potrebno je prvenstveno objasniti trenutno stanje zdravstvenih ustanova u Republici Hrvatskoj te funkcionalnost njihovih sustava poslovanja. Prema podacima Hrvatske udruge poslodavaca, u 2018. godini bilo je 1170 privatnih zdravstvenih ustanova, ordinacija, poliklinika i klinika. Od navedenih 1170 ustanova, 235 se bavi stomatologijom, a 80 od tih 235 poliklinika se nalazi unutar grada Zagreba.[1]

To znači da osoba koja ima problema s zubima i živi u Zagrebu ima 80 različitih opcija za biranje stomatološke ordinacije. Kada ta osoba traži stomatološku ordinaciju, prvo će potražiti odgovarajuće lokacije ordinacija, zatim će usporediti cijene i prema tome odabrati najprikladniju. Međutim, proces naručivanja nije uvijek jednostavan. Mnogo puta se dogodi da ordinacije ne odgovaraju na telefonski poziv, tako da je osoba primorana zvati mnogo puta kako bi stupila u kontakt sa djelatnicima ordinacije. Pisanje e-mailova također nije najpraktičnije rješenje budući da se može čekati do 48 sati za odgovor, ovisno o gužvi i vremenskom periodu slanja.

Uz to, osoba koja zove polikliniku uglavnom ne zna radno vrijeme ordinacije, što uzrokuje neizvjesnost hoće li joj odgovarati ponuđeni datum i vrijeme termina. Samim time jednostavni proces naručivanja postaje višednevni zadatak.

2.2. Prijedlog rješenja za efikasnije upravljanje naručivanjem

Zdravlje i dobrobit svakom pojedincu zasigurno je životni prioritet. No, često se događa da ljudi odgađaju naručivanje pregleda i posjete doktorima zbog stresa i nedostatka vremena. Međutim, ovo rješenje je osmišljeno upravo s ciljem da se omogući jednostavnije i brže naručivanje na liječnički pregled bez dodatne napetosti.

Proces funkcionira tako da se pacijent registrira i ulogira u aplikaciju, a zatim kroz jednostavne filtere i pretraživanja pronalazi željenu zdravstvenu uslugu. Nakon toga, moći će pogledati sve poliklinike koje nude traženu uslugu i odabrati onu koja mu najviše odgovara. Kada se pacijent odluči za polikliniku, putem aplikacije će dobiti uvid u radno vrijeme poliklinike, kao i popis slobodnih termina na koje se može prijaviti.

Poliklinike također imaju korist od ovog rješenja, jer objavom radnog vremena i slobodnih termina liječničkog pregleda postaju efikasniji i znatno ubrzavaju svoje poslovanje. Pacijenti imaju mogućnost sam birati datum i vrijeme pregleda na osnovi dostupnosti, a poliklinike imaju pojednostavljeno interno vođenje termina pacijenata i mogućnost komunikacije unutar same aplikacije.

Ukratko, ova aplikacija ima za cilj pružiti pacijentima jednostavan i brzi proces naručivanja pregleda bez dodatnog stresa i komplikacija, kao i pomoći poliklinikama u učinkovitijem upravljanju naručivanjem.

2.3. Usporedba sa sličnim rješenjima

Na hrvatskom tržištu nedostaje jednostavno rješenje koje bi pacijentima omogućilo da se na učinkovit i jednostavan način naruče na liječnički pregled i tako dobiju odgovarajuću i pravovremenu medicinsku skrb. Dosadašnje aplikacije fokusirane su na pojedinačne

aspekte, poput hitne medicinske pomoći na zahtjev ili naručivanje lijekova, ali one ne kombiniraju sve funkcije u jedinstveni i korisnički prilagođeni proizvod na jednostavan način. Različite aplikacije također ne omogućuju pacijentima da uspoređuju lokacije i cijene poliklinika, te da samostalno odaberu termin pregleda na osnovu dostupnosti.

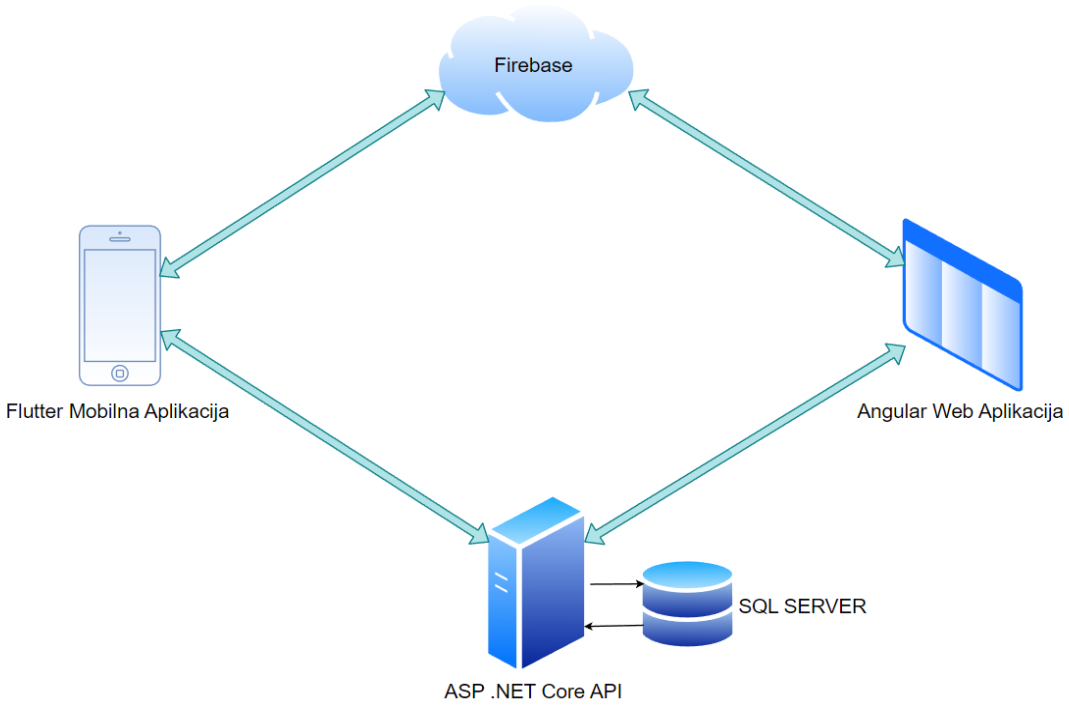
Stoga, opisana aplikacija koja kombinira sve ove funkcije predstavlja inovativno rješenje na hrvatskom tržištu, a njena dostupnost bi značajno olakšala život pacijentima i poslovanje poliklinikama. Kombinacijom funkcija naručivanja pregleda, usporedbe lokacija i cijena te omogućavanjem samostalnog odabira termina, pacijenti bi dobili jedinstveni alat za organiziranje svojih medicinskih potreba, bez korištenja različitih aplikacija i bez nepotrebnog stresa i gubitka vremena. Ovakvo rješenje bi bilo korisno i za liječnike te ostale zdravstvene djelatnike, smanjujući broj nepotrebnih poziva i povećavajući efikasnost njihove poslovne prakse. Ukratko, opisana aplikacija bi predstavljala jedinstven i važan dodatak hrvatskom tržištu medicinskih aplikacija te bi značajno poboljšala kvalitetu života pacijenata i olakšala poslovno funkcioniranje poliklinika.

3. Arhitektura sustava

Programsko rješenje se sastoji od tri komponente: REST API-a, Firebasea te web i mobilne aplikacije. Mobilna aplikacija (za pacijente/poliklinike) će biti razvijena u okruženju Flutter s programskim jezikom Dart, dok će web aplikacija (za administratore) koristiti Angular i programski jezik Typescript. Firebase za razgovornu (engl. *chat*) funkcionalnost, te također će biti korišten za autentifikaciju mobilnih i web aplikacija.

REST API povezuje cijelo tehnološko rješenje zajedno i razvijat će se korištenjem C# .Net core-a s relacijskom SQL bazom podataka te *Entity Framework Core* također poznat kao (engl. ObjectRelational Mapper, skraćeno ORM).

3.1. Skica i opis sustava



Slika 3.1 Shema sustava

Kao što je na skici sustava prikazano, sve su komponente na jedan ili drugi način međusobno povezane. Na primjer, radi se o mobilnoj aplikaciji za zdravstvene usluge. Korisnik će

prilikom prvog korištenja aplikacije odabrati želi li se registrirati kao pacijent ili kao poliklinika. Nakon toga će unijeti svoju e-mail adresu i željenu lozinku, a mobilna aplikacija će poslati upit na Firebase poslužitelj kako bi se provjerilo postoji li već korisnik koji je registriran s tom e-mail adresom te je li odabrana lozinka dovoljno sigurna. Ukoliko je sve u redu, Firebase će uzvratiti podatak ispravnosti lozinke.

Nakon toga, mobilna aplikacija će poslati HTTP POST zahtjev na REST API s unesenim podacima o registraciji, koji će zatim vratiti HTTP OK (200) status kod. Kada se korisnik pokuša prijaviti, mobilna aplikacija će se opet obratiti Firebase poslužitelju kako bi provjerila korisnikove podatke, a zatim će poslati upit na REST API kako bi provjerila je li ovo prva prijava korisnika. Na temelju toga, aplikacija će otvoriti odgovarajuću početnu stranicu ili dodatnu registracijsku stranicu, ovisno o tome je li se korisnik registrirao kao pacijent ili poliklinika.

3.2. Korišteni jezici, tehnologije i okruženja

U ovom projektu, korišteni su najnoviji tehnološki alati i jezici za izradu REST API-a, mobilne aplikacije i web aplikacije. REST API-om su omogućene interakcije između klijentskog i serverskog dijela aplikacije, a za njegovo razvijanje korišteni su C#, ASP.Net Core i *Entity framework Core* unutar programskog okruženja Visual Studio.

REST API je sofisticirani tip API-ja koji je izgrađen s posebnim skupom dizajnerskih načela koja promiču korištenje standardnih sučelja i bez tjelesne komunikacije. To ga čini idealnim rješenjem za razvoj web aplikacija koje zahtijevaju skalabilnost, interoperabilnost i fleksibilnost. REST API je API koji se pridržava dizajnerskih načela REST, ili arhitektonskog stila prijenosa reprezentacije stanja.

Uz to, za autentifikaciju se koristi Firebase, što osigurava sigurnost i privatnost korisničkih podataka. Firebase pruža brzu i jednostavnu autentifikaciju, što omogućuje da se korisnici brzo i lako prijave u aplikaciju, a istovremeno čuva njihove podatke u sigurnosti.

Mobilna aplikacija koja je izrađena za krajnje korisnike, razvijena je u Dart programskom jeziku uz pomoć Flutter platforme unutar Android Studija. Web aplikacija, namijenjena administraciji, korištenjem TypeScript programskog jezika i Angular platforme koristeći Visual Studio Code.

3.2.1. Klijentski sloj

Klijentski sloj čini korisničko sučelje koje korisnici koriste za interakciju s aplikacijom. On se sastoji od mobilne aplikacije za krajnje korisnike i web aplikacije za administratora. Mobilna aplikacija razvijena je u Dart programskom jeziku uz pomoć Flutter platforme i omogućava korisnicima jednostavno korištenje aplikacije putem njihovog pametnog telefona. Web aplikacija, s druge strane, razvijena je u TypeScript programskom jeziku i Angular platformi te omogućava administratoru brz i efikasan pregled i upravljanje podacima. Oba sučelja su dizajnirana tako da budu intuitivna i jednostavna za korištenje.

Flutter je moderna platforma za izradu mobilnih aplikacija koja je razvijena od strane Google-a. Ovaj razvojni alat koristi Dart programski jezik koji je dizajniran za razvoj performantnih aplikacija sa suvremenim funkcionalnostima. Flutter omogućuje developerima, odnosno razvojnim inženjerima da brzo i efikasno izrade interaktivne i animirana korisnička sučelja.

Infrastruktura fluttera se sastoji od sustava *widgeta*, Flutter enginea, *Dart* programskog jezika i drugih dodatnih komponenti. *Widget* sustav omogućuje razvojnim timovima da brzo i efikasno razvijaju interaktivna i atraktivna korisnička sučelja, dok Flutter *engine* upravlja prikazivanjem i interakcijom sa sučeljem. Dart programski jezik, koji se koristi u razvoju aplikacije, omogućuje brzo i efikasno programiranje.

Glavne karakteristike Fluttera su fleksibilnost i personalizacija. Razvojni inženjeri imaju veliku slobodu kod dizajna aplikacije, što omogućuje stvaranje unikatnog izgleda. Korištenjem widgeta, koje Flutter nudi, aplikacija može dobiti suvremeni izgled i funkcionalnost bez korištenja dodatnih biblioteka ili komponenata.

Flutter također omogućuje brz i lak razvoj, što je ključno za brzo lansiranje aplikacije na tržište. Brza *reload* funkcionalnost omogućuje razvojnim inženjerima da jednostavno testiraju i vrlo brzo vide rezultate promjena koje su napravili. Ova funkcija smanjuje potrebno vrijeme za razvoj i pojednostavljuje proces razvoja aplikacija.

Flutter predstavlja jedinstvenu platformu za razvoj mobilnih aplikacija koja kombinira performanse, fleksibilnost, brz razvoj i suvremene funkcionalnosti. [2]

Angular framework je jedan od najpopularnijih *frameworkova* za izradu web aplikacija. Angular je *open-source framework* koji je razvijen i održavan od strane Google-a, odlikuje se funkcionalnostima koje omogućuju jednostavnu, a opet efikasnu izradu dinamičnih i

interaktivnih web aplikacija. Angular koristi TypeScript programski jezik koji pruža dodatne prednosti u smislu sigurnosti i veće produktivnosti.

Angular funkcionira na principu *Model-View-Controller* arhitekture (skraćeno MVC) i sastoji se od različitih komponenti koje funkcioniraju zajedno kako bi osigurale kvalitetnu funkcionalnost aplikacije. *Model* čuva podatke, *View* prikazuje podatke korisniku, a *Controller* upravlja interakcijom između *Modela* i *Viewa*. Ova arhitektura osigurava da se svi dijelovi aplikacije međusobno ne miješaju i da se lako može razvijati i nadograđivati.

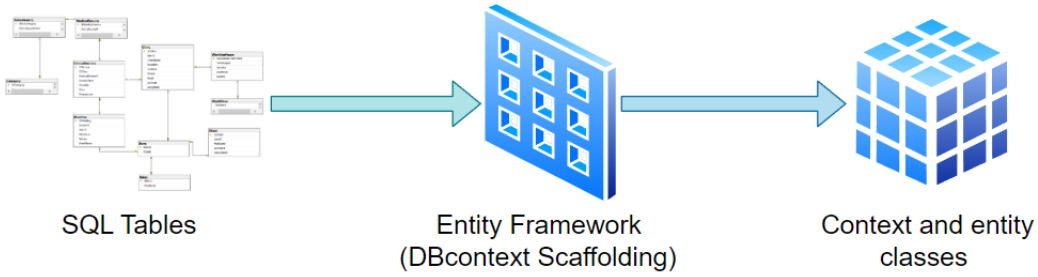
Angular također podržava različite funkcionalnosti koje omogućuju jednostavno kreiranje interaktivnih i dinamičnih korisničkih sučelja. Tu spadaju funkcionalnosti kao što je vezivanje podataka (engl. *data binding*), korištenje direktiva i servisa te podrška za mobilne aplikacije. Angular također ima dobro razvijenu zajednicu koja daje podršku i različite alate za izradu aplikacija.

Ukratko, Angular je snažan i funkcionalan radni okvir (engl. *framework*) za izradu web aplikacija koji omogućuje jednostavnu i efikasnu izradu dinamičnih i interaktivnih aplikacija. S obzirom na svoje prednosti, Angular je popularna opcija za razvojne inženjere koji žele napraviti kvalitetne i funkcionalne web aplikacije.[3]

3.2.2. Poslužiteljsko-podatkovni sloj

Poslužiteljsko-podatkovni sloj je srce aplikacije, koje čuva i upravlja svim podacima unutar aplikacije. REST API-jem su omogućene interakcije između klijentskog i poslužiteljsko-podatkovnog sloja, a za njegovo razvijanje korišteni su C#, ASP.Net Core i *Entity framework Core*. Ova tehnologija omogućuje efikasno i sigurno čuvanje podataka, a korištenjem relacijske SQL baze podataka i *Entity Framework Core* - ORM. Koristiti će se *database first* pristup uz pomoć *dbcontext* scaffoldinga što omogućuje brzu i jednostavnu integraciju i upravljanje podacima. Firebase će biti korišten za autentifikaciju korisničkih računa te će osigurati sigurnost korisničkih podataka i za funkcionalnost razgovora unutar aplikacije.

Ukratko, ovaj sloj će biti zadužen za pohranu, obradu te upravljanje podacima, kao i za omogućavanje jednostavne i brze komunikacije između klijenta i poslužitelja.



Slika 3.2 *Database first aproach*

Asp.NET Core API, kao dio Asp.NET Core platforme predstavlja alat koji omogućuje izradu moderne, brze i fleksibilne web aplikacije. On se temelji na principima programiranja REST i omogućuje programerima da izrade mikro servise za distribuiranje podataka i funkcionalnosti među različitim klijentskim aplikacijama.

Asp.NET Core API podržava više jezika, uključujući C#, F# i Visual Basic, što ga čini vrlo pristupačnim za programere različitih stručnih područja. Osim toga, Asp.NET Core API integrira se s *Entity Framework Core* kojim se omogućuje lako upravljanje podacima i jednostavniji rad s bazama podataka.

Asp.NET Core API se može izvoditi na različitim operacijskim sustavima, uključujući Windows, Mac i Linux, što ga čini vrlo prilagodljivim i pristupačnim za različite okruženja razvoja.

Asp.NET Core API predstavlja jedan od najboljih alata za razvoj moderne, brze i fleksibilne web aplikacije. Omogućuje programerima da izrade mikro servise, pri čemu koriste različite jezike, platforme i formate podataka.[4]

Firebase je platforma u oblaku (engl. *cloud*) za razvoj aplikacija koju razvija i održava Google. To je alat za brz razvoj aplikacija bez potrebe za dodatnim infrastrukturnim resursima. Firebase se koristi za pohranu podataka, autentifikaciju korisnika, komunikaciju s bazom podataka i mnoge druge funkcije potrebne za izgradnju aplikacije. Integracija Firebasea u aplikaciju učinila bi razvoj bržim i jednostavnijim, omogućujući razvojnom timu da se fokusira na funkcionalnosti aplikacije umjesto infrastrukture.

Firebase autentifikacija također je jedna od ključnih funkcionalnosti Firebase platforme. Autentifikacija korisnika omogućava aplikaciji da prepozna tko se prijavljuje i da osigura sigurnost podataka. Firebase autentifikacija podržava više načina prijave, uključujući e-mail i lozinku, društvene mreže i Google račun. Integracija Firebase autentifikacije u aplikaciju učinila bi da proces prijave korisnika bude brz i jednostavan, a također bi osigurala sigurnost podataka.[5]

4. Implementacija programskog rješenja

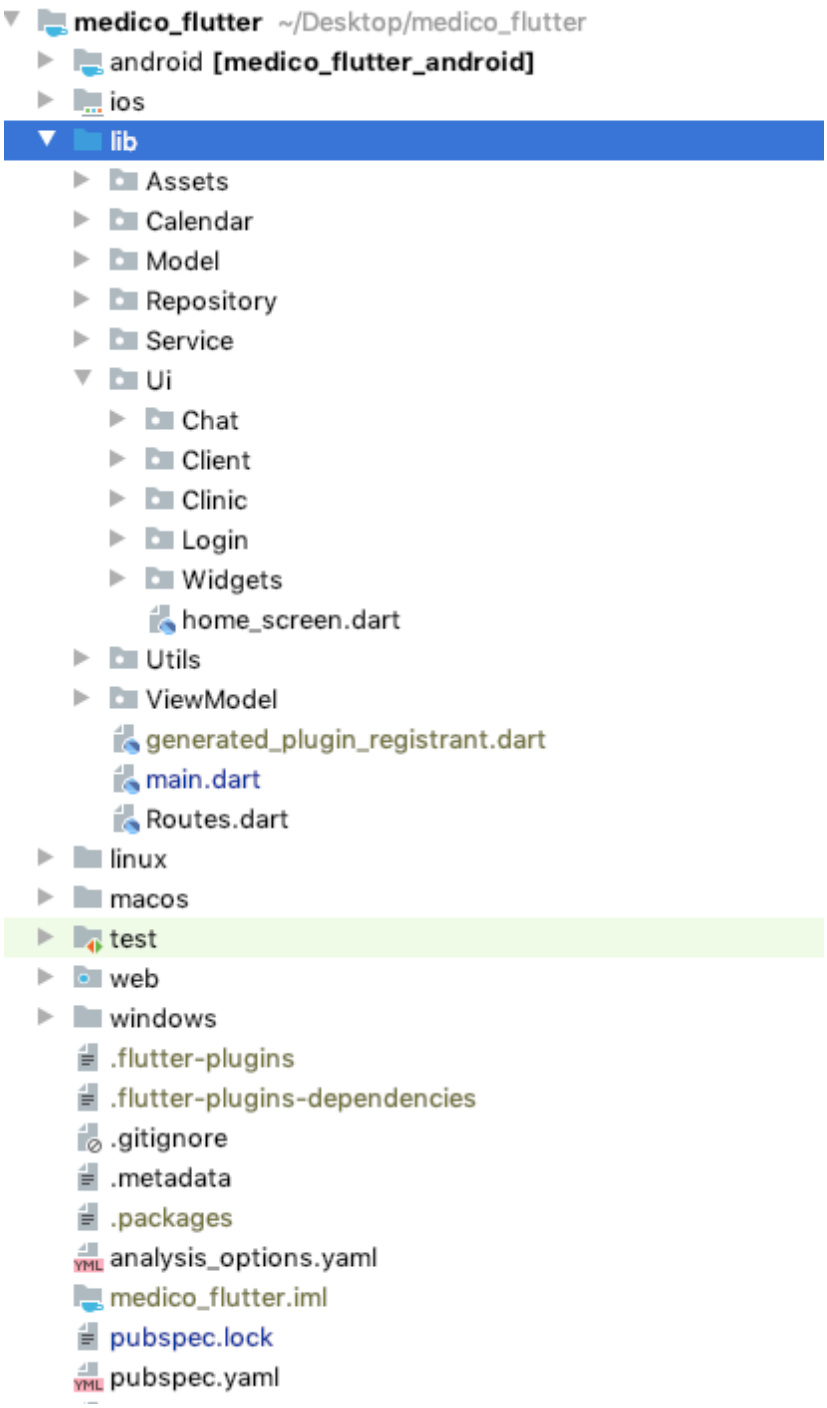
U ovom poglavlju će biti opisan cjelokupan proces izrade i implementacije svake komponente sustava. Namjera je dati što detaljniji prikaz procesa kako bi se učinilo jasnijim korištenje određenih tehnologija i jezika. Osim opisa procesa, bit će priloženi i primjeri koda i slike ekrana koji će dodatno pomoći u razumijevanju implementacije.

4.1. Razvoj komponenata

U ovom poglavlju bit će detaljno opisan razvoj komponenata sustava. Fokus će biti na opisivanju korištenih tehnologija, jezika i alata te načinu na koji su one integrirane kako bi se stvorila funkcionalna cjelina. Proces razvoja komponenata će biti objašnjen korak po korak i uključivat će primjere koda koji će pomoći u boljem razumijevanju. Ovim poglavljem želi se osigurati jasan uvid u tehničke aspekte izrade komponenata te dati potrebne informacije kako bi se svaka od njih lako implementirala u sustav.

4.1.1. Izrada mobilne aplikacije

Flutter aplikacija koja se ovdje opisuje napravljena je korištenjem Model-View-ViewModel (skraćeno MVVM) arhitekture uz *Repository pattern* za komunikaciju s API-jem i sloj usluga (engl. *service layer*) za pojednostavljivanje koda. Cilj ovog pristupa je osigurati čitljivost i lakoću održavanja koda.

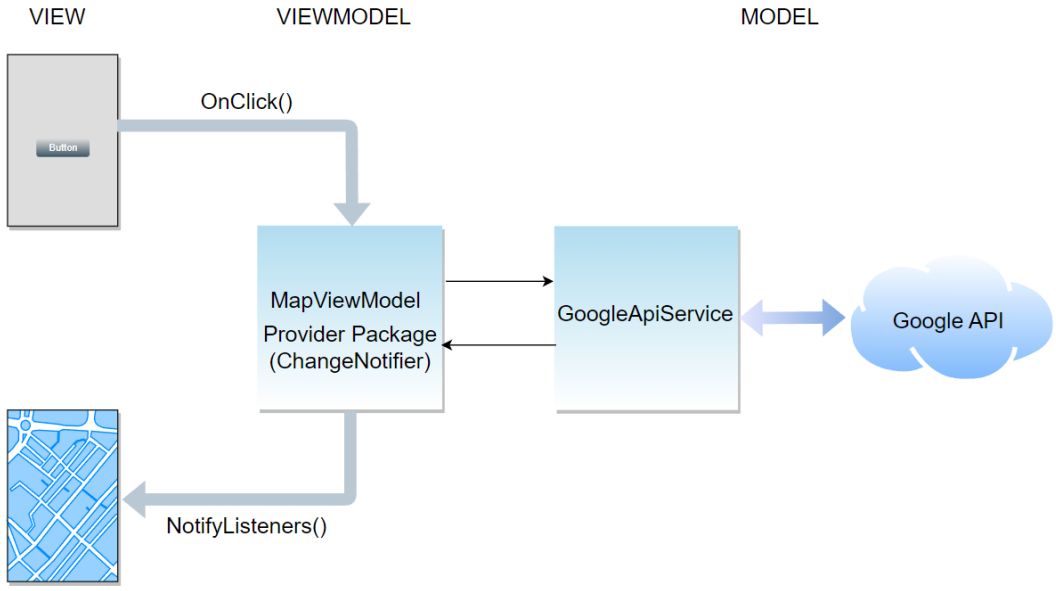


Slika 4.1 Arhitektura same Flutter aplikacije

Raspored datoteka unutar Flutter aplikacije organiziran je u nekoliko različitih komponenti sustava. Unutar direktorija *lib* smješteni su izvorni kodovi aplikacije, koji se sastoje od više različitih datoteka s proširenjima *.dart.* Te datoteke sadrže izvorni kod aplikacije, a mogu se pokretati iz glavne datoteke *main.dart.* Ostale važne datoteke su *pubspec.yaml* datoteka koja

sadrži informacije o ovisnostima i paketima koji se koriste unutar aplikacije. Datoteke za lokalizaciju nalaze se unutar direktorija */assets/locale*, dok su sve statičke datoteke poput slika, fontova i sl. smještene unutar direktorija */assets*. Slika 4.1 u dokumentaciji prikazuje primjer organizacije datoteka unutar Flutter aplikacije.

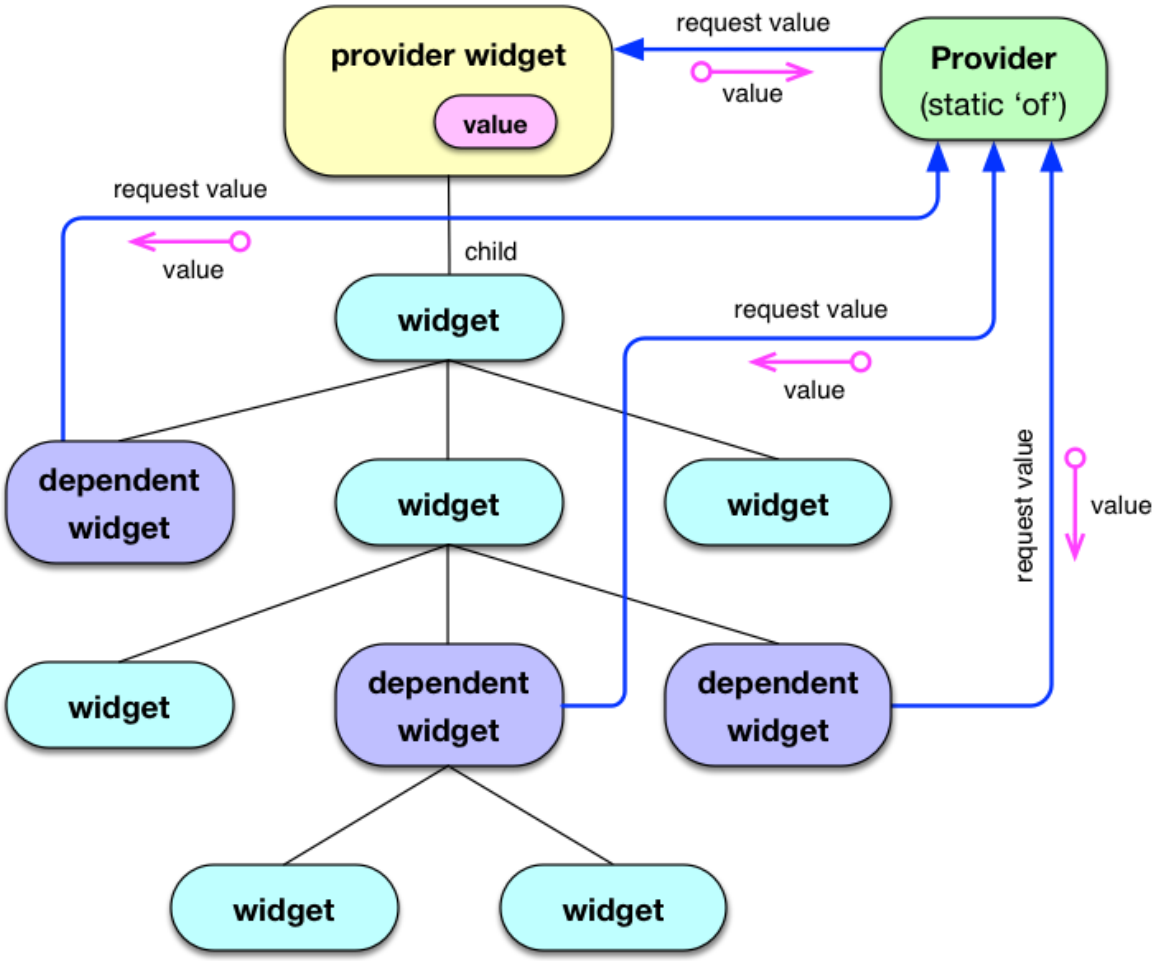
Korištenjem MVVM arhitekture, odvaja se logika aplikacije od korisničkog sučelja. Model predstavlja podatke i logiku poslovne aplikacije, *View* odražava izgled aplikacije, dok *ViewModel* prenosi informacije između *Modela* i *Viewa*. Ova arhitektura omogućuje da se poveća modularnost i čitljivost koda te da se lakše razvija i održava aplikacija.



Slika 4.2 Implementacija *Provider* paketa unutar MVVM

Repository pattern korišten je za komunikaciju s API-jem i za dohvaćanje podataka. *Repository* sloj omogućuje da se izolira kod za komunikaciju s API-jem te da se omogući lako mijenjanje toga koda bez utjecaja na ostatak aplikacije.

Sloj usluga (engl. *service layer*) namijenjen je pojednostavljivanju koda i pružanju dodatne funkcionalnosti. Ovaj sloj omogućuje da se centralizira logika aplikacije u jednom mjestu te da se lakše implementiraju dodatne funkcionalnosti.



Slika 4.3 State management logika u *Provider* paketu[6]

Upravljanje stanjem aplikacije (engl. *state management*) je ključan aspekt u razvoju bilo koje aplikacije, pa tako i u razvoju Flutter aplikacija. Jedan od popularnih paketa za upravljanje stanjem u Flutteru je *Provider* koji nudi jednostavan i efikasan način upravljanja stanjem aplikacije.

Uz pomoć *Provider* paketa, moguće je očuvati i prenijeti podatke u cijeloj aplikaciji te osigurati da se ti podaci koriste u svakom dijelu aplikacije. Na taj način se izbjegavaju problemi s globalnim varijablama i dijeljenjem podataka između različitih *widgeta*.

Provider koristi arhitekturu baziranu na *stream*-ovima i *Inherited Widgetu* kako bi se podaci distribuirali kroz aplikaciju. To znači da u slučaju da se podaci promijene, svi *widgeti* koji koriste te podatke će se automatski ažurirati.

Svaki od ovih dijelova pridonio je izradi jednostavne, čitljive i lako održive aplikacije koja funkcionira efikasno.

Flutter je ekstremno modularan te ujedno s time dolazi mnoštvo odabira u vezi paketa i zavisnostima. Flutterova solucija za upravljanje tih zavisnosti i paketa dolazi u obliku *Pubspec.Yaml*-a.

```
18  version: 1.0.0+1
19
20  environment:
21    sdk: ">=2.17.0 <3.0.0"
22
23  dependencies:
24    flutter:
25      sdk: flutter
26    flutter_localizations:
27      sdk: flutter
28
29    cupertino_icons: ^1.0.2
30    firebase_core: ^2.3.0
31    firebase_auth: ^4.1.5
32    firebase_messaging: ^14.1.0
33    firebase_storage: ^11.0.13
34    http: ^0.13.5
35    shared_preferences: ^2.0.15
36    toggle_switch: ^2.0.1
37    provider: ^6.0.5
38    place_picker: ^0.9.20-nullsafety
39    google_maps_flutter: ^2.1.1
40    geolocator: ^8.0.0
41    intl: ^0.17.0
42    syncfusion_flutter_calendar: ^20.4.44
```

Slika 4.4 Primjer *Pubspec.yaml* dijela iz samog rada

Pubspec.yaml se mora koristiti zato što on sadrži informacije o paketima koji su potrebni za rad aplikacije. Također, ako se žele dodati nove funkcionalnosti u aplikaciju, moraju se unijeti u *pubspec.yaml* i instalirati ih iz *pubspec.yaml*.

```
Future<void> loginUser(BuildContext context) async {
  try{
   bool loginAuth = await
_firebaseService.logIn(_emailController.text,
_passwordController.text);
```

```
      if(loginAuth == false ) return;
       var status = await
_api.checkIfFirstLogin("Users/Login",
_firebaseService.getUID());
       if(status.statusCode ==
244){Navigator.pushNamed(context,"Home");}
       else{
         !status.success ? StatusDialog.show(context, status)
: Navigator.pushNamed(context, status.result.roleName);
       }
   } catch(e){
       print(e);
       StatusDialog.showAuthError(context,e);
     }
   }
```

Kôd 4.1 Primjer loginUser funkcije iz `AuthViewModel` klase

Primjer koda prikazuje funkciju loginUser iz `AuthViewModel` klase. Funkcija prima `BuildContext` parametar koji omogućuje korištenje *build* metode za izgradnju *StatusDialog widgeta*. Funkcija također koristi `_emailController` i `_passwordController` varijable koje sadrže unose iz *TextField widgeta* u viewu. Nakon što se korisnički unos provjeri funkcijom `_api.checkIfFirstLogin` iz `ApiRepo` klase, provjerava se radi li se o prvom prijavljivanju bez dodatnih informacija. Na kraju, ovisno o odgovoru od `_api.checkIfFirstLogin`, prikazuje se odgovarajući *Dialog* ili se korisnik preusmjerava na daljnju stranicu. *Login* i registracija su miks Firebase autentifikacije i REST API-ja.

```
class SharedPrefs{
  static Future<SharedPreferences> get _instance async =>
_prefsInstance ??= await SharedPreferences.getInstance();
  static SharedPreferences? _prefsInstance;

  static Future<SharedPreferences> init() async {
    _prefsInstance = await _instance;
    return _prefsInstance!;
  }
```

```
  static Future<String>  getString(String key) async {
    var prefs = await _instance;
    return prefs.getString(key)!;
  }

  static Future<bool> setString(String key, String value)
async {
    var prefs = await _instance;
    return prefs.setString(key, value);
  }
}
```

Kôd 4.2 `SharedPrefs Singleton` klasa

SharedPreference je paket u Flutteru koji omogućuje jednostavno pohranjivanje i dohvaćanje podataka u aplikaciji. Koristi se za spremanje manjih količina podataka poput postavki korisnika, jezika ili bilo kojih drugih podataka koji su važni za rad aplikacije. *SharedPreference* nudi jednostavno i brzo čitanje i pisanje podataka te se podaci mogu pohraniti na trajnu memoriju uređaja. *SharedPreference* paket u danom projektu se koristi kako bi se olakšalo čuvanje i dohvaćanje osnovnih podataka o korisniku. Ovi podaci se spremaju lokalno na uređaju korisnika i mogu se lako dohvatiti u bilo kojem dijelu aplikacije.

Na slici (Slika 4.5) vidljiva je `SharedPrefs` *singleton* klasa koja se koristi za jednostavno čitanje i pisanje podataka u `SharedPreferences`-u. Klasa ima privatnu statičku metodu `_instance` koja kreira jedinstvenu instancu `SharedPreferences`-a. Metoda `init()` se koristi za inicijalizaciju i vraćanje instancu `SharedPreferences`-a. Metoda `getString()` vraća `String` vrijednost pohranjenu u `SharedPreferences`-u prema ključu, a `setString()` metoda pohranjuje `String` vrijednost pod određenim ključem. Sve metode asinkrono koriste `_instance` metodu kako bi osigurale da se kreira samo jedna instanca `SharedPreferences`-a.

`RoleManager` klasa je odgovorna za razlikovanje različitih uloga korisnika i upravljanje njihovim pristupom i dozvolama unutar aplikacije. Ovo omogućuje aplikaciji da prikaže različite ekrane i funkcionalnosti ovisno o ulozi korisnika.

U aplikaciji je integriran Google Maps servis kako bi se korisnicima omogućilo jednostavno pronalaženje lokacije poliklinike. Osim što pruža prikaz mape, ovaj servis omogućuje i postavljanje markera na željene lokacije te izračunavanje udaljenosti od trenutne lokacije

korisnika do poliklinike. Ove funkcionalnosti su implementirane korištenjem Google Maps Flutter paketa koji pruža jednostavne API pozive za prikazivanje mape i integraciju ostalih funkcionalnosti.

```
  @override
  Future<ActionResult<List<Meeting>>>
getAppointmentListClient(String endpoint, String id) async {
    ActionResult<List<Meeting>> result =
ActionResult<List<Meeting>>();
    try{
      var response = await
_httpHelper.httpGetSpecific(_mainUrl+endpoint,id);
      List<Meeting> appointments =
Meeting.fromJsonList(response.body);
      for(var ap in appointments) {
        if(ap.userId != null){
          ap.client = await
getClient("Clients/ID",ap.userId!).then((value) =>
value.result);
        }
      }
      result.setSuccess(appointments);
    }
    catch(e){
      print(e);
      result.setFailed(e.toString(), -1);
    }
    return result;
  }
```

Kôd 4.3 Primjer koda iz `ApiRepo` klase

Uz pomoć `RepoFactory` klase, `Http` klase i `ApiRepo/IApiRepo`, podaci se dohvaćaju iz API-ja. `RepoFactory` klasa služi za stvaranje instance `ApiRepo` klase, dok `Http` klasa služi za slanje HTTP zahtjeva i primanje HTTP odgovora. `ApiRepo` klasa implementira `IApiRepo` sučelje koje definira metode za dohvaćanje podataka iz API-ja. Korištenje ovih klasa omogućuje da se jednostavno i učinkovito dohvaćaju podaci.

```
 @override
Widget build(BuildContext context) {
  return Scaffold(
```

```
      appBar:  AppBar(
        title: Text(widget.appBarText),
        backgroundColor: Colors.transparent,
        elevation: 0,
      ),
    body: SfCalendar(
      backgroundColor: Color(0xff1d2334),
      minDate: DateTime.now(),
      maxDate: DateTime.now().add(Duration(days:
(14).toInt())),
      view: CalendarView.day,
      dataSource: MeetingDataSource(_appointments),
      timeSlotViewSettings: TimeSlotViewSettings(
        timeFormat: 'H:mm',
        startHour: 7,
        endHour: 20,
        timeInterval: Duration(minutes: 60),
      ),
      onTap: (CalendarTapDetails details) {
        if (details.appointments == null) return;
        final Meeting meeting = details.appointments!.first
as Meeting;
        _showReservationDialog(meeting);
      },
    ),
    );
  }
}
```

Kôd 4.4 Primjer koda za pozivanje Sfkalendar Widgeta

Flutterov deklarativni pristup programiranju omogućuje opisivanje onoga što bi korisničko sučelje trebalo prikazati u trenutnom stanju aplikacije. Uspoređujući trenutno stanje sa željenim stanjem, Flutter koristi vlastitu *widget* arhitekturu za brzu i efikasnu izgradnju korisničkog sučelja. Na primjer, ako se promijeni stanje aplikacije, Flutter neće mijenjati izravno korisničko sučelje, već će izračunati razlike u *widget* stablu i napraviti samo potrebne promjene.

SfCalendar paket se koristi za prikaz i upravljanje terminima kalendara unutar same aplikacije. Ovaj paket nudi različite poglede kalendara kao što su dan, tjedan, mjesec, godina

i raspored te ima podršku za prikaz događaja, prilagodljive teme i mnoge druge mogućnosti prilagodbe.

SfCalendar *widget* omogućuje prikaz kalendara s danim terminima u Flutter aplikaciji. U kodu (kôd 4.3) prikazan je primjer korištenja ovog widgeta. Uz nekoliko parametara koji se prosljeđuju, kao što su početni, krajnji datum te prikaz termina kalendara, SfCalendar kreira željeni kalendar. Također, *widget* omogućuje prilagođavanje izgleda i funkcionalnosti kalendara korištenjem različitih opcija i funkcija koje pruža.

```
class MeetingDataSource extends CalendarDataSource {
  MeetingDataSource(List<Meeting> source) {
    appointments = source;
  }

  MeetingDataSource.empty();

  @override
  DateTime getStartTime(int index) {
    return appointments![index].from;
  }

  @override
  DateTime getEndTime(int index) {
    return appointments![index].to;
  }

  @override
  String getSubject(int index) {
    return
displayAppointment(index)+getAppointmentTimes(index);
  }

  @override
  Color getColor(int index) {
    if(appointments![index].userId == null) return
Color(0xff236caf);
    return appointments![index].userId !=
SharedPrefs.getString("UserID") ? Color(0xffdd2d4a) :
Color(0xff1b332a);
  }
```

```
  @override
  bool isAllDay(int index) {
    return false;
  }

  String displayAppointment(int index){
    var name = appointments![index].eventName;
    if(appointments![index].userId == null) return name;
    if(appointments![index].client != null) return
appointments![index].client.firstName+'
'+appointments![index].client.lastName +' / '+ name;
    return appointments![index].userId !=
SharedPrefs.getString("UserID") ? name+' (Zauzeto)' :  name+'
(Vaš Termin)';
  }

  String getAppointmentTimes(int index){
    return ' / '+ DateFormat("HH-
mm").format(getStartTime(index)) +'-'+ DateFormat("HH-
mm").format(getEndTime(index));
  }
}
```

Kôd 4.5 `MeetingDataSource` klasa koja implementira `CalendarDataSource`

Klasa `MeetingDataSource` implementira `CalendarDataSource`, što omogućuje da se koristi u *SfCalendar widgetu* kako bi se prikazali željeni podaci. U kodu se definiraju metode poput `getStartTime`, `getEndTime` i `getSubject` koji omogućuju dobivanje početnog vremena događaja i završno vrijeme. Također se definiraju metode `getSubject` i `getColor` koje ovisno o korisniku postavljaju odgovarajuću boju na kalendaru i naslov događaja. Metode `displayAppointment` i `getAppointmentTimes` koriste se kako bi se prikazali detalji o događaju koji se prikazuje.

4.1.2. Izrada web aplikacije

Ova Angular aplikacija koristi Firebase autentifikaciju i Google Maps uslugu te se temelji na arhitekturi projekta koja se sastoji od komponenti, usluga i modula.

Autentifikacija korisnika ostvaruje se putem Firebase REST API-ja koji je integriran u aplikaciju te se osigurava siguran i jednostavan pristup aplikaciji. Uz to, aplikacija koristi Google Maps uslugu za prikaz geografske lokacije poliklinika.

Komponente su glavni gradivni elementi aplikacije, a svaka komponenta ima svoj HTML, CSS i TypeScript kod. Ove komponente su organizirane u modulima, koji se koriste za organizaciju funkcionalnosti u logičke cjeline.

Osim toga, aplikacija koristi usluge koje pružaju funkcionalnosti poput pozivanja API-ja, obrade podataka ili rad s datotekama. Te usluge se mogu koristiti u bilo kojoj komponenti koja ih treba, što poboljšava modularnost aplikacije.

Sve ove arhitektonske odluke omogućuju izgradnju skalabilne i održive aplikacije koja koristi Firebase autentifikaciju i Google Maps uslugu na efikasan način.

Aplikacija koristi *dependency injection* za učinkovitu organizaciju i dijeljenje koda između različitih komponenti. *Dependency injection (DI)* je dizajn obrazac koji omogućuje uvođenje ovisnosti u objekt izvana, što omogućava lakše testiranje, ponovnu upotrebu i fleksibilnost u kasnijoj fazi razvoja.

U ovom projektu se DI koristi kako bi se povezali različiti servisi s komponentama. Na primjer, firebase autentifikacija i Google Maps servis se koriste u različitim dijelovima aplikacije i DI pomaže da se ove ovisnosti ubrizgaju u komponente koje ih trebaju.

Ovaj projekt koristi Angularov mehanizam DI koji se temelji na ključu i vrijednosti. Ključevi su tipovi servisa, a vrijednosti su instance tih servisa. Kada se zatraži servis, Angular pronalazi pravu vrijednost prema ključu i ubrizgava je u komponentu.

```
import { Injectable } from '@angular/core';
import { AngularFireAuth } from '@angular/fire/auth';
@Injectable({ providedIn: 'root'})
export class AuthService {
 constructor(private afAuth: AngularFireAuth) { }
  async login(email: string, password: string): Promise<any>
{
    try {
      const userCredential = await
this.afAuth.signInWithEmailAndPassword(email, password);
      return userCredential.user;
```

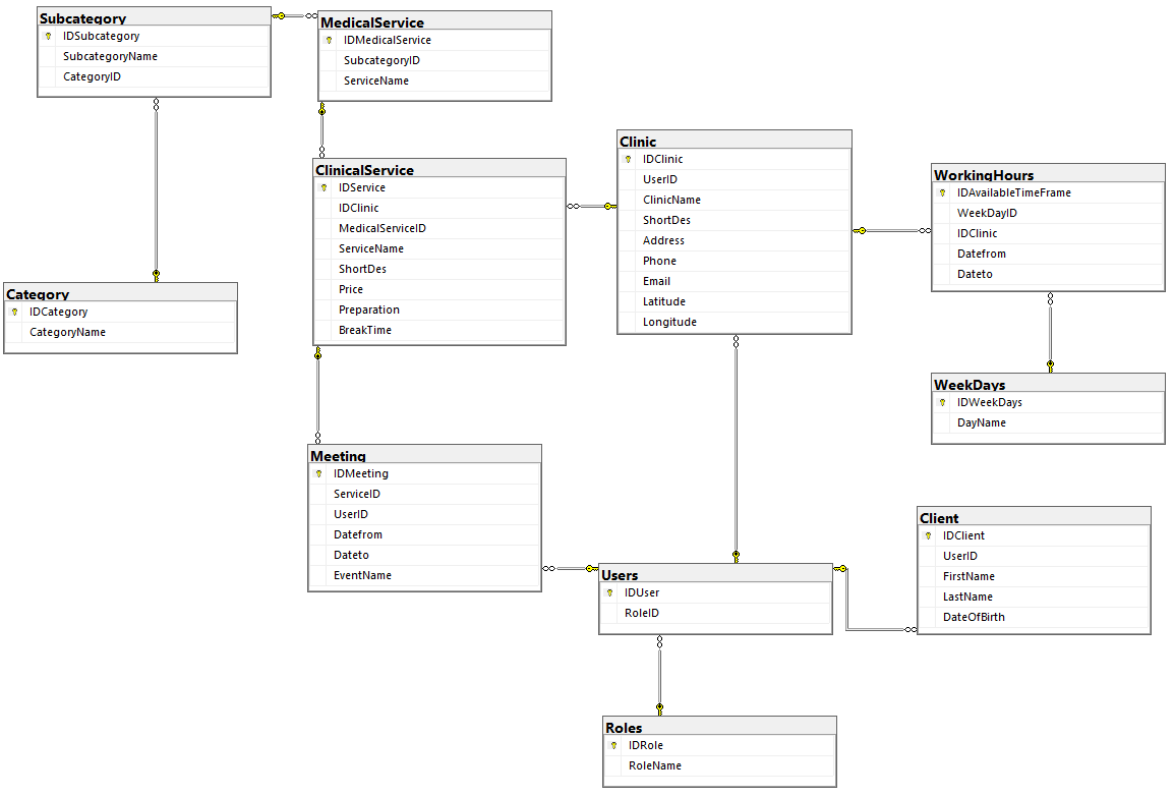
```
    } catch (error) {
      console.error('Error signing in', error);
      throw error;
    }
  }

  async logout(): Promise<void> {
    try {
      await this.afAuth.signOut();
    } catch (error) {
      console.error('Error signing out', error);
      throw error;
    }
  }
}
```

Kôd 4.6 Primjer DI unutar Angular projekta

U klasi `AuthService` primijenjen je DI na nekoliko načina. Prvo, postoji dekorator `@Injectable`, koji govori Angularu da se ovu klasu može ubrizgati s ovisnostima. Zatim konstruktor koji prima jedan parametar: `private afAuth: AngularFireAuth`. Ovdje se događa stvarno ubrizgavanje ovisnosti. AngularFireAuth usluga je deklarirana kao ovisnost `AuthService`-a te je prosljeđena instanci `AuthService`-a kada se stvori.

Korištenjem DI na ovaj način, `AuthService` može koristiti `AngularFireAuth` uslugu za obradu autentifikacije korisnika, a da ne brine o tome kako se usluga instancira ili upravlja. Jednostavno deklarira ovisnost o `AngularFireAuth`-u u konstruktoru, a Angular okvir se brine za ostalo. To čini kod čišćim, modularnijim i lakšim za održavanje.

4.1.3. Izrada programskog sučelja REST



Slika 4.5 *Entity Relationship* dijagram baze podataka

SQL baze podataka su neizostavan alat u današnjem svijetu. Od velike je važnosti pravilno organizirati SQL bazu podataka jer to olakšava brzo i učinkovito pronalaženje, upravljanje i analizu podataka. Jedna od ključnih karakteristika dobro dizajnirane SQL baze podataka je dobro povezivanje tablica, što znači da su podaci raspoređeni u različitim tablicama, a povezani su kroz zajedničke atribute. Ovo omogućava da se podaci drže na jednom mjestu te da se brzo i lako pristupa podacima koji su povezani na temelju zajedničkih atributa. Samim time dobro organizirana baza podataka drastično ubrzava te ujedno i olakšava razvoj REST API-ja.

Slika zorno prikazuje kako baza sadrži 11 entiteta: *Roles, Users, Client, Clinic, Category, Subcategory, MedicalService, ClinicalService, Meeting, WeekDays, WorkingHours*.

Središnja tablica projekta su *Users*, koja služi kao osnova za sve druge tablice. Ona je odvojena od ostalih tablica kako bi se u slučaju željene nadogradnje novih vrsta korisnika mogla dalje referencirati bez ikakvih promjena u ostalim tablicama. ID korisnika u ovoj

tablici je zapravo `FirebaseID` koji se stvara prilikom stvaranja Firebase entiteta kako bi se kasnije lakše mogao referencirati.

Uloga korisnika u mobilnoj i web aplikaciji definirana je u tablici *Roles*.

Tablice *Clinic* i *Client* se popunjavaju na temelju odabira klijenata tijekom registracije i prijave.

Tablica *Clinic* sadrži sve osnovne podatke o poliklinici te dodatno ima parametre geografske širine i dužine kako bi se točno prikazivala na Google Maps servisu.

Tablica *WorkingHours* služi za pohranu radnog vremena klinike po danima.

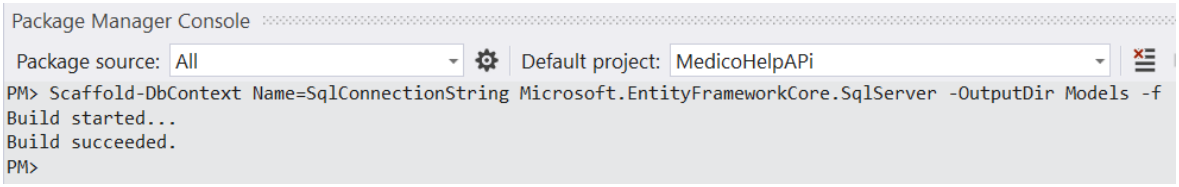
Tablice *Category*, *Subcategory* i *MedicalService* služe kao jednostavni filteri za pretraživanje podataka u tablici *ClinicalService*.

Tablica servisa *ClinicalService* prikazuje podatke o traženoj usluzi.

Tablica *Meetings* se koristi za punjenje *SfCalendar widgeta* mobilne aplikacije s terminima.

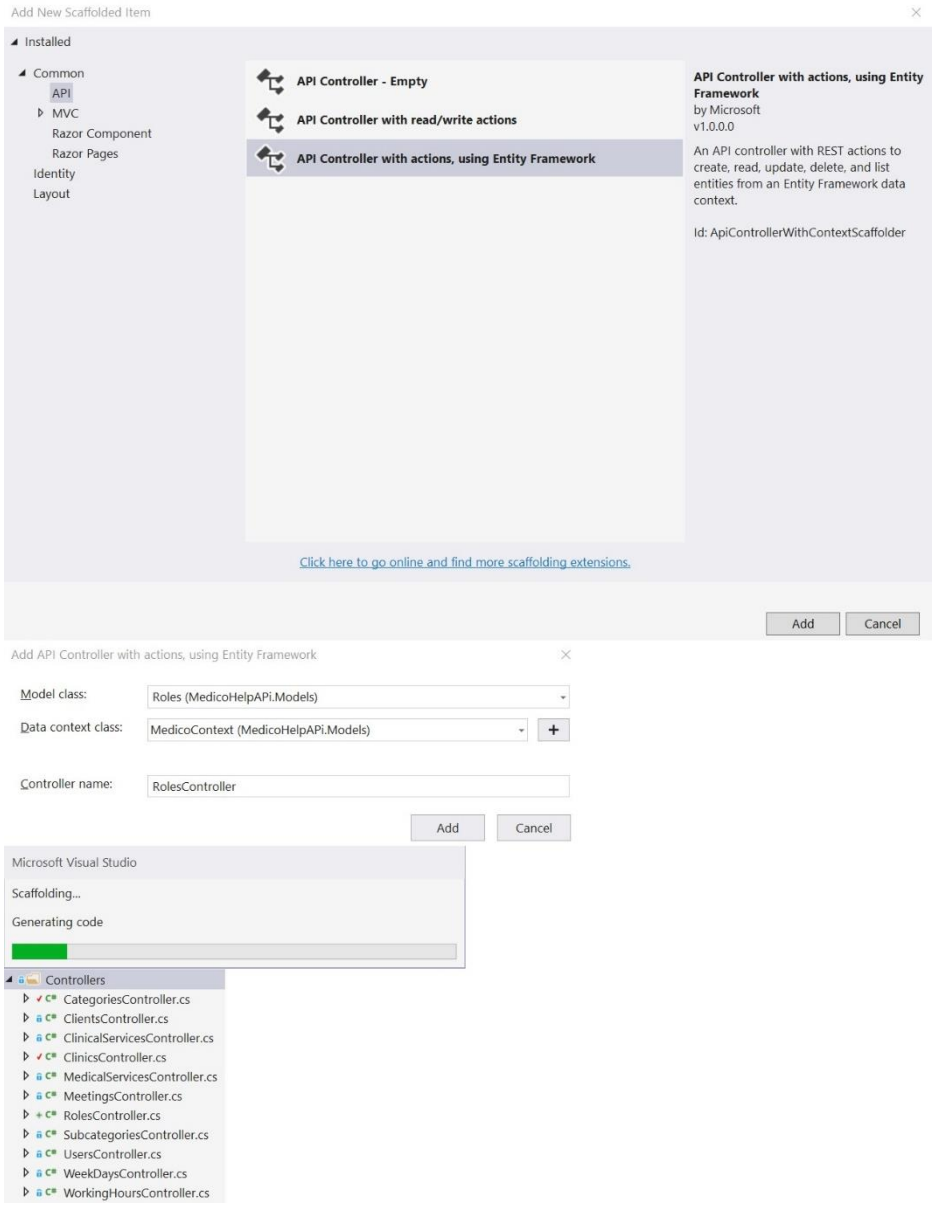
Za rad s bazom podataka, korištena je SQL baza podataka, što je najčešće korištena baza podataka na svijetu i ima široko podršku u industriji. Ova tehnologija omogućuje jednostavno upravljanje podacima, što je ključno za izradu web aplikacija.

U ovom radu korišten je *Entity Framework Core* kao glavni alat za rad s podacima u bazi podataka. *Entity Framework Core* – ORM omogućava izradu aplikacija s manjim kodom, bržim razvojem i većom produktivnošću. S *Entity Framework Core* se može definirati entitete koje odgovaraju tablicama u bazi podataka, a *Entity Framework Core* će se brinuti za sve radnje s podacima, uključujući upite, spremanje i ažuriranje podataka.



Slika 4.6 *Packet Manager Console* sa *Scaffold* komandom

Korišten je pristup *Database First* za rad s *Entity Framework Core*-om, što znači da se prvo kreirala baza podataka i zatim se u slučaju trenutnog REST API-ja koristio *DbContext Scaffolding* unutar *Packet Manager Console* kako bi automatski generirali klase *DbContext* i entiteta iz baze podataka. Ova tehnika smanjuje početni *boilerplate* kod i drastično ubrzava razvoj same aplikacije.



Slika 4.7 Generiranje kontrolera uz *Entity Framework*

Kontroleri su bili stvoreni prema postupcima prikazanom na slici, a za njihovu izradu bilo je potrebno instalirati *nuget* pakete za *Entity Framework* i automatsko generiranje koda. Nakon toga, samo nekoliko klikova je potrebno da se kontroler kreira pomoću Scaffolda, koristeći *Context* i *entitete* koje su prethodno definirani.

Jedna od ključnih značajki ASP.NET Core-a su njegovi API kontroleri, koji su zaduženi za obradu zahtjeva i generiranje odgovora.

```
        [HttpGet("Login/{id}")]
        public async Task<ActionResult<Roles>>
CheckIfFirstLogin(string id)
        {
            var user = await _context.Users.FindAsync(id);
            var role = _context.Roles.Where(u => u.Idrole ==
user.RoleId).Single();
            if (user == null) return NotFound();
            if (_context.Client.Any(e => e.UserId == id)) {
Response.StatusCode = 244; }
            if (_context.Clinic.Any(e => e.UserId == id)) {
Response.StatusCode = 244; }
            return role;
        }
```

Kôd 4.7. Primjer kontrolera za provjeru login-a

Ovaj kod predstavlja GET metodu nazvanu `CheckIfFirstLogin` koja se poziva kada se upita `Login/{id}`. Metoda prima ID kao parametar.

Unutar metode, prvo se dohvaća korisnik iz baze podataka koristeći njegov ID koji se prosljeđuje kao parametar metode. Zatim se dohvaća uloga korisnika koristeći njegov ID iz tablice *Roles*. Ako korisnik nije pronađen u bazi podataka, vraća se nepronađen (engl. *NotFound*) rezultat.

Zatim se provjerava ima li korisnik ID iz tablice *Client* i *Clinic*. Ako je to slučaj, postavlja se odgovarajući statusni kod odgovora na 244.

Konačno, vraća se uloga korisnika.

Ukratko, ova metoda služi za dohvaćanje uloge korisnika i provjeru postojanja korisnika u tablicama *Client* i *Clinic*.

API kontroleri su klase koje nasljeđuju `ControllerBase` klasu, a služe za obradu HTTP zahtjeva i vraćanje odgovora u formatu koji je prikladan za API komunikaciju (npr. JSON, XML). Sadrže niz metoda koje su odgovorne za obradu različitih `HTTP` metoda (*GET, POST, PUT, DELETE*, itd.) te metoda koje služe za vraćanje specifičnih podataka.

Hangfire nuget paket je alat za upravljanje i zakazivanje poziva metoda u .NET aplikaciji. Omogućuje izvršavanje zadataka u pozadini, kao što su slanje e-mailova, generiranje izvještaja, izvršavanje različitih obrada i zadatka u pozadini i drugo. *Hangfire* se integrira u .NET aplikaciju kroz konfiguraciju servisa i pozivanje metoda unutar aplikacije koje treba izvršiti.[7]

```
public void ScheduleMeetingDeletion(DateTime dateto, Guid
meetingId)
{
BackgroundJob.Schedule(() => DeleteMeeting(meetingId),
dateto);
}

public void DeleteMeeting(Guid meetingId)
{
    using (var db = new MedicoContext())
    {
        var meeting = db.Meetings.FirstOrDefault(m =>
m.Idmeeting == meetingId);
        if (meeting != null)
        {
            db.Meetings.Remove(meeting);
            db.SaveChanges();
        }
    }
}
```

Kôd 4.8 *Hangfire* funkcija za zakazivanje vremena brisanja

Kod koji je dan koristi *Hangfire* paket za zakazivanje poziva metode `DeleteMeeting` na dan određen parametrom `dateto` i sa ID-om sastanka `meetingId` kao parametrom. Ova metoda se poziva na pozadinskom threadu i neće ometati glavni tok aplikacije. Metoda uklanja sastanak iz baze podataka. Provjerava postoji li sastanak sa zadanim ID-om u bazi podataka. Ako postoji, sastanak se briše iz baze podataka. Ako ne postoji, ništa se ne događa.

Metoda `ScheduleMeetingDeletion` ima dva parametra. Prvi parametar je `dateto` koji označava vrijeme kada se metoda `DeleteMeeting` treba pozvati. Drugi parametar je `ID` sastanka koji se treba izbrisati. Kada se metoda `ScheduleMeetingDeletion` pozove, ona će pozvati metodu `DeleteMeeting` koristeći *Hangfire*.

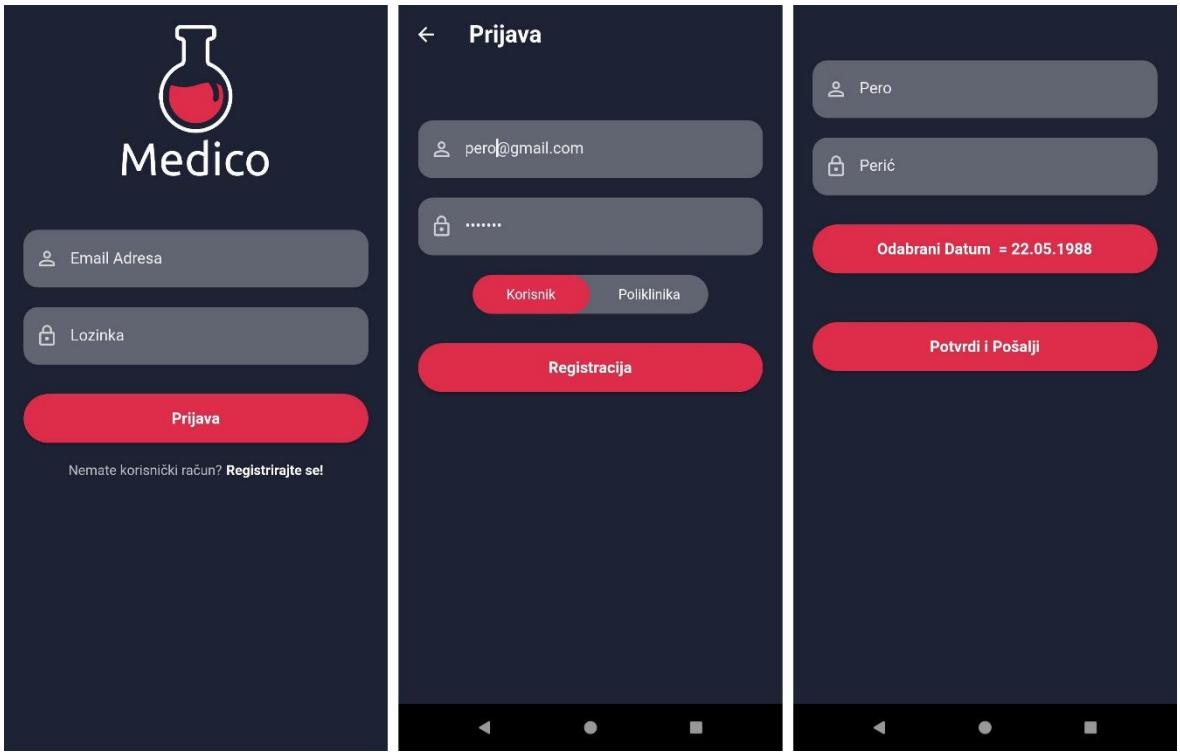
4.2. Opis gotovih funkcionalnosti

U idućem poglavlju detaljno će biti opisane sve funkcionalnosti koje su implementirane u okviru projekta. Svaka funkcionalnost će biti popraćena slikama ekrana, koje će vizualno prikazati kako izgleda korisničko sučelje i kako korisnik može koristiti tu funkcionalnost. Kroz detaljan opis funkcionalnosti, objasnit će se koraci potrebni za korištenje svake funkcije, kao i očekivani rezultati koje korisnik može očekivati prilikom korištenja te funkcije.

4.2.1. Mobilna aplikacija za klijente

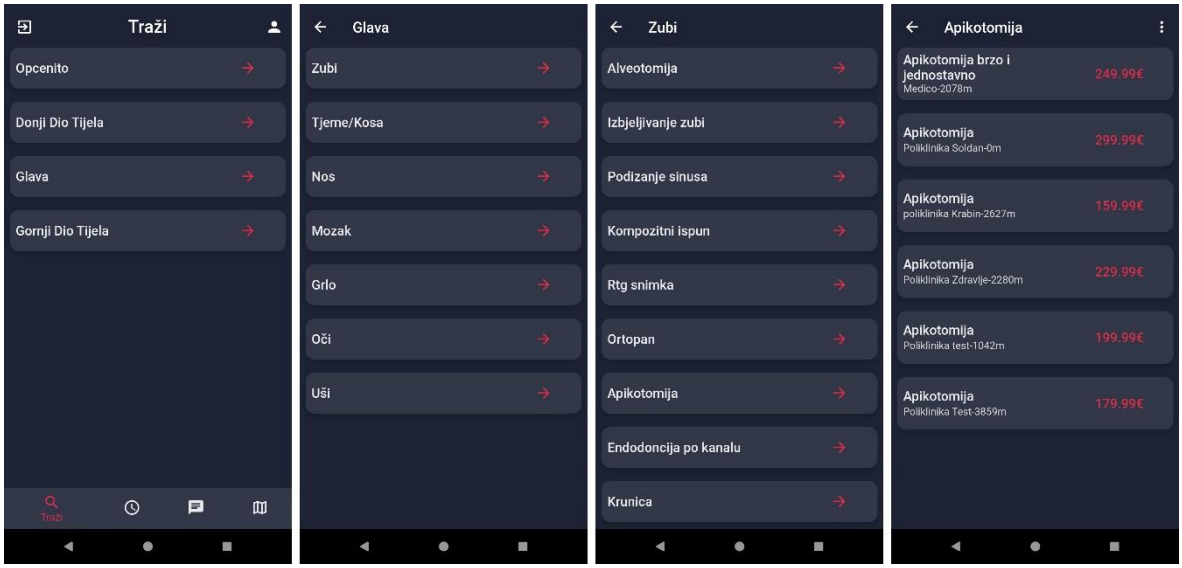
Pri prvom pokretanju aplikacije, aplikacija koristi *SharedPreference* paket kako bi provjerila je li već spremljen korisnik.

Ukoliko se korisnik već nalazi u memoriji uređaja, aplikacija će prikazati ekran koji odgovara njegovoj ulozi (obični korisnik/poliklinika). U slučaju da korisnik još nije spremljen, aplikacija će otvoriti ekran za prijavu kako bi se korisnik mogao prijaviti u sustav.



Slika 4.8 Primjer Registracije

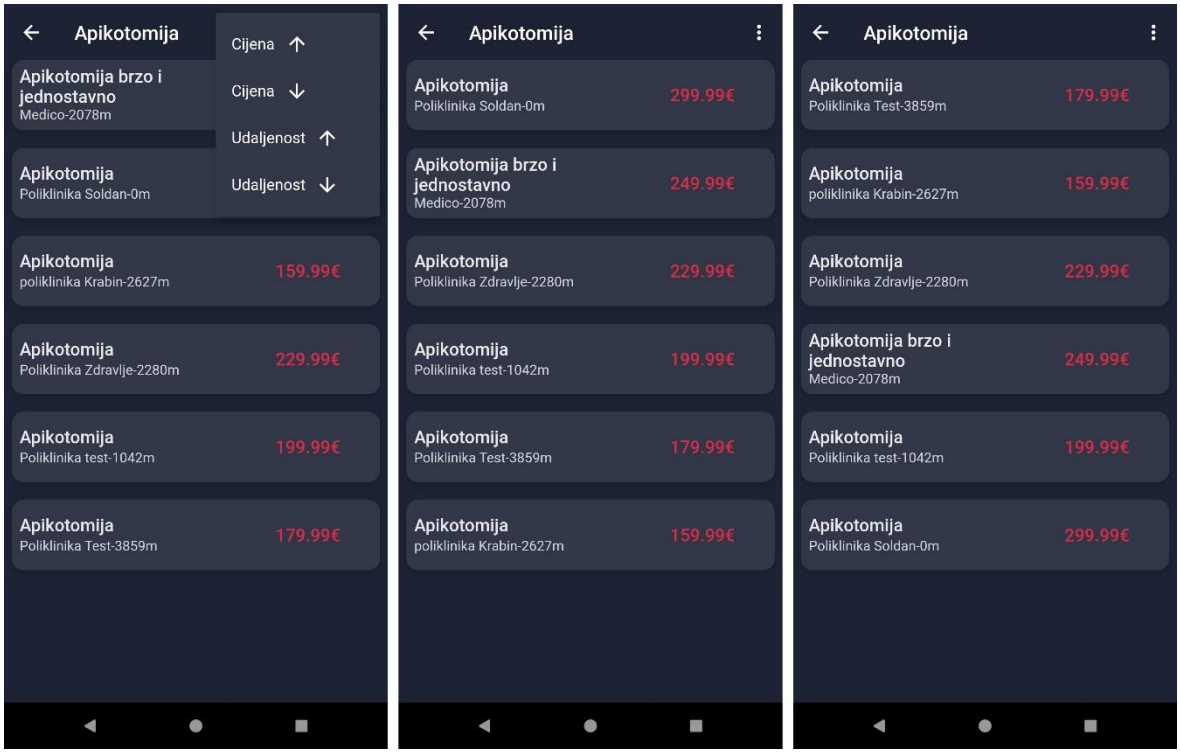
Ekran za prijavu korisnika sadrži polje za unos korisničkog *emaila*, polje za lozinku i gumb za registraciju. Za funkciju prijave korisnika, kao što je i prethodno rečeno (detaljan primjer u naslovu 3.1) koristi se miks Firebasea i vlastitog REST API-a. Zatim ekran za registraciju izgleda više manje isto uz dodatan prekidač za odabir Korisnika/Poliklinike, dok ekran za dodatne informacije ima u sebi 2 polja, jedno za ime, drugo za prezime te gumb za odabir datuma koji otvara prozor za odabir datuma uz korištenje *DateSelector widgeta*.



Slika 4.9 Pretraživanje dostupnih kategorija

Kada se korisnik ulogira, prvi element u donjoj navigacijskoj traci (engl. *bottom navigation bar*) je jednostavan pretraživač. Dizajn je jednostavan i pristupačan svakome. Odabir se vrši u redoslijedu glava - zubi - apikotomija - usluge koje nude poliklinike. Podaci se ispisuju uz pomoć `itemBuilder`-a i sastoje se od `Container`-a koji ima `child: ListTile` s tekstom i `onTap()` funkcijom.

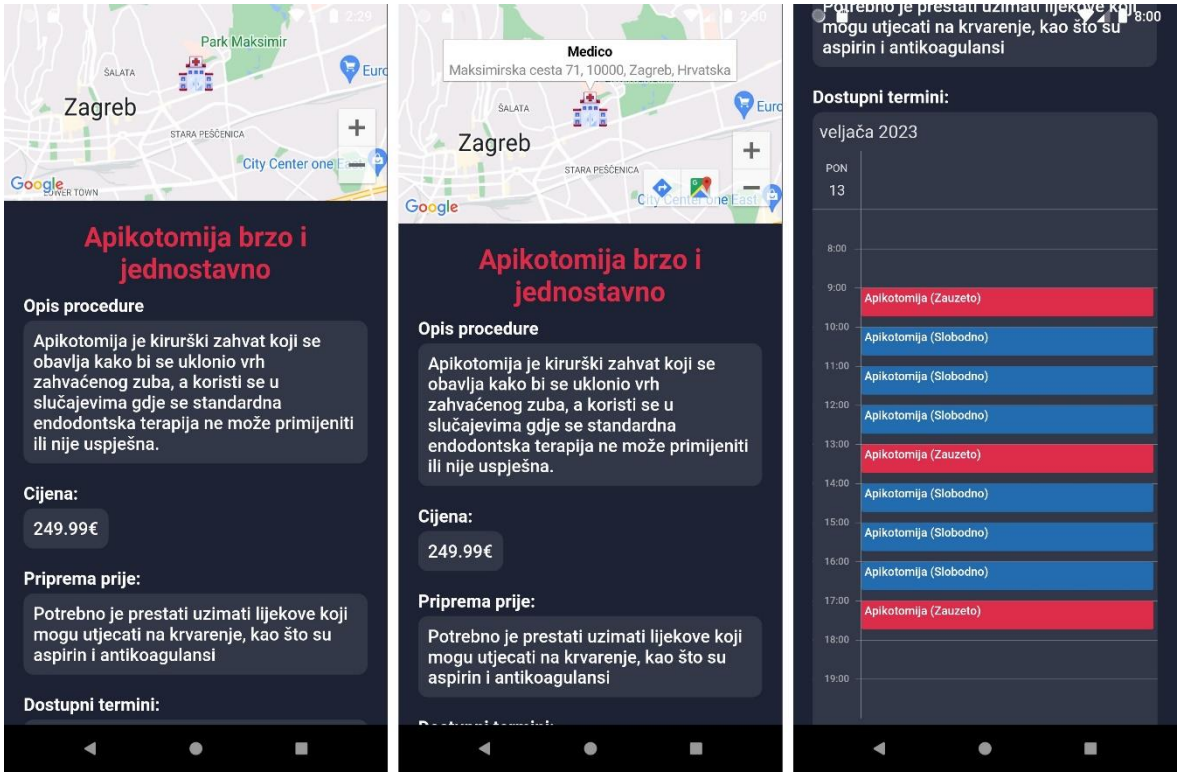
Podaci se dobivaju preko REST API-a s GET metodom. Svaka kategorija ima svoj jedinstveni UUID ID koji se koristi za pozivanje GET metode. Nakon toga, na temelju dobivenog UUID-a, poziva se potkategorija s API-ja te se podaci učitavaju u listu putem klase `ApiRepo` i funkcije `fromJsonList`. Uzeći u obzir kako dart ne podržava refleksiju, svaka klasa koje želi koristiti Json serijalizaciju/deserijalizaciju mora imati implementirane `fromJson` i `toJson` funkcije.



Slika 4.10 Primjer funkcionalnosti sortiranja

Nakon što korisnik dođe do željene usluge, prikazuju se nazivi usluga, nazivi poliklinika, njihova udaljenost od trenutne lokacije korisnika i cijena usluge. Udaljenost se računa pomoću Google Distance Matrix API-ja.

Filtri se otvaraju pomoću *widgeta* `PopupMenuButton`, koji se nalazi u `appBar`-u. Filtri funkcioniraju po principu sortiranja i usporedbe elemenata prema drugim elementima. Primjer linije koda koja opisuje sortiranje cijena uzlazno: `services.sort((a, b) => a.price.compareTo(b.price));`

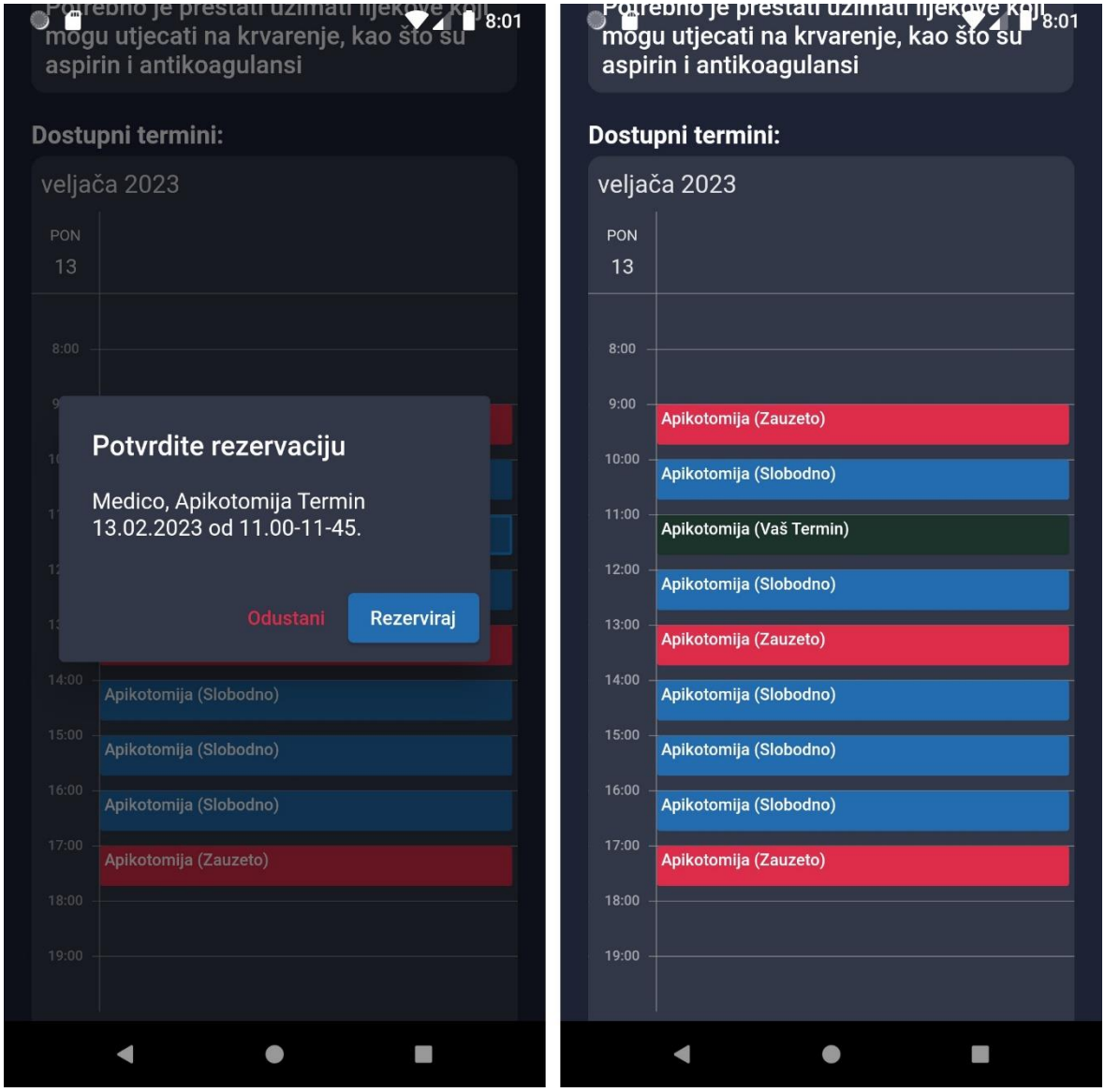


Slika 4.11 Detaljni prikaz usluge i termini

Detaljni zaslon medicinske usluge (`detailed_medicalservice_screen`) se otvara nakon odabira usluge na prethodnom zaslonu i sastoji se od `Scaffold`-a koji ima `SingleChildScrollView` element u tijelu. Tako se omogućava klizanje ekrana u oba smjera. Na vrhu zaslona nalazi se *GoogleMap widget* koji prima lokaciju iz klase `Clinic` kao parameter i *Marker* parametar unutar kojeg se može staviti posebna ikona zajedno s adresom i naslovom.

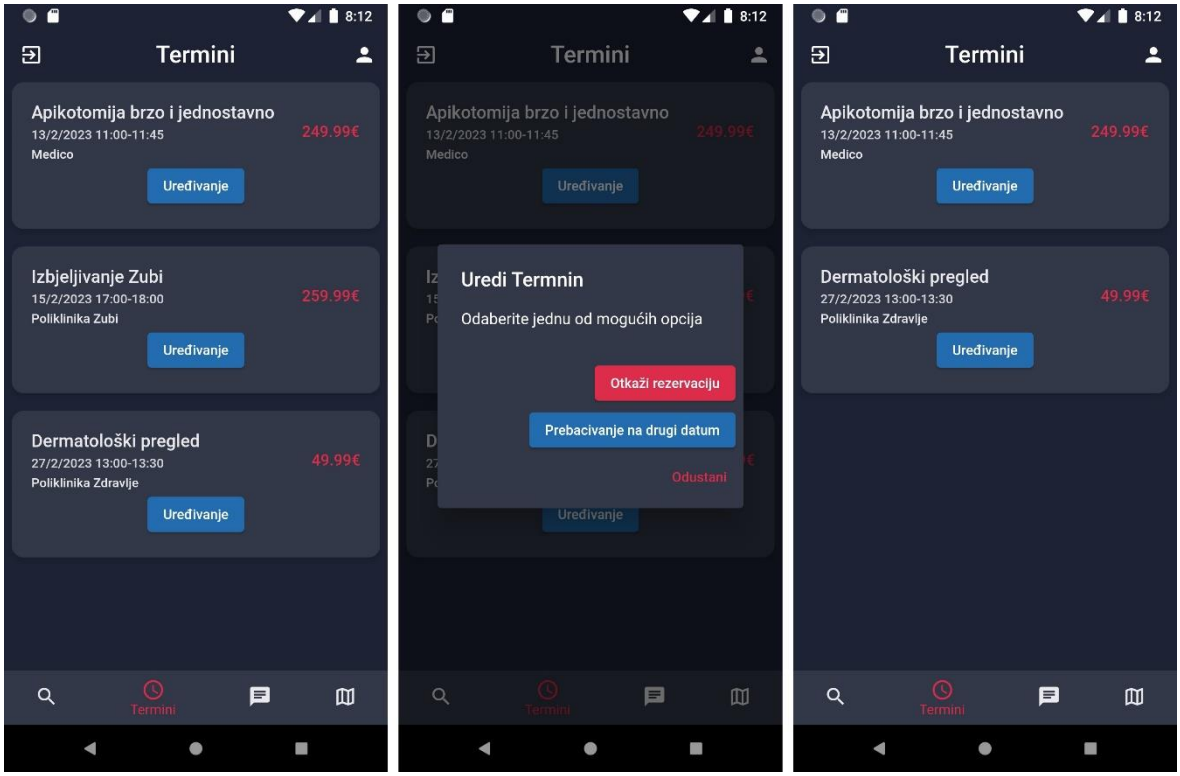
Klikom na ikonu otvara se informacijski prozor klinike s prikazom imena i adrese iz klase `Clinic`. Klikom na informacije otvara se `clinic_info_screen` koji je prikazan na

slici (Slika 4.17). Na dnu ekrana nalazi se *SfCalendar widget* koji sadrži termine za odabrani servis. *SfCalendar widget* omogućava horizontalno klizanje ekrana kako bi se pregledali termini u sljedećim danima.



Slika 4.12 Primjer rezerviranja termina

Klikom na slobodni termin, pojavit će se dijaloški okvir za potvrdu. Odabirom „odustani“, koristi se `Navigator.pop()` za izbacivanje posljednjeg elementa rute (engl. *route*) stoga (eng. *stack*), koji u ovom slučaju predstavlja dijaloški okvir. Ako se odabere „rezerviraj“, funkcija `bookAppointment` poziva se uz pomoć `SfCalendarViewModela`, API REPO-a i HTTP PUT zahtjeva na REST API. Ako je zahtjev uspješan, *provider* funkcija `notifyListeners()` se poziva, a SfCalendar se osvježava sa ažuriranim podacima.



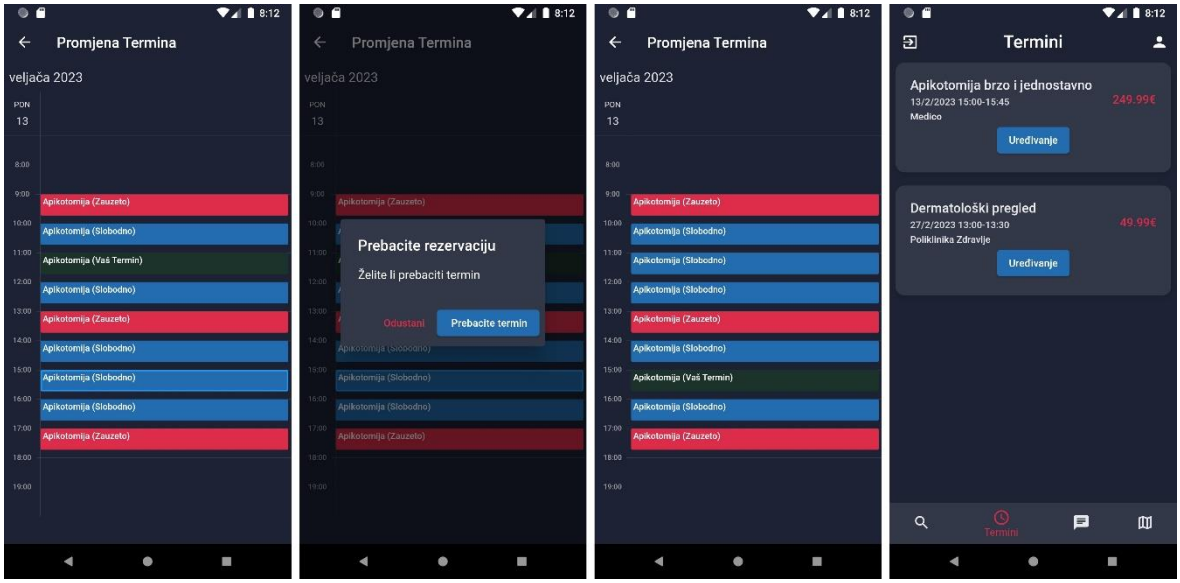
Slika 4.13 Primjer rezerviranih termina

Ekran `BookedAppointmentsScreen` predstavlja drugi element donje navigacijske trake (engl. *bottom navigation bar*) te prikazuje sve rezervirane termine korisnika pozivom REST API-ja s UserID-om, koji se sprema u `SharedPrefs` klasi nakon prijave.

```
  Future<List<Meeting>> getBookedAppointments(BuildContext
context) async {
    List<Meeting> appointments = <Meeting>[];
    try{
      var output = await
_api.getBookedAppointments("Meetings/Booked",SharedPrefs.getS
tring("UserID"));
      if(!output.success){
        StatusDialog.show(context, output);
      }
      appointments = output.result;
    }catch(e){
      print(e);
    }
    return appointments;
  }
```

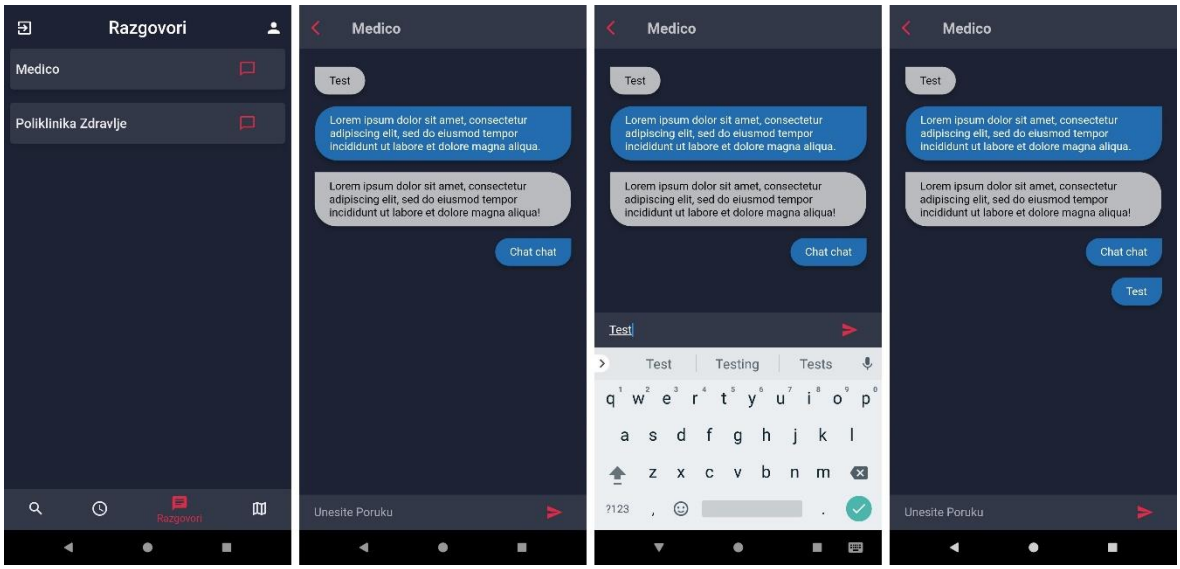
Kôd 4.9 `getBookedAppointments` funkcija

Podaci o rezerviranim terminima se prikazuju kroz popis (engl. *list*) koristeći `itemBuilder`. Brisanje termina se također ostvaruje kroz `ApiRepo` klasu uz HTTP PUT gdje se ažurira odabrani termin na način da se rezervirani korisnik briše.



Kôd 4.10 Promjena termina rezervacije

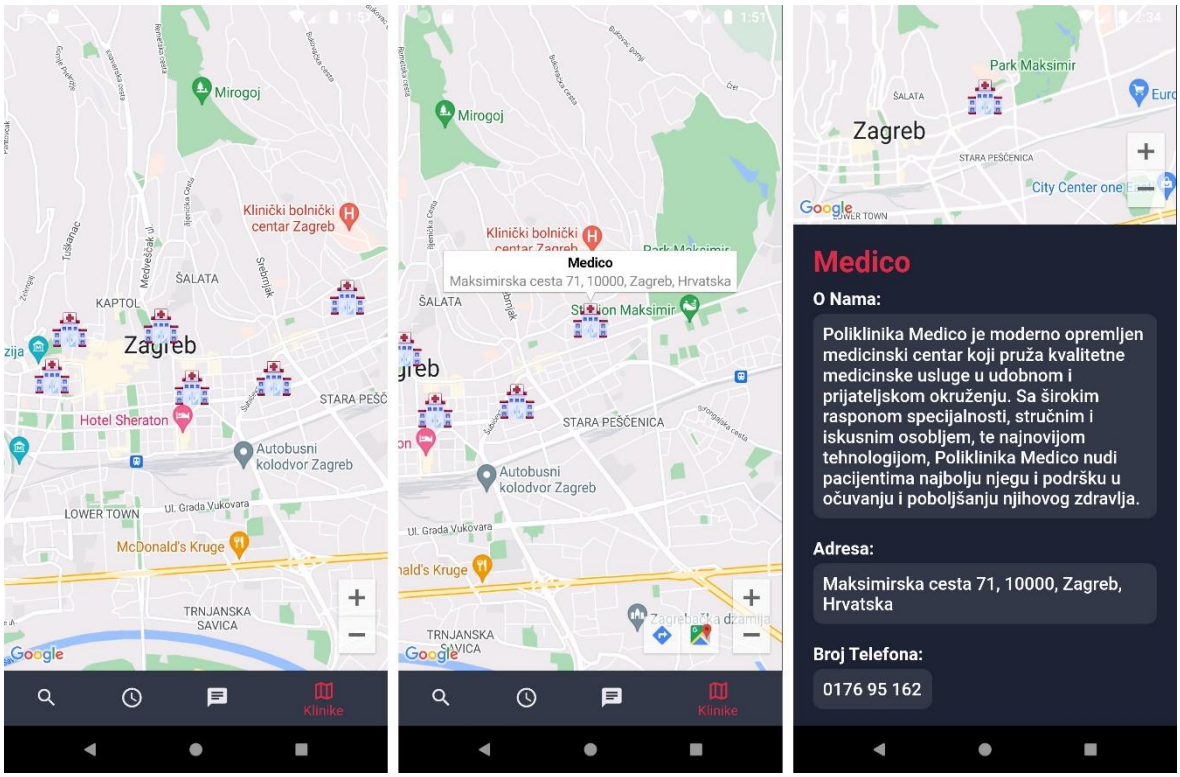
Klikom na promjenu termina dolazi se do *Sfcalendar widget*a koji je ovaj put učitan preko cijeloga ekrana. Prebacivanje rezervacije obavlja se uz `rescheduleAppointment` funkciju koja prima ID od prošlog *appointmenta* i trenutni `userID` te ga šalje uz HTTP PUT na API, u slučaju da je signal *OK SfCalendar* se ažurira i povratom nazad vidljivo je da se i `BookedAppointmentsScreen` ažurirao s novim podacima.



Slika 4.14 Chat funkcionalnost

Trenutna slika pokazuje ekran razgovora (engl. *chat screen*), koji je treći element donje navigacijske trake (engl. *bottom navigation bar*). U slici (4.15) prilikom rezerviranja termina usluge također se odvila i funkcija `createChatRoom` koja prima u parametrima ID od klijenta i ID od poliklinike te kreira novi `chat_room` u *Cloud Firestore* bazi podataka.

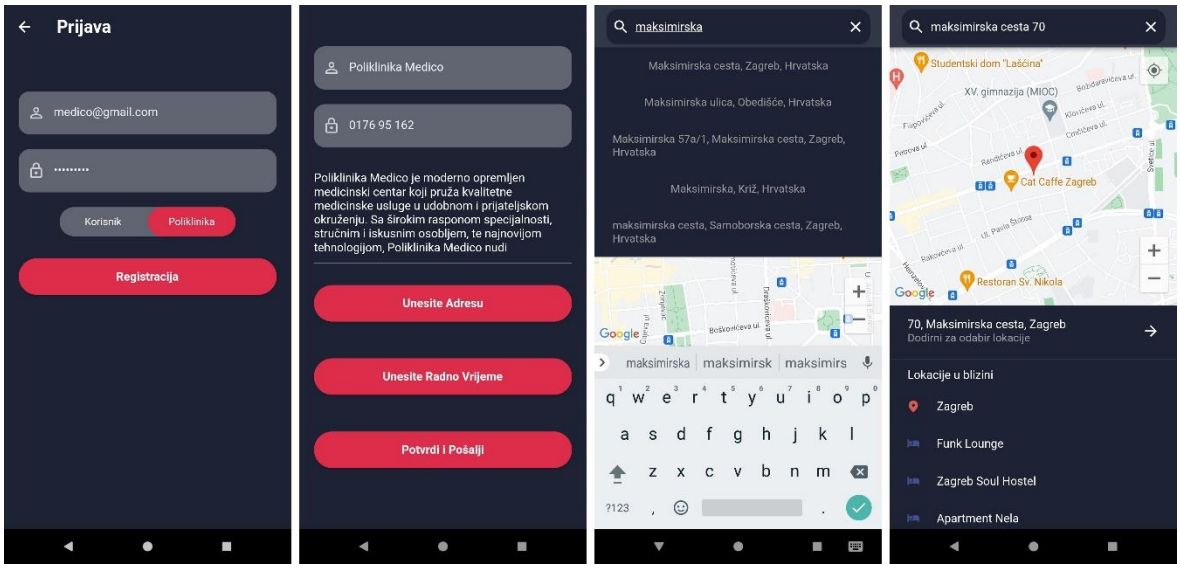
Kada korisnik odabere ekran za razgovore, prikazat će mu se svi razgovori u kojima je on prisutan, odnosno svi razgovori koji sadrže njegov ID. Klikom na pojedini razgovor otvara se ekran za prikaz razgovora u kojem su vidljive sve prijašnje poruke, ako ih ima. Funkcionalnost razgovora radi putem *Cloud Firestore* baze podataka koju omogućuje *Firebase servis*. *Cloud Firestore* baza podataka radi na tipu *observable* objekta koju su idealni za takve funkcionalnosti jer se ažuriraju sa svakim novim unosom podataka.



Slika 4.15 Mapa s poliklinikama

Na slici je vidljiv ekran `MapClinicsScreen`, koji je četvrti element donje navigacijske trake (engl. *bottom navigation bar*). Na ovom ekranu poziva se funkcija `_onMapCreated`, koja putem REST API-ja dohvaća sve dostupne klinike i pretvara ih u `marker`-e pomoću funkcije `_createMarkers`. Zatim se učitava Google Maps *widget* s trenutnom lokacijom i prikazanim `marker`-ima. Klikom na *marker* otvara se mali skočni prozor, a ako se na njega klikne, otvara se `ClinicInfoScreen` koji u *Contructoru* prima `Clinic` kao parametar na osnovu toga ispisuje podatke.

4.2.2. Mobilna aplikacija za poliklinike



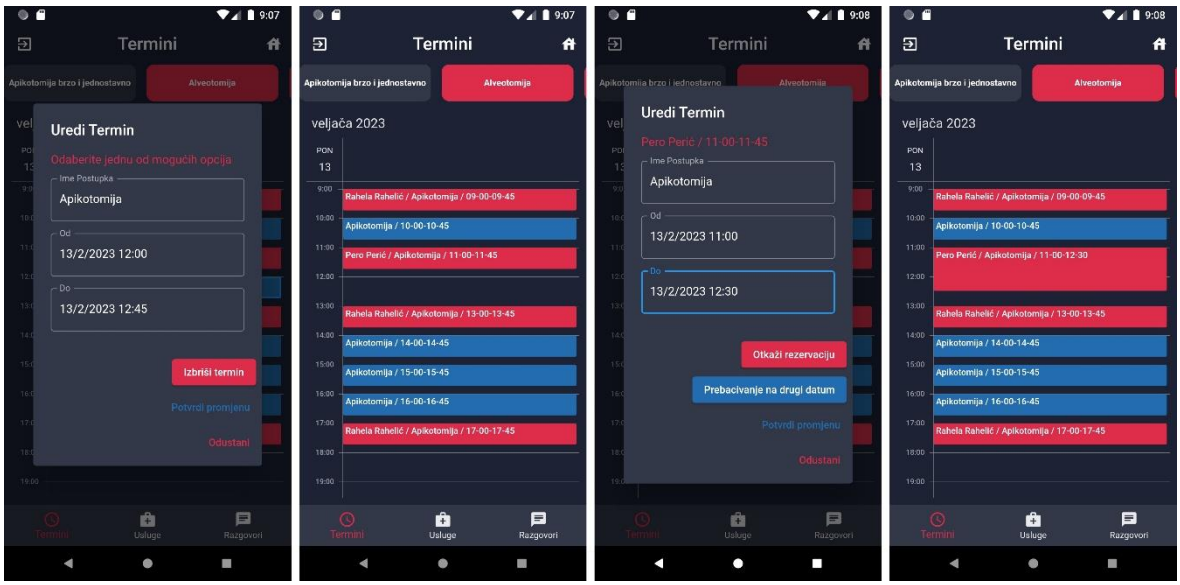
Slika 4.16 Prijava poliklinike

Početni dio otvaranja aplikacije i samog *login*-a i registracije je identičan kao i kod poglavlja 4.2.1. Do glavne razlike dolazi kada se krene ispunjavati ekran za dodatne informacije nakon provjere u kojoj postoji odabir unosa adrese. Klikom na dani gumb se otvara *PlacePicker widget* koji je dodan preko `place_picker` *dependency*-a iz `pubspec.yaml`. On kao parametre prima trenutnu lokaciju i Google API *key* te ujedno s njime se dobiva jednostavni ekran s tražilicom lokacija, samu interaktivnu Google mapu i funkcionalnost odabira na mapi koja vraća *latitude* i *longitude* ujedno i sami naziv mjesta/ulice.

Slika 4.17 Početni zaslon klinike

Ekran "Termini" je početni element donje navigacijske trake (engl. *bottom navigation bar*) nakon *logina* i dodavanja uloge poliklinike. Ekran se sastoji od `ListView.builder` *widgeta* koji ima orijentaciju horizontalnog listanja (engl. *scrolling*). Unutar sebe ima ispisane sve usluge koje trenutno ulogirana poliklinika nudi, a ta lista je dobivena uz `clinicId` i GET na REST API preko funkcije `getClinicalServiceList` unutar `ApiRepo` klase. Klikom na željenu uslugu se uz `onTap` otvara *Expanded widget* koji unutar sebe ima *SfCalendar widget*. *SfCalendar* na osnovu `medicalServiceId` iz

odabrane usluge dobiva podatke iz API-ja i na specifičan način za poliklinike ih ispisuje. Također se vidi ime i prezime klijenta koji je rezervirao termin.

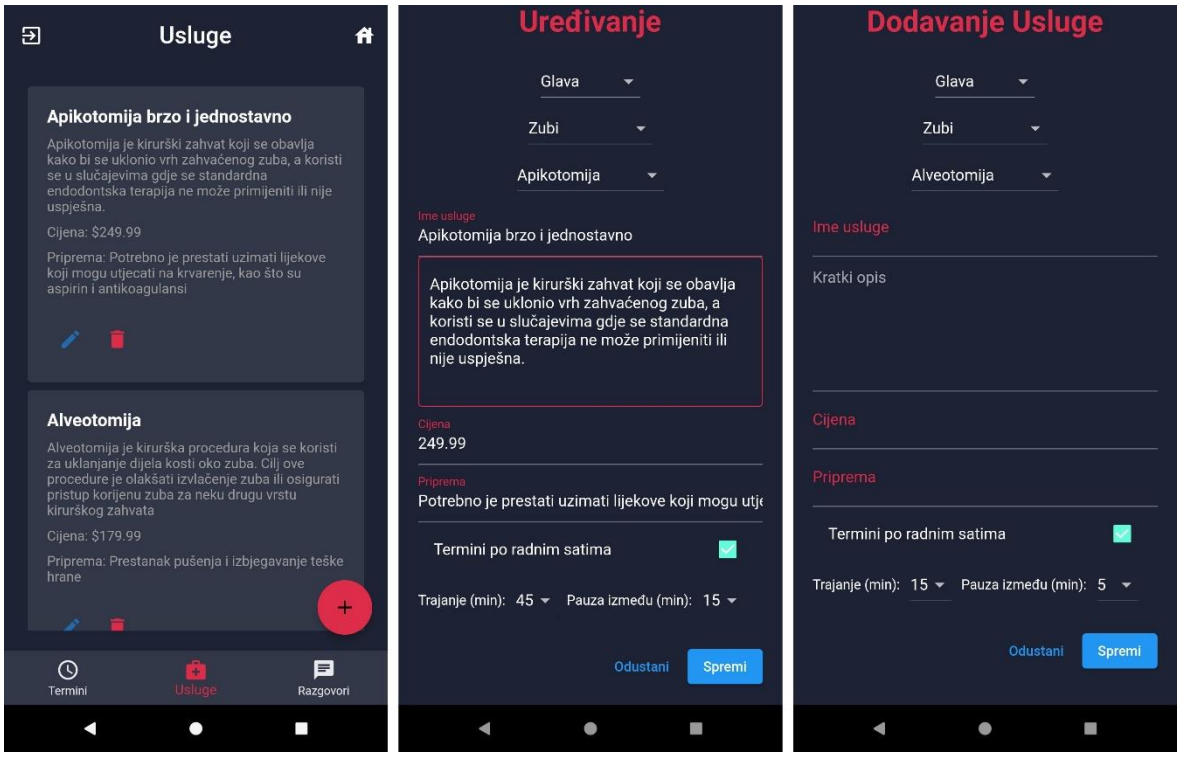


Slika 4.18 Funkcionalnost uređivanja termina

Klikom na prazni termin dolazi odskočni prozor s funkcionalnostima promjene imena postupka i vremenskim intervalom samog termina. Također moguće je i izbrisati odabrani termin ili odustati od bilo kakvog postupka.

U slučaju brisanja šalje se `meetingId` uz metodu *DELETE* na REST API i `nofitifyListeners` se poziva unutar `deleteAppointment` funkcije čime se pokreće ponovna izgradnja i ažuriranje podataka unutar *Sfcalendara*.

U slučaju da je stisnuti termin već zauzet pojavi se odskočni prozor drugačijeg dizajna koji se kreira na osnovu ako je `userId` unutar `Meeting` klase različit od `null`. On ima dodatne opcije otkazivanja trenutnog termina ili prebacivanja na drugi datum (slika 4.16).

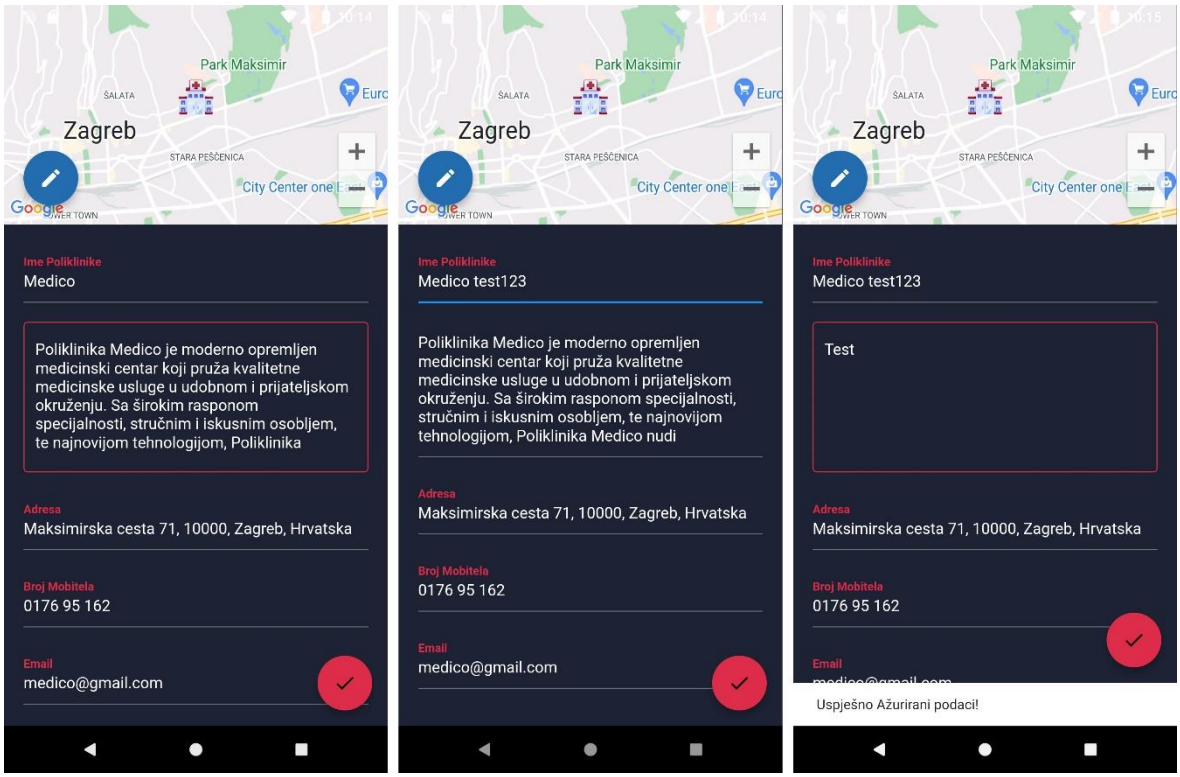


Slika 4.19 CRUD funkcionalnost usluga

U drugom elementu donje navigacijske trake (engl. *bottom navigation bar*) nalazi se mogućnost upravljanja uslugama koje poliklinika nudi. U `CrudServicesViewModel`-u se nalaze funkcije za izvršavanje CRUD operacija nad uslugama, kao što su brisanje, dodavanje i ažuriranje usluga, te funkcija za dohvaćanje svih usluga koje klinika nudi. Sve ove funkcije koriste ovisnosti koje se ubacuju preko `ApiRepo` klase, kroz sučelje `IApiRepo`.

DropDownButton widgeti se popunjavaju pozivom funkcija `getCategory`, `getSubCategory` i `getMedicalService`, gdje se odabirom jedne vrijednosti u prvom, drugom ili trećem izborniku, popunjava se vrijednost u drugom ili trećem izborniku na osnovu ID-a odabrane vrijednosti u prethodnom izborniku.

Zadnji element donje navigacijske trake (engl. *bottom navigation bar*) je razgovor (engl. *chat*) koji je bio u detalje objašnjen u prijašnjem podnaslovu 4.2.1 ispod slike 4.17.



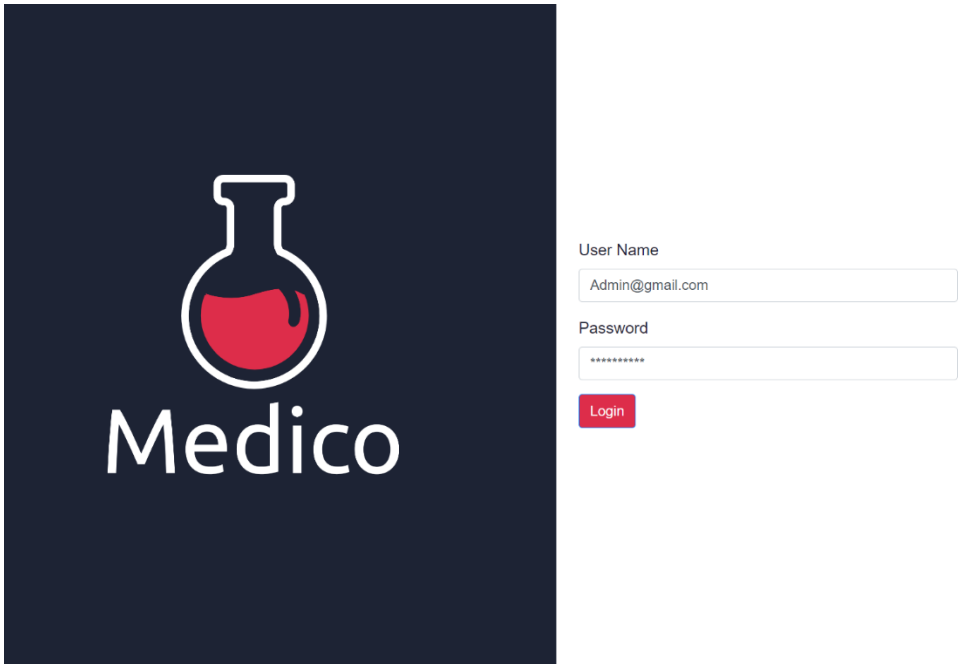
Slika 4.20 Podaci profila klinike

Finalna funkcionalnost koju ima poliklinika je ekran za promjenu podataka o samoj poliklinici on se sastoji od `SingleChildScrollView` koji unutar sebe ima GoogleMap *widget* s `FloatingActionButton` za ponovni odabir adrese. Postoje `TextFormField` za promjenu imena, opisa, adrese, broja telefona, email-a i gumb za promjenu radnog vremena koji otvara odskočni prozor s trenutno upisanim radnim vremenom. Također na dnu stranice je `FloatingActionButton` koji služi za slanje ažuriranih podataka s HTTP PUT na REST API.

U slučaju uspješnog ažuriranja podataka na dnu stranice nam iskače *snackBar widget*.

4.2.3. Web aplikacija za administratora

Isto kao i u mobilnoj aplikaciji, i na prvi pogled, sučelje za autentifikaciju je prvi ekran koji se prikazuje korisniku. Međutim, za razliku od mobilne aplikacije, ova je aplikacija primarno namijenjena adminu cijelog sustava,. Na ovom se sučelju nudi samo mogućnost prijave administratora u sustav.



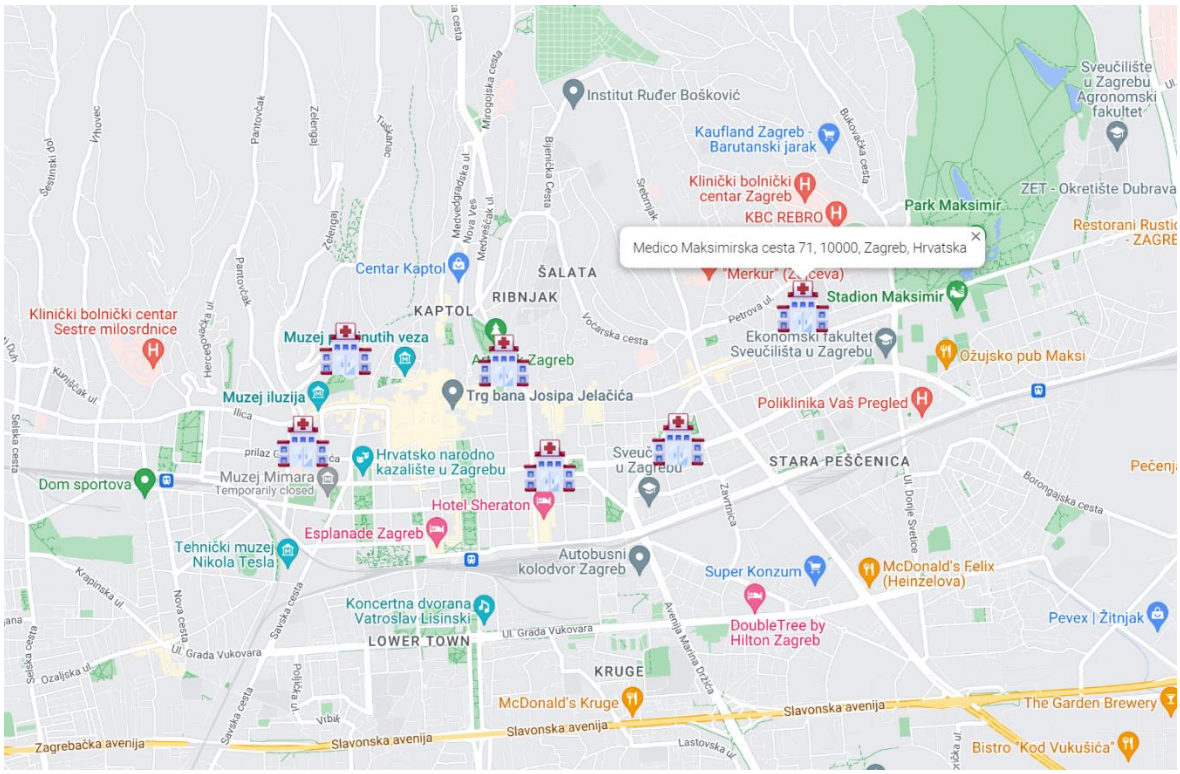
Slika 4.21 Admin - prijava

Klikom na gumb *login* aktivira se funkcija `login` (kod 4.4) u klasi `AuthService` koja služi za prijavu korisnika u aplikaciju putem Firebase autentifikacije. Funkcija prima dva argumenta: `email` i `password`, koji predstavljaju korisničke podatke potrebne za prijavu.

U funkciji se poziva metoda `signInWithEmailAndPassword` nad `afAuth` objektom, koji predstavlja instancu `AngularFireAuth` usluge, a koja omogućuje autentifikaciju korisnika preko Firebase autentifikacijskog sustava. Ta metoda prima `email` i `password` argumente te vraća objekt koji sadrži informacije o prijavljenom korisniku (engl. *user credentials*).

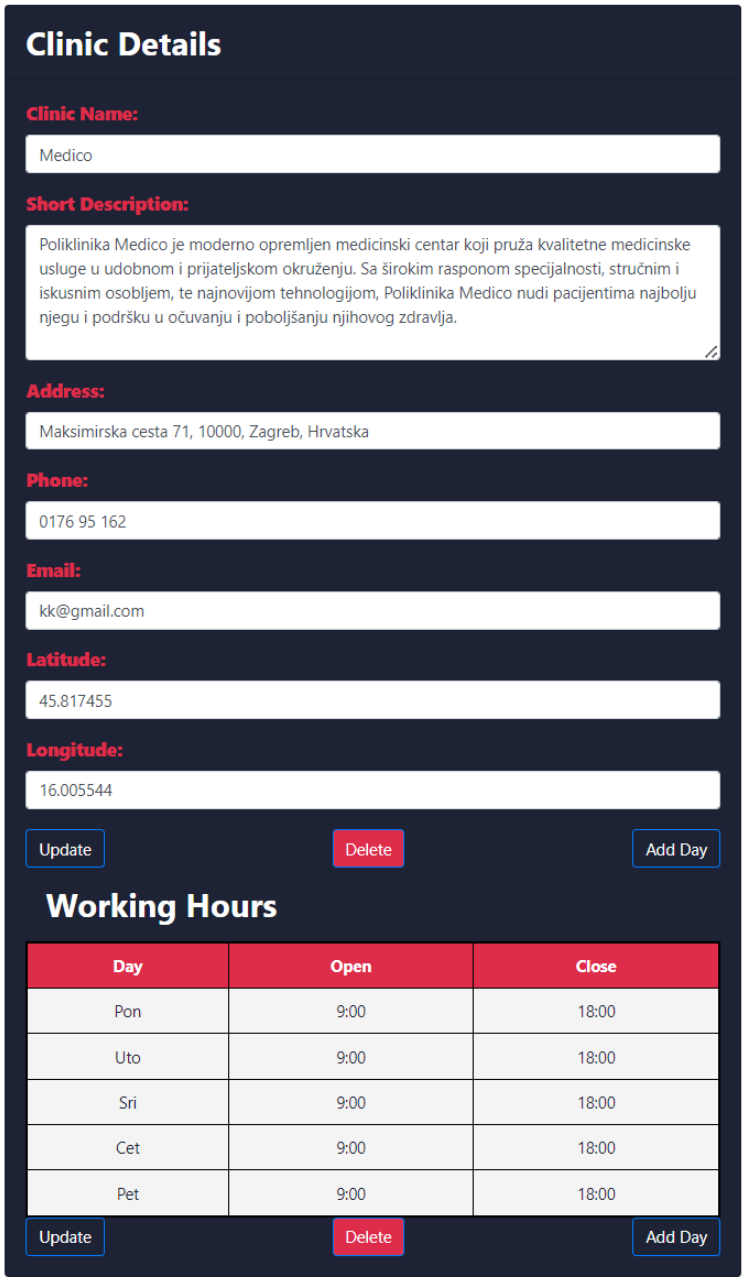
U slučaju da se dogodi greška prilikom prijave, bilo zbog pogrešnih korisničkih podataka ili nekog drugog razloga, funkcija hvata grešku putem *try-catch* bloka te ispisuje grešku u konzolu te je baca dalje. U suprotnom, funkcija vraća informacije o prijavljenom korisniku `userCredential.user` u obliku obećanja `Promise<any>`.

Nakon logina, postoji opcija pregleda svih komponenti koje su dostupne za crud: *Roles, Users, Client, Clinic, Category, Subcategory, MedicalService, ClinicalService, Meeting, WeekDays, WorkingHours*. U suštini sve crud operacije se odvijaju preko *ApiService* sloja koja unutar sebe sadrži HTTP POST, GET, PUT i DELETE metode za svaku od napomenutih klasa unutar REST API-a.



Slika 4.22 Prikaz mape poliklinika na Web aplikaciji

Poliklinike se mogu tražiti preko *map* komponente, prilikom `ngOnInit` `MapComponent` klase mapa se učitava i ujedno puni s `marker`-ima na jako sličan način kao i unutar mobilne aplikacije gdje se u ovom slučaju preko ApiService klase poziva GET metoda na REST API za sve poliklinike te se one pune u `marker` objekte i ispisuju na mapi kao ikone poliklinike unutar kojih su spremljeni ime i adresa. Zatim ako se klikne još jednom na odskočni prozor otvara se `EditClinicComponent` klasa.



Slika 4.23 Detalji klinike prikazani na admin aplikaciji

`EditClinicComponent` je komponenta koja služi za dodavanje nove ili uređivanje postojeće poliklinike ovisno o tome prosljeđuje li joj se `null` ili `Clinic` klasa. HTML komponenta se sastoji od sedam *input text* polja koji se po redu zovu; *clinic name, address, phone, e-mail, latitude, longitude*, i jednog *text area* polja *short description.*

Input text Address unutar sebe sadrži Google *autocomplete* za lokacije koji sprema ime adrese, *latitude* i *longitude* prilikom odabira. Ažuriranje (engl. *Update*) šalje HTTP PUT za

`WorkingHours` i za `Clinic` klase prema REST API-u dok brisanje (engl. *delete*) šalje HTTP DELETE i briše sve asocirano s primarnim ključem (engl. *primary key*) te ujedno i sam *Clinic* objekt. Donja tablica je uz ApiService popunjena sa HTTP GET metodom na osnovu `clinic.id` klikom na bilo koji red iz stupca radnih dana (pon-pet) dobiva se opcija uređivanja ili brisanja određenog radnog dana.

5. Testiranje i analiza gotovog programskog rješenja

Napravljeno je programsko rješenje koje bi povezivalo pacijente s poliklinikama u Republici Hrvatskoj. Ovime bi se pacijentima značajno olakšalo naručivanje na termine u poliklinikama, a poliklinikama bi se omogućila lakša organizacija radnih resursa poput radnog vremena, organizacije rada i kvalitete samog poslovanja.

Ovo programsko rješenje je predstavljeno i objašnjena je općenita ideja i zamisao samog programskog rješenja skupini liječnika dentalne medicine unutar jedne privatne ordinacije dentalne medicine u gradu Zagrebu.

Oni su bili poprilično zadovoljni s danim jednostavnim filterima, brzim unosom usluga i samom mapom za pronalazak dostupnih poliklinika. Od njih su iz prve ruke stečene povratne informacije koje su većinom bile pozitivne. Laka funkcionalnost same aplikacije je od velike važnosti za sve uključene strane u ovu aplikaciju jer omogućuje bolje korištenje vremena kao resursa.

Predložili su i određene ideje koje bi unaprijedile sam sustav. Jedna od predloženih ideja je dodavanje slika koje prikazuju željenu uslugu potencijalnom pacijentu i slike samih poliklinika. Vanjski izgled poliklinike, dizajn interijera, fotografije osoblja, fotografije zadovoljnih pacijenata i slično su kvalitetan pribor za utjecaj na odluku pacijenta o izboru poliklinike u kojoj će obavljati svoje zdravstvene preglede. Ovo bi povećalo razinu diferencijacije poliklinika i stvorilo mentalnu sliku kod pacijenata o mjestu u koje se potencijalno naručuju. Također, ovo bi moglo privući nove pacijente koji žele kvalitetniju i personaliziranu uslugu te će na temelju slika donijeti odluku o poliklinici u kojoj će rezervirati svoj termin za pregled unutar aplikacije.

Sljedeća ideja je dodavanje još jednog korisnika koji bi bio liječnik s mogućnošću pregleda isključivo vlastitog tjednog rasporeda, a uz vlastitu chat opciju također bise mogla poboljšati učinkovitost sustava. Liječnik bi mogao lakše organizirati svoje vrijeme i komunicirati s pacijentima putem aplikacije, što bi dodatno poboljšalo korisničko iskustvo.

Glavni cilj ovog programskog rješenja je bilo olakšati generalnoj populaciji brzo i efikasno, a opet lako pronalaženje slobodnih termina za medicinske preglede, bez nepotrebnog

čekanja, dodatnih razgovora i nejasnoća oko prebacivanja termina ili dodatnih poziva. Programsko rješenje je uspješno postiglo sve ciljeve opisane u početnom dokumentu prijave teme, a navedeno je zaključeno na temelju pozitivnih reakcija članova stomatološke ordinacije.

Ovaj cijeli programski sklop koji je razvijen funkcionira asinkrono, što je bilo planirano od početka razvoja projekta. Osim toga, u razvoju ovog programa korišteni su principi Inversion of Control (skraćeno IoC) i Dependency Injection (hrv. ubrizgavanje ovisnosti) koji pomažu u smanjenju ovisnosti između različitih dijelova aplikacije, povećavaju modularnost i olakšavaju testiranje.

Repository pattern koji je korišten također je važan princip razvoja aplikacija, jer se time osigurava pravilno upravljanje podacima i omogućava brže i učinkovitije pretraživanje podataka. Arhitektura MVVM (engl. *Model-View-ViewModel*) korištena u ovom rješenju omogućava jasnu razdiobu odgovornosti između logike aplikacije i korisničkog sučelja, što olakšava održavanje i proširivanje aplikacije.

Uzimajući u obzir sve navedeno, programsko rješenje nije samo zadovoljilo početni funkcionalni cilj, već je uspješno implementiralo brojne inženjerske principe i najbolje prakse koje su ključne za razvoj visokokvalitetnih softverskih rješenja. Ovaj pristup osigurava da je aplikacija fleksibilna i skalabilna.

Zaključak

U ovom završnom radu predstavljeno je inovativno programsko rješenje za učinkovitije upravljanje naručivanjem pacijenata u poliklinikama u Republici Hrvatskoj. Na temelju navedenih nedostataka dosadašnjeg načina naručivanja pacijenata u zdravstvenom sustavu Hrvatske te usporedbe sa sličnim ponuđenim rješenjima napravljeni su prijedlozi rješenja za efikasnije upravljanje naručivanjem pacijenata u poliklinike. Rješenje omogućuje lakše naručivanje pacijenata bez dugih čekanja i zbunjujućih procesa prije samih termina kod poliklinika te omogućuje poliklinikama bolju organizaciju i uvid u termine pregleda, vrijeme trajanja pregleda, organizaciju cjelokupnog poslovanja i lakšeg kontaktiranja s pacijentima.

Programsko rješenje je implementirano kroz mobilnu aplikaciju za pacijente i poliklinike, web aplikaciju za administratora i REST API koji spaja cjelokupno rješenje. Arhitektura sustava se sastoji od klijentskog, poslužiteljskog-podatkovnog sloja i programske tehnologije poput Flutter, Angular, Firebase i C# .Net Core. Mobilna aplikacija pruža programsko rješenje koje će zadovoljiti želje i pacijenata i klijenata, odnosno poliklinika u Hrvatskoj, za efikasnijim i bržim upravljanjem naručivanja za preglede i ostale potrebe pacijenata. Sve uključene strane će lakše i jednostavnije ulaziti u međusobne odnose, doći će do efikasnijeg i zadovoljnijeg načina spajanja između njih.

Prije pisanja danog rada autor nije imao iskustva u Flutter frameworku i Dart programskom jeziku, no unutar 2 mjeseca svakodnevnog rada odrađeno je cjelokupno programsko rješenje, a ono se primarno baziralo na tehnologijama koje nikada nisu obrađene unutar akademskog plana. Velikim dijelom zahvaljujući odličnim temeljima koje mogu biti stečene uz Visoko učilište Algebra. Ovdje studenti nisu samo učili radi učenja već radi razumijevanja i apliciranja danog znanja. Navedeno nije bilo usmjereno isključivo na jedan programski jezik nego na cjelokupnu logiku, arhitekturu i smisao programskih sustava.

Za kraj, unatoč tome što je aplikacija dovedena do željene razine smatram da uvijek ima mjesta za poboljšanja, unutar općenitih funkcionalnosti dodavanje slika u usluge i poliklinike, to bi promijenilo i izgled i korisničko iskustvo aplikacije na bolje.

Zbog svega već navedenog, smatra se kako je postignuto sve što je bilo planirano i ostavljena je mogućnost za buduća proširenja u slučaju da postoji interes i želja za njihovom implementacijom. Ovakvo programsko rješenje koje bi pacijentima i poliklinikama uvelike

ubrzalo proces kontaktiranja, naručivanja, sklapanja odnosa i dogovora, do sada nije postojalo u ovom obliku u Republici Hrvatskoj stoga ovakva inovacija programskog rješenja vodi do visoke učinkovitosti na strani svih uključenih. Brži i lakši način istraživanja poliklinika, lakše kontaktiranje s navedenim poliklinikama, bolja predodžba o lokacijama i vrstama pregleda koje nude poliklinike i kompletan sadržaj termina na dlanu omogućuje pacijentima kvalitetnije i jednostavnije vlastito vođenje zdravstvenih potreba i problematika. Uporabom ovog programskog rješenja došlo bi do veće efikasnosti poliklinika u njihovom poslovanju. Veća produktivnost rada, bolja organizacija radnog vremena, popis svih termina na jednom mjestu i lakše kontaktiranje sa pacijentima samo su neki od benefita koje ova aplikacija pruža.

Popis kratica

API	*Application Programming Interface*	Aplikacijsko programsko sučelje
CRUD	*Create Read Update Delete*	Kreiraj Čitaj Ažuriraj Izbriši
DI	*Dependency injection*	Ubacivanje ovisnosti
EF	*Entity Framework*	*Entity Framework*
HTML	*Hypertext Markup Language*	*Hypertext Markup Language*
HTTP	*HyperText Transfer Protocol*	*HyperText Transfer Protocol*
ID	*Identifier*	Identifikator
IoC	*Inversion of Control*	Inverzija Kontrole
JSON	*JavaScript Object Notation*	*JavaScript Object Notation*
MVC	*Model-View-Controller*	*Model-View-Controller*
ORM	*Object Relational Mapping*	*Object Relational Mapping*
UUID	*Universal Unique Identifyer*	Univerzalni Unikatni Identifikator
XML	*Extensible Markup Language*	*Extensible Markup Language*

Popis slika

Popis kôdova

Literatura

[1] HUP. “Lani su pacijenti u privatnim zdravstvenim ustanovama ostavili više od 3 milijarde kuna”, dostupno na: https://www.hup.hr/lani-su-pacijenti-u-privatnim-zdravstvenim-ustanovama-ostavili-vise-od-3-milijarde-kuna.aspx, datum pristupa: 20.1.2023.

[2] Flutter„What Is Flutter?“, dostupno na: https://docs.flutter.dev/ , datum pristupa: 20.1.2023.

[3] Angular, „What Is Angular?“, dostupno na: https://angular.io/guide/what-is-angular , datum pristupa: 1.2.2023.

[4] ROTH, D., ANDERSON, R., LUTTIN, S., Overview Of ASP.NET CORE“, dostupno na: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1. , datum pristupa: 1.2.2023

[5] „Get Started With Firebase Autentification On Flutter“: https://firebase.google.com/docs/auth/flutter/start , datum pristupa: 1.2.2023.

[6] Lapp, T., J.,“Understanding Provider in Diagrams — Part 1: Providing Values“, dostupno na: https://medium.com/flutter-community/understanding-provider-in-diagrams-part-1-providing-values-4379aa1e7fd5, datum pristupa 3.2.2023

[7] Hangfire, „Hangfire – Getting Started“, dostupno na: https://docs.hangfire.io/en/latest/getting-started/index.html ,datum pristupa: 5.2.2023.