Višeplatformi razvoj mobilne aplikacije za praćenje zdravstvenog stanja nefroloških bolesnika i stvaranje preporuka

Barbić, Roko

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:229990

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij

VIŠEPLATFORMSKI RAZVOJ MOBILNE APLIKACIJE ZA PRAĆENJE ZDRAVSTVENOG STANJA NEFROLOŠKIH BOLESNIKA I STVARANJE PREPORUKA

Diplomski rad

Roko Barbić

Osijek, 2024

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Roko Barbić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1266R, 07.10.2022.
JMBAG:	0165081282
Mentor:	prof. dr. sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Josip Balen
Član Povjerenstva 1:	prof. dr. sc. Goran Martinović
Član Povjerenstva 2:	prof. dr. sc. Krešimir Nenadić
Naslov diplomskog rada:	Višeplatformi razvoj mobilne aplikacije za praćenje zdravstvenog stanja nefroloških bolesnika i stvaranje preporuka
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada treba opisati izazove i probleme pri liječenju i praćenju nefroloških bolesnika na bolničkom ili kućnom liječenju. Uzimajući u obzir stanje u području i postojeća slična rješenja, treba definirati funkcionalne i nefunkcionalne zahtjeve višeplatformske aplikacije, predložiti model, arhitekturu i dizajn aplikacije, te postupak praćenja pacijenata, postupak obavješćivanja pacijenta i liječnika o odstupanjima u stanju, liječenju i prehrani, te stvaranje preporuka ovisno o utvrđenom stanju. Mobilna aplikacija treba omogućiti

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	19.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	26.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	03.10.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 03.10.2024.

Ime i prezime Pristupnika:	Roko Barbić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1266R, 07.10.2022.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Višeplatformi razvoj mobilne aplikacije za praćenje zdravstvenog stanja nefroloških bolesnika i stvaranje preporuka**

izrađen pod vodstvom mentora prof. dr. sc. Goran Martinović

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

Sadržaj

1. UVOD 1

2. PREGLED TRENUTNOG STANJA I POSTOJEĆA RJEŠENJA 2

2.1. Nefrologija i njezini bolesnici 2

2.2. Izazovi kod liječenja i terapija 2

2.3. Pregled stanja područja 4

2.3.1. Mobilna aplikacija Suyi 4

2.3.2. Fresenius myCompanion 4

2.3.3. CKD Helper 5

2.3.4. Mi Kidney 5

2.3.5. eGrađani, Portal zdravlja 5

2.4. Idejno rješenje aplikacije 6

3. MODEL I GRAĐA MOBILNE I WEB APLIKACIJE 7

3.1. Funkcionalni zahtjevi 7

3.2. Nefunkcionalni zahtjevi 7

3.3. Dijagram rada i komponente aplikacije 8

3.4. Arhitektura i dizajn aplikacije 11

4. PROGRAMSKO RJEŠENJE APLIKACIJE 13

4.1. Korištene programske tehnologije 13

4.1.1. Razvojno okruženje 13

4.1.2. Tehnologije za razvoj web aplikacije 13

4.1.3. Tehnologije za razvoj mobilne aplikacije 14

4.1.4. Baza podataka 15

4.1.5. Posluživanje i postavljanje aplikacije 15

4.2. Prikaz programskog rješenja 16

4.2.1. Upravljanje podatcima 16

4.2.2. Prijava i registracija korisnika 19

4.2.3. Programsko rješenje na strani poslužitelja 23

4.2.4. Programsko rješenje na strani klijenta - mobilna aplikacija 26

4.2.5. Programsko rješenje na strani klijenta, web aplikacija 30

5. PRIKAZ I ISPITIVANJE RADA APLIKACIJE 34

5.1. Mobilna aplikacija 34

5.2. Web aplikacija 39

5.3. Ispitivanje i analiza 44

6. ZAKLJUČAK 46

LITERATURA 47

ŽIVOTOPIS 49

SAŽETAK 50

ABSTRACT 51

PRILOZI 52

1. UVOD

Glavna funkcija bubrega u ljudskom tijelu je filtriranje otpadnih produkata i suviška tekućine iz krvi. Kada bubrezi počnu gubiti svoju sposobnost filtracije, opasne količine tekućine, elektrolita i otpadnog materijala počinju se nakupljati u tijelu, što može dovesti do smrti ukoliko te vrijednosti dosegnu kritične razine. Smatra se da je kronična bubrežna bolest (KBB) prisutna kod 10% odrasle populacije, a riječ je o bolesti kod koje bubrezi postupno gube svoju funkciju. KBB se može podijeliti u pet stadija, ovisno o razini funkcije bubrega. Peti stadij je najteži i uključuje bolesnike čija je bubrežna funkcija ispod 15% i kod takvih bolesnika potrebno je liječenje dijalizom ili u konačnici transplantacija bubrega. Rano otkrivanje je ključno u sprječavanju napredovanja bolesti, no to je često izazovno jer KBB u ranim fazama gotovo nema simptoma. Zbog toga je važno u općoj populaciji razviti svijesnost o postojanju KBB-a posebice zato što su najčešći uzroci povišeni krvni tlak i šećerna bolest. Kada pacijentima bude dijagnosticirana KBB, moraju biti kontinuirano praćeni kako bi se usporila progresija bolesti. Progresija bolesti se može usporiti redovitim uzimanjem lijekova te zdravim životnim navikama. Zbog toga je vrlo važno unaprijediti zdravstvenu pismenost opće populacije što se može postići i putem mobilnih aplikacija.

Ovaj diplomski rad ima za cilj razviti sustav koji bi pomogao u cjelokupnom procesu liječenja pacijenata, a sastoji se od dva dijela - web i mobilne aplikacije. Web aplikacija medicinskom osoblju omogućuje unos i spremanje informacija o pacijentu, poput laboratorijskih nalaza, dnevnika dijaliza i terapija, te vizualizira podatake kako bi liječnici imali bolji uvid u situaciju. Također, prilikom pregleda laboratorijskih nalaza, sustav liječnicima pruža vizualno upozorenje o parametrima koji odstupaju od željenih vrijednosti i potporu u stvaranju savjeta za pacijente. Za pacijente je namjenjena mobilna aplikacija preko koje mogu unositi izmjerene vrijednosti krvnog tlaka i tjelesne težine te imaju pristup informacijama o obavljenim dijalizama i savjetima liječnika. Mobilna aplikacija je više platformska kako bi bila dostupna što većem broju pacijenata.

U drugom poglavlju detaljno je opisana kronična bubrežna bolest i izazovi u njenom liječenju. Treće poglavlje obrađuje arhitekturu, dizajn i zahtjeve sustava. Četvrto poglavlje prikazuje korištene tehnologije i opisuje programsko rješenje, dok peto poglavlje predstavlja izgled korisničkih sučelja, objašnjava upotrebu aplikacija i analizira njihov rad.

2. PREGLED TRENUTNOG STANJA I POSTOJEĆA RJEŠENJA

U ovom poglavlju razmotrit će se trenutno stanje i postojeća rješenja vezana uz temu rada, uz analizu tehnologija i pristupa koji se trenutno koriste.

2.1. Nefrologija i njezini bolesnici

Nefrologija je grana interne medicine koja se bavi proučavanjem rada, funkcijom i bolestima bubrega i organa mokraćnog sustava. Stoga se u odjelu za nefrologiju i dijalizu liječe bolesnici s različitim bubrežnim bolestima, arterijskom hipertenzijom, te oni koji imaju kronično i akutno zatajenje bubrega.

Kronična bubrežna bolest (kronično zatajenje bubrega, KBB) je stanje u kojem bubrezi gube sposobnost uklanjanja otpadnih produkata metabolizma i suviška tekućine iz organizma. Nagomilavanje ovih tvari može dovesti do oštećenja drugih organa i organskih sustava te narušiti funkcioniranje cijelog organizma. Kod kronične bubrežne bolesti uglavnom se radi o stadijima 1- 3 (od mogućih 5), no ponekad ona napreduje do završnog stadija (stadij 5) kada dolazi do potpunog propadanje bubrežne funkcije tada više ne pomaže niti jedna terapijska opcija izuzev nadomještanja bubrežne funkcije. Bolesnici koji imaju KBB, odnosno kod kojih je izgubljeno više od 50% normalne funkcije bubrega imaju povećani rizik za nastajanje bolesti srca i krvnih žila, samim time i povećani rizik prijevremene smrti. Zbog toga je važno ove bolesnike pratiti i preporučiti im razne dijetetske mjere [1].

HRNBF (Hrvatski registar nadomještanja bubrežne funkcije) obuhvaća podatke samo za bolesnike koji se liječe nadomještanjem bubrežne funkcije (NBF), a ne obuhvaća bolesnike u ranijim stadijima KBB. Prema njegovim podatcima, izvještaja za 2014. godišnji prirast broja bolesnika u završnom stupnju bubrežnog zatajenja (ZSBZ) u Hrvatskoj iznosio je 4,3 % [2].

2.2. Izazovi kod liječenja i terapija

Kako je ova bolest u svojim ranim stadijima praktički bez simptoma, osobe se ne javljaju liječniku te tijekom vremena bolest postepeno napreduje te u konačnici može dovesti do potpunog i trajnog zatajenja bubrežne funkcije (završnog stadija KBB). To znači da je u ovom stadiju funkcija bubrega toliko narušena (bubrezi imaju ≤10% normalne funkcije) da je potrebno razmotriti početak nadomještanja bubrežne funkcije dijalizom ili transplantacijom.

Glavni cilj liječenja jest usporavanje napredovanja KBB do njenog završnog stadija. Da bi se to postiglo, potrebno je što ranije postaviti dijagnozu, te što bolje kontrolirati glavni uzrok zatajenja i sve rizične čimbenike koji mogu dovesti do daljnjeg napredovanja bolesti.

Prvi korak u uspješnom liječenju KBB je otkrivanje poremećaja koji je do nje doveo. Neki od tih poremećaja su reverzibilni, primjerice pretjerano korištenje lijekova koji mogu narušiti bubrežnu funkciju ili oslabljena opskrba bubrega krvlju. Rana dijagnostika i liječenje navedenih uzroka može dovesti do oporavka bubrežne funkcije ili usporiti njeno napredovanje.

Arterijska hipertenzija ili visok krvni tlak je prisutna u 80-85% osoba s KBB. Glavni cilj pri kontroli i usporavanju napredovanja KBB je kontrola arterijskog tlaka te se bolesnicima preporučuje svakodnevno mjerenje vrijednosti tlaka u kućnim uvjetima. Brojni su lijekovi na raspolaganju u kontroli arterijske hipertenzije [3].

Također kod liječenja KBB-a bolesnicima je vrlo često potrebna prilagodba prehrane kako bi se usporio razvoj bolesti. Potrebno je smanjiti udio soli u prehrani kako bi se spriječilo nakupljanje suvišne tekućine i bolje regulirao arterijski tlak. Napredovanje kronične bubrežne bolesti može se usporiti i posebnim režimom prehrane sa smanjenim udjelom proteina. Neki bolesnici mogu profitirati i od dijete temeljene na hrani biljnog podrijetla. Jedna od najčešće savjetovanih modifikacija prehrane je i ona o smanjenom unosu kalijem bogate hrane. Povišene vrijednosti kalija u krvi mogu remetiti rad stanica te dovesti do životno ugrožavajućih poremećaja. Razina kalija u krvi može se regulirati uvođenjem diuretika, ali i izbjegavanjem namirnica za koje se zna da su bogate kalijem.

Usprkos liječenju i redovitim kontrolama u nekih bolesnika dolazi do postepenog trajnog pogoršanja KBB i nepopravljivog narušavanja bubrežne funkcije do stadija u kojem je potrebno razmotriti početak njenog nadomještanja. Najbolji način nadomještanja bubrežne funkcije jest transplantacija bubrega, no ona nije moguća kod svih bolesnika zbog njihovih ostalih bolesti i stanja.

U bolesnika koji nisu pogodni za transplantaciju ili do same transplantacije potrebno je početi nadomještanje bubrežne funkcije dijalizom. Dvije su vrste dijalize, hemodijaliza (pročišćavanje krvi pomoću aparata, provodi se u zdravstvenoj ustanovi više puta tjedno) i peritonejska dijaliza (pročišćavanje krvi putem potrbušnice trbušne maramice, provodi se u kućnim uvjetima od strane samog bolesnika ili njegovih najbližih).

Vodeći se procjenama da 10% odrasle populacije u svijetu ima određeni stupanj kronične bubrežne bolesti, prema [4] procjenjuje se da je u Hrvatskoj kroničnih bubrežnih bolesnika do 300000.

2.3. Pregled stanja područja

“Mobile health” ili mHealth, kako ga Svjetska zdravstvena organizacija (WHO) naziva, znači korištenje mobilnih i bežičnih tehnologija za poboljšanje zdravlja. U 2022. godini, na Apple App Store-u je bilo dostupno 41.517 aplikacija za zdravlje [5]. Prednosti ovih aplikacija su to što su lako dostupne, pomažu u dosezanju ljudi koji inače nemaju lagani pristup medicinskoj skrbi, prevladavaju geografske barijere i mogu precizno prikupljati podatke zahvaljujući tehnologijama kao što su GPS i prepoznavanje slika [6].

2.3.1. Mobilna aplikacija Suyi

U svrhu istraživanja koje je provedeno u Kini [7], razvijena je aplikacija Suyi. U istraživanju je sudjelovalo 5.329 bolesnika sa KBB, te je mobilnu aplikaciju koristilo njih 2.492. Aplikacija Suyi obuhvaća funkcionalnosti poput upravljanja osobnim informacijama, podsjetnike za naknadne preglede, fiziološke i psihološke procjene, zdravstveno obrazovanje, upravljanje ambulantnim lijekovima, kućno upravljanje prehranom, sesije obrazovanja uživo za pacijente, povratne informacije o procjeni i sustave za upozoravanje na rizik. Postupak korištenja aplikacije je takav da se pacijenti prvo upisuju u Centar za upravljanje kroničnim bubrežnim bolestima (KBB), gdje se uspostavlja medicinski karton. Aplikacija Suyi instalira se na mobilni uređaj pacijenta, sinkronizirajući dijagnostičke i pregledne podatke iz bolnice sa sučeljem u aplikaciji. Medicinske sestre zatim šalju planove zadataka za kućno upravljanje, materijale za obrazovanje pacijenata, ljestvice procjene i zakazuju sesije obrazovanja uživo za pacijente putem aplikacije. Pacijenti primaju te zadatke upravljanja i učitavaju svoje podatke u aplikaciju, kao što su tjelesna temperatura, težina, krvni tlak i razina šećera u krvi. Također pristupaju obrazovnim materijalima i ispunjavaju odgovarajuće procjene znanja. Medicinske sestre s punim radnim vremenom na računalu primaju učitane informacije iz pacijentove aplikacije, provode sveobuhvatnu procjenu i pružaju povratne informacije o procjeni.

Grupa koja je koristila aplikaciju i ambulantnu skrb pokazala je superiorne rezultate u usporedbi s grupom koja je imala tradicionalan pristup bez uporabe aplikacije. Do kraja istraživanja, uočeno je da su laboratorijski podaci za fosfor, natrij, trigliceride i hemoglobin pokazali značajna poboljšanja.

2.3.2. Fresenius myCompanion

Aplikacije od strane Fresenius Medical Care, namjenjena je pacijentima koji nadomještaju bubrežnu funkciju dijalizom u nekim od NephroCare klinika. Mogućnosti aplikacije su praćenje dnevnika svih terapija, propisanih lijekova te laboratorijskih nalaza, također korisniku

nudi sadržaj edukacijskog tipa sa korisnim savjetima vezanim uz dijalizu i sa zdravim receptima za kuhanje [8]. Ovakav pristup liječenju pomaže korisnicima da dobiju bolji uvid u trenutno zdravstveno stanje.

2.3.3. CKD Helper

U [9] je opisan je razvoj mobilne aplikacije CKD Helper (engl. Chronic Kidney Disease) namjenjene zdravstvenim djelatnicima. Aplikacija koristi algoritame koji pomažu u otkrivanju kronične bubrežne bolesti (KBB) te prepoznavanju drugih povezanih bolesti, poput Anemije i Mineralne bolesti kostiju (MBD). Aplikacija također preporučuje planove liječenja koji su preuzeti iz nacionalnih smjernica. Naglasak aplikacije je na ranom otkrivanju KBB-a, budući da se bolest u većini slučajeva dijagnosticira u kasnijim fazama, što znatno otežava i poskupljuje proces liječenja.

2.3.4. Mi Kidney

Odjel za bubrežne bolesti u bolnici St. James u Irskoj dizajnirao je i razvio mobilnu aplikaciju pod nazivom Mi Kidney. Aplikacija ima za cilj pružanje informacija i edukativnog sadržaja kako bi pacijenti prepoznali rizične faktore koji mogu dovesti do kronične bubrežne bolesti (KBB). Također, aplikacija poboljšava komunikaciju između liječnika i pacijenta te obavještava pacijenta o njegovom stanju. Ovakav pristup doprinosi liječenju jer pacijent postaje aktivan i informiran o svom zdravstvenom stanju te ima lakši pristup informacijama koje mogu pomoći u vođenju zdravijeg načina života. Provedeno je pilot-istraživanje u kojem je većina sudionika bila zadovoljna aplikacijom, a pozitivnu promjenu u vidu smanjenja BMI (engl. *Body Mass Index*) postiglo je 45% sudionika u razdoblju od 12 tjedana [10].

2.3.5. eGrađani, Portal zdravlja

Portal zdravlja nije u potpunosti sličan aplikaciji koju ovaj projekt nastoji razviti, ali je važno spomenuti jer omogućuje građanima RH uvid u vlastite medicinske podatke iz Centralnog zdravstvenog informacijskog sustava Republike Hrvatske. Prijava se vrši putem e-građani portala koristeći token banke, mToken ili elektroničku osobnu iskaznicu. Mogućnosti koje Portal zdravlja nudi uključuju pregled podataka o cijepljenju, provjeru propisanih uputnica, laboratorijskih nalaza te nalaza iz bolnica i specijalističkih ustanova. Također, korisnici mogu vidjeti propisane lijekove i preuzete lijekove te slati zahtjeve za recepte. Aplikacija je dostupna i na mobilnim uređajima [11].

2.4. Idejno rješenje aplikacije

Potrebno je razviti sustav koji bi služio nefrološkim bolesnicima u njihovom liječenju. Sustav bi se sastojao od web aplikacije namijenjene doktorima i medicinskim sestrama, koja bi im omogućila lakšu kontrolu nad podacima kao što su laboratorijski nalazi, dnevnici dijalize, terapije lijekova i opći podaci o pacijentima. Također, omogućilo bi im komunikaciju i lakše davanje preporuka pacijentima uz pomoć sustava koji bi predlagao određene preporuke s obzirom na vrijednosti parametara u nalazu. S druge strane mobilna aplikacija bila bi namijenjena pacijentima, omogućujući im praćenje dnevne terapije lijekovima, bilježenje tjelesne mase, te zapisivanje izmjerenih vrijednosti krvnog tlaka. Pacijenti bi također primali obavijesti o dijalizi i poruke od doktora s korisnim savjetima. Mobilna aplikacija bi bila više platformska s ciljem dosezanja što većeg broja ljudi.

3. MODEL I GRAĐA MOBILNE I WEB APLIKACIJE

U ovom poglavlju bit će navedeni funkcionalni i nefunkcionalni zahtjevi te prikazani model i struktura web i mobilne aplikacije

3.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi opisuju koja ponašanja bi sustav trebao imati, odnosno mogućnosti koje bi sustav trebao moći izvršiti. Pošto se sustav u ovom projektu sastoji od dva dijela, postoji podjela s obzirom na korisnike i platformu koja se koristi.

Funkcionalni zahtjevi koje web aplikacija ima su:

- Doktori i medicinsko osoblje mogu se registrirati i prijaviti
- Mogućnost dodavanja novih pacijenta unoseći njihove osobne podatke, dijagnozu i rezultate testova na hepatitis B i C
- Pregled svih pacijenata te mogućnost pretraživanja i filtriranja pacijenata prema specifičnim kriterijima
- Unos novih laboratorijskih nalaza i zabilježbi o dijalizi
- Propisivanje lijekova pacijentima
- Mogćnost pisanja preporuka i savjeta pacijentima
- Mogućnost vizualizacije medicinskih podataka o pacijentu

Mobilna aplikacija ipak se razlikuje po funkcionalnostima od web aplikacije, jer su njezini korisnici pacijenti. Njeni funkcionalni zahtjevi su:

- Pacijenti se mogu prijaviti u mobilnu aplikaciju
- Mogu unositi podatke o dnevnim aktivnostima: lijekove koje su popili, izmjerenu težinu i krvni tlak
- Pregled prethodnih dnevnika
- Pregled doktorovih preporuka i upozorenja
- Dobivanje informacija o izvršenoj dijalizi
- Generiranje zdravih recepta koristeći AI sustav

3.2. Nefunkcionalni zahtjevi

Za razliku od funkcionalnih zahtjeva koji definiraju što sustav treba raditi, nefunkcionalni zahtjevi definiraju kako sustav treba raditi. Najčešće su to određene kvalitete poput brzine, sigurnosti i održivosti. Nema potrebe za podijelom na web i mobilnu aplikaciju što se tiče

nefunkcionalnih zahtjeva pošto oba sustva teže postići isti cilj. Neki od nefunkcionalnih zahtjeva su:

- Sigurnost podataka u bazi, enkripcija lozinke
- Aplikacija mora biti responzivna i prilagođena različitim uređajima i rezolucijama
- Mobilna aplikacija mora biti jednostavna za korištenje i optimizirana za Android i iOS uređaje
- Vizualizacija podataka mora biti brza i responzivna čak i kada prikazuje podatke iz dužeg vremenskog perioda
- Aplikacija mora biti dostupna bez prekida

3.3. Dijagram rada i komponente aplikacije

Dijagram rada aplikacije na slikama 3.1 i 3.2 prikazuje tijek korištenja aplikacije te sve procese kroz koje korisnik prolazi ili može proći, ovisno o njegovim ovlastima i potrebama.

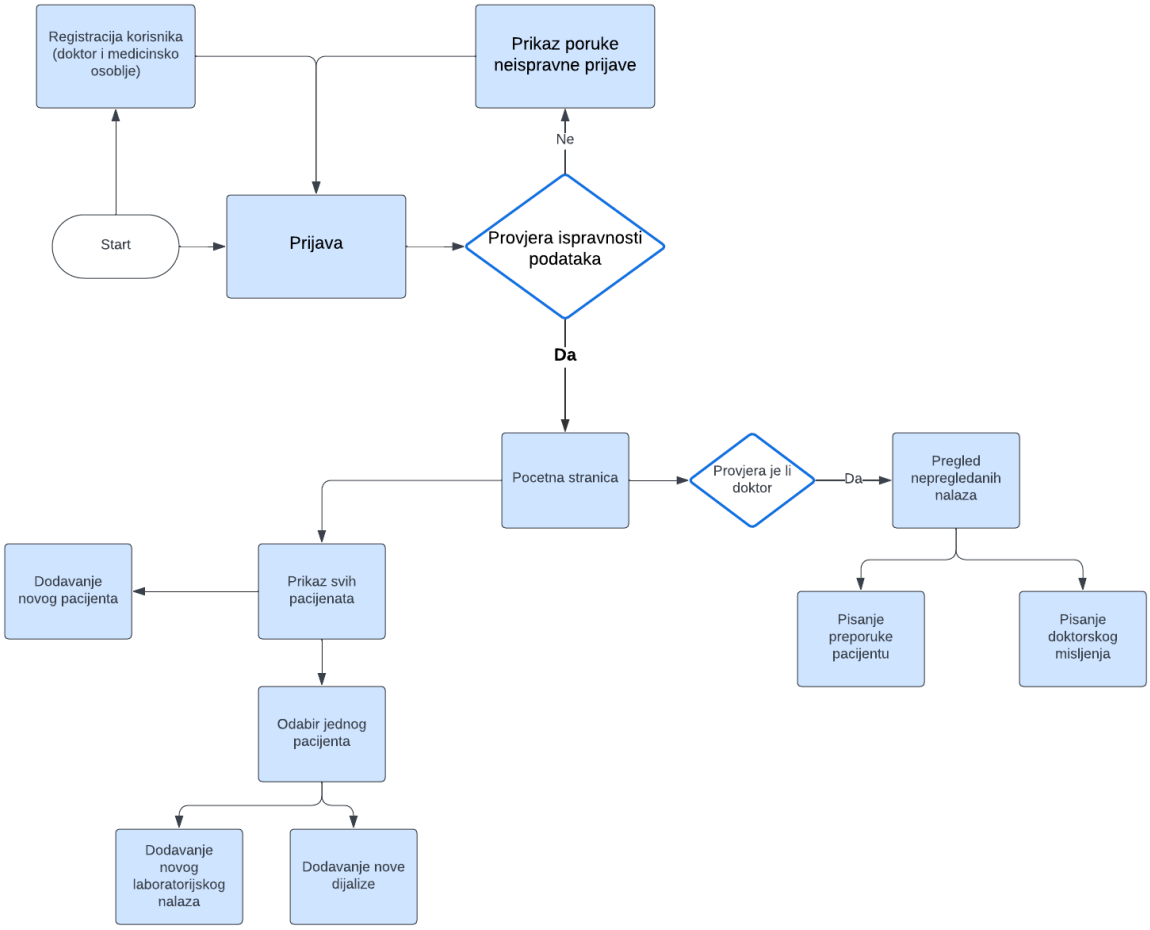
Prilikom pokretanja web aplikacije, korisnik (doktor ili medicinsko osoblje) ima opciju prijave ili registracije. Nakon uspješne prijave, korisnik se nalazi na početnom zaslonu koji prikazuje osnovne informacije o aplikaciji, dok mu se na navigacijskoj traci nude dodatne opcije. Te opcije uključuju pregled pacijenata, a ako je korisnik doktor, i pregled laboratorijskih nalaza koji još nisu pregledani.

Kada korisnik odabere opciju pregleda pacijenata, web aplikacija ga preusmjerava na stranicu s tablicom koja sadrži sve pacijente. Na toj stranici korisnik može pretraživati pacijente prema kriterijima kao što su ime i prezime. Također, aplikacija nudi mogućnost dodavanja novih pacijenata, a odabirom te opcije otvara se nova stranica na kojoj se unose informacije o pacijentu. Nakon unosa novog pacijenta, korisnik se vraća na stranicu sa svim pacijentima. Na toj stranici nalaze se i gumbi s akcijama za upravljanje pacijentovim podacima, poput dodavanja laboratorijskih nalaza, izvješća o dijalizi i opcija za detaljan pregled pacijenta.

Svaka od tih akcija vodi korisnika na odgovarajuću stranicu za unos ili pregled, ovisno o odabranom zadatku. Stranica za pregled detalja o pacijentu prikazuje sve osnovne informacije o pacijentu, rezultate testova na hepatitis B i C, sve laboratorijske nalaze i izvješća o dijalizi, koje je moguće pretraživati prema datumu. Također, prikazuju se podaci o terapiji koju pacijent prima, a doktoru se nudi mogućnost propisivanja novih lijekova. Na dnu stranice nalazi se

vizualizacija laboratorijskih nalaza kroz vrijeme, kao i mogućnost pregleda lijekova koje je pacijent uzimao u određenom razdoblju.

Ako korisnik na navigacijskoj traci odabere opciju pregleda nepregledanih nalaza, otvara se stranica sa svim nalazima koje je potrebno pregledati. Klikom na gumb za pregled, doktor je preusmjeren na stranicu s detaljima nalaza, gdje se od njega očekuje unos komentara i mišljenja o nalazu. Na toj stranici postoji i gumb za aktivaciju pomoći koja ističe parametre u nalazu koji izlaze izvan preporučenih granica. Doktor također ima opciju pisanja preporuka ili sugestija pacijentu, također sustav mu može i preporučiti koju preporuku da napiše s obzirom na vrijednosti parametara tog nalaza.

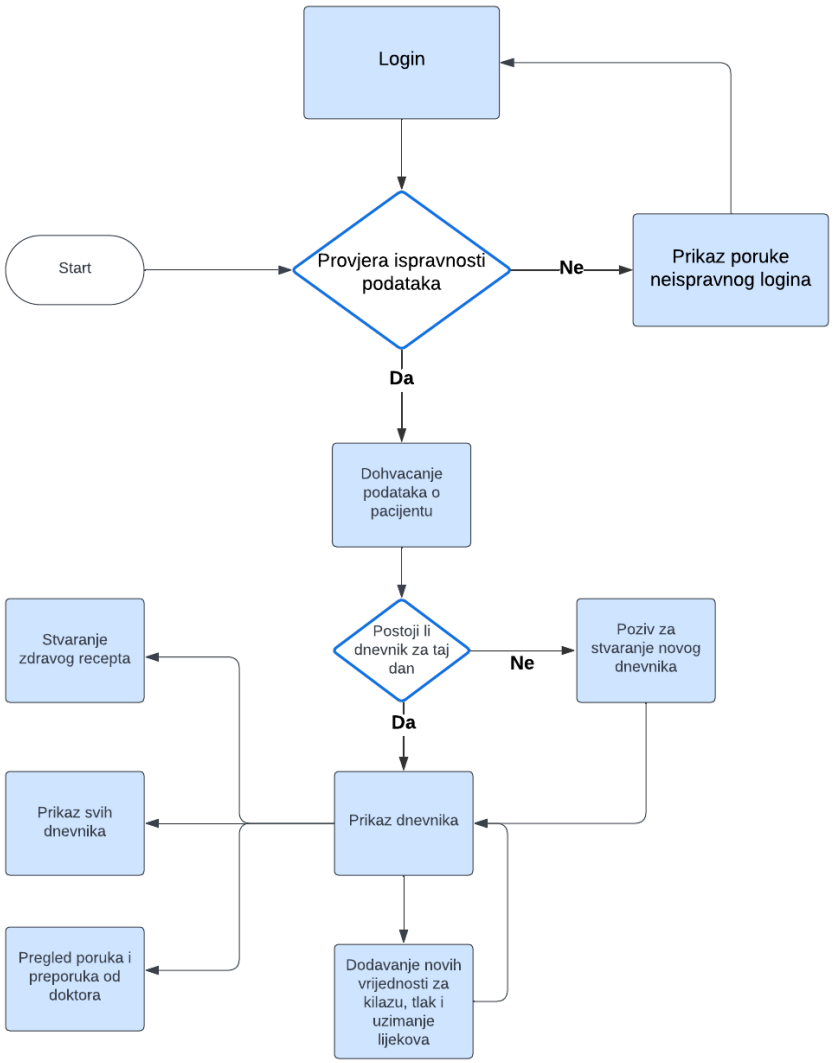


Slika 3.1. Prikaz tijeka korištenja web aplikacije

Kod pokretanja mobilne aplikacije, prikazuje se zaslon za prijavu ukoliko pacijent nije već prijavljen, prijavu izvršava pomoću svoje e-pošte adrese i šifre koju mu je dodijelio liječnik. Nakon prijave, aplikacija automatski stvara novi dnevnik za pacijenta, ako već nije stvoren za

taj dan. U dnevniku pacijent može unositi podatke o lijekovima koje je uzeo taj dan, izmjerenoj težini i vrijednostima krvnog tlaka tijekom dana. Navigacijska traka aplikacije nudi dodatne funkcionalnosti, poput pregleda svih prethodnih dnevnika, preporuka i upozorenja od doktora, te mogućnost generiranja zdravog recepta.

Generiranje zdravog recepta funkcionira na način da pacijent unese vrstu hrane koju želi jesti, a aplikacija mu prikazuje recept koji zadovoljava njegove prehrambene želje uzimajući u obzir da je recept namjenjen ljudima sa visokim krvnim tlakom. Na slici 3.2 može se vidjeti dijagram rada aplikacije.



Slika 3.2. Prikaz tijeka korištenja mobilne aplikacije

3.4. Arhitektura i dizajn aplikacije

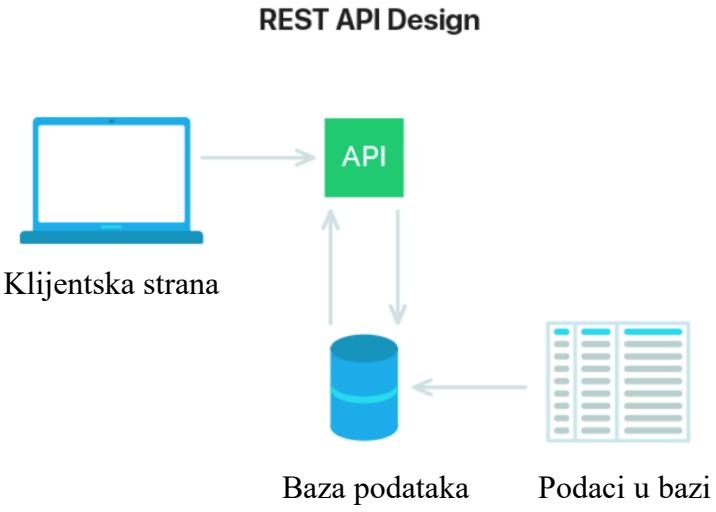
Web aplikacija je izgrađena po arhitekturnom obrascu MVC (engl. *Model-View-Controller*) koji dijeli aplikaciju na tri osnovna dijela: Model, View i Controller. Ovakav pristup omogućuje odvajanje odgovornosti unutar aplikacije, čime se olakšava održavanje i skaliranje sustava [12]. Model upravlja poslovnom logikom i strukturom podataka, View predstavlja korisničko sučelje i osigurava prikaz podataka, dok Controller služi kao posrednik između Modela i Viewa, povezujući korisničke zahtjeve s odgovarajućim podacima i operacijama. Ključna uloga Controllera je omogućavanje komunikacije između klijentske strane i baze podataka, što je ostvareno putem RESTful API-ja.

API (engl. *Application Programming Interface*) je skup definicija i protokola za izgradnju i integraciju softverskih aplikacija. API često djeluje kao "ugovor" između pružatelja informacija (engl. *server*) i korisnika informacija (klijenta), utvrđujući koje informacije klijent treba dostaviti u zahtjevu i koje informacije poslužitelj vraća kao odgovor.

REST (engl. *Representational State Transfer*), reprezentacija stanja resursa je odgovorno za definiranje ograničenja za razmjenu resursa između klijenata i poslužitelja [13].

RESTful API koristi HTTP metode kao što su GET, POST, PUT, i DELETE za komuniciranje s resursima. Kada klijent putem RESTful API-ja pošalje zahtjev, prenosi se reprezentacija stanja resursa prema klijentu ili endpointu. Ta informacija se obično dostavlja u formatu kao što je JSON (engl. *JavaScript Object Notation*), koji je najčešće korišten zbog toga što je čitljiv i ljudima i strojevima [14].

U ovom projektu, Controller iz web aplikacije služi kao poslužitelj, te izlaže RESTful API endpoint-e, koji omogućuju komunikaciju između klijentske strane (web i mobilna aplikacija) i baze podataka sustava. Ovi endpoint-i definiraju sve operacije koje se mogu izvršiti nad podacima, neke od njih su kreiranje novih pacijenata i ažuriranja njihovih nalaza.



Slika 3.3. Prikaz REST API arhitekture [15]

4. PROGRAMSKO RJEŠENJE APLIKACIJE

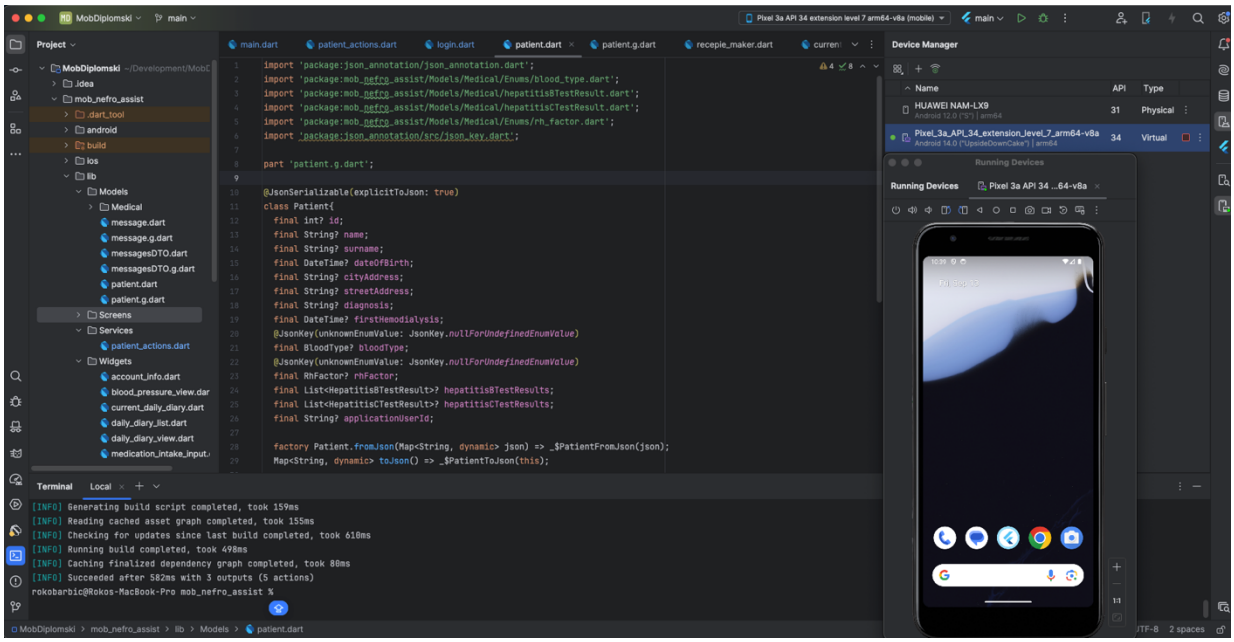
U ovom poglavlju bit će opisano programsko rješenje aplikacije, uključujući detalje o implementaciji, korištenim alatima i tehnologijama.

4.1. Korištene programske tehnologije

Proces razvoja ovog sustava zahtjevao je korištenje raznih tehnologija kako bi se postigli željeni rezultati.

4.1.1. Razvojno okruženje

Korištene su Rider i Intellij kao razvojna okruženja (eng. IDE, *Integrated Development Environment*). JetBrains Rider podržava .NET Core i .NET Framework, što omogućava razvoj web aplikacije i servisa unutar jednog okruženja. Pruža veliku ponudu alata za analizu koda, debuggiranje i testiranje, te integraciju sa sustavima za verzioniranje poput Git-a. Intellij IDEA je svestrano razvojno okruženje koje podržava razvoj u različitim programskim jezicima i razvojnim okvirima (eng. *framework*), uključujući Flutter. Korištenjem Flutter plugina i s integracijom Android SDK, IntelliJ IDEA omogućava efikasan razvoj i testiranje mobilnih aplikacija unutar jednog okruženja.



Slika 4.1. Intellij IDEA i Android emulator

4.1.2. Tehnologije za razvoj web aplikacije

Za razvoj web dijela projekta korišten je .NET 8 razvojni okvir koji je razvio Microsoft, otvorenog je koda, što znači da je slobodan za korištenje te da ga zajednica programera

kontinuirano nadograđuje [16]. .Net koristi C# kao primarni programski jezik, a to je objektno orijentirani jezik koji ima automatsko upravljanje memorijom koristeći mehanizam „*garbage colector*“, također C# je strogo tipiziran (engl. *strongly-typed*) i podržava sigurnost tipova (engl. *type-safe*). To bi značilo da svaka varijabla u C#-u mora imati definiran tip podatka, jednom definiran tip varijable se više ne može promijeniti, dok sigurnost tipova osigurava da program ne radi s podacima na način koji nije ispravan ili očekivan.

Za kreiranje modela i entiteta u bazi podataka korišten je Entity Framework, jer omogućuje lako mapiranje objekata iz koda na tablice u relacijskoj bazi podataka, samim time pojednostavljuje rad s podacima i eliminira potrebu za pisanjem SQL (engl. *Structured Query Language*) upita.

Prezentaciju podataka korisniku, odnosno korisničko sučelje aplikacije isprogramirano je korištenjem HTML-a (engl. *HyperText Markup Language*), koji je prezentacijski jezik za izradu strukture web stranica. Za stiliziranje i izgled korišten je CSS (engl. *Cascading Style Sheets*), što omogućuje definiranje stila i izgleda HTML elemenata. Konkretno, korišten je Bootstrap, CSS okvir (engl. *framework*) koji pruža već gotove dizajnerske komponente i stilove. Bootstrap olakšava izradu responzivnih i modernih korisničkih sučelja, smanjujući vrijeme razvoja i potrebu za pisanjem vlastitog CSS koda. Također, korišten je i JavaScript, skriptni jezik koji ima mogućnost manipulacije HTML elementima, validacije formi, te samim time pridonosi interaktivnosti i dinamici korisničkog sučelja.

4.1.3. Tehnologije za razvoj mobilne aplikacije

Za razvoj mobilne aplikacije korišten je Flutter, razvojni SDK (engl. *Software Development Kit*) kojeg je razvio Google. Flutter je otvorenog koda, što znači da je slobodan za korištenje, a zajednica programera i Google ga kontinuirano nadograđuju. Flutter omogućuje razvoj aplikacija za više platformi (Android, iOS, web i desktop) koristeći jedan zajednički kod, čime se značajno smanjuje vrijeme i složenost razvoja. Kao cjelovit SDK, Flutter uključuje sve što je potrebno za razvoj poput renderinga, API-ja, pa sve do alata za testiranje i razvoj aplikacija [17].

Dart je programski jezik koji se koristi unutar Fluttera. Dart je objektno orijentirani jezik, dizajniran kako bi bio brz i efikasan za razvoj korisničkih sučelja. Jedna od glavnih prednosti Fluttera je njegov widget-based pristup. Svaki element u korisničkom sučelju, poput gumba, tekstualnih polja ili slika predstavlja se kao widget, što omogućava visoku fleksibilnost i ponovnu upotrebu komponenti unutar aplikacije.

4.1.4. Baza podataka

U ovom projektu korištena je PostgreSQL baza podataka, to je relacijska baza podataka koja organizira podatke u tablice koje su međusobno povezane putem relacija. Kao što je već navedeno, koristi se Entity Framework (engl. ORM, *Object Relational Mapper*) za rad s bazom podataka.

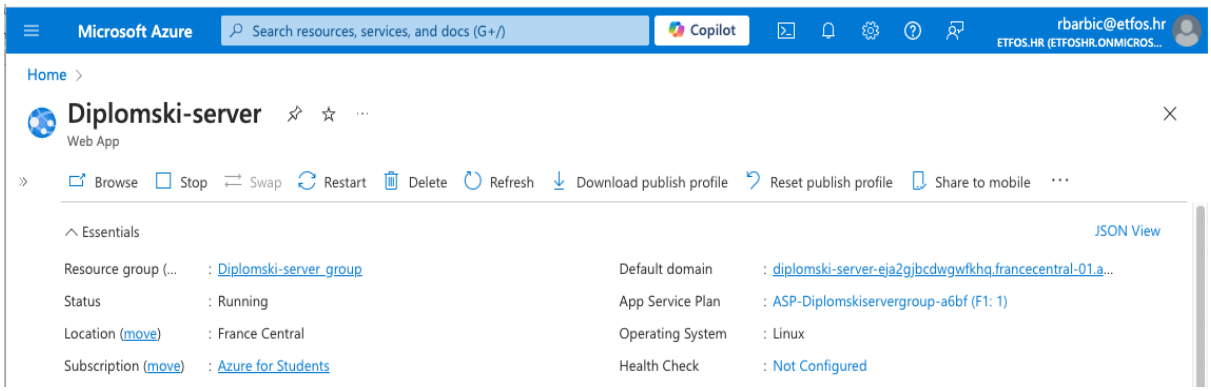
Tijekom razvoja baza podataka PostgreSQL bila je pokrenuta na lokalnom stroju putem Docker kontejnera, što je omogućilo brzu i jednostavnu instalaciju te izolaciju baze podataka od ostatka sustava. Za upravljanje i pregled baze podataka korišten je Azure Data Studio, alat koji nudi sučelje za izvršavanje SQL upita i pregled strukture baze podataka.

4.1.5. Posluživanje i postavljanje aplikacije

Nakon završetka lokalnog razvoja web aplikacije i njezinih servisa, pristupilo se fazi postavljanja (engl. *deploy*) aplikacije na produkcijsko okruženje kako bi bila dostupna te tako olakšala razvoj mobilne aplikacije.

Za hostanje baze podataka korištena je Render platforma. Render nudi usluge udomljavanja baza podataka u oblaku, a u ovom slučaju korišten je besplatni plan koji omogućuje rad s bazom podataka u trajanju od 30 dana [18]. PostgreSQL baza po[dataka je migrirana na Render kako bi bila dostupna aplikaciji putem internetske veze. Ova platforma pruža jednostavno rješenje za upravljanje bazom podataka, uključujući automatske sigurnosne kopije i skaliranje u slučaju potrebe za većim resursima.

Za postavljanje web aplikacije korišten je Azure Web App. Azure Web App je usluga u okviru Microsoft Azure platforme koja omogućuje jednostavno postavljanje, upravljanje i skaliranje web aplikacija u oblaku. Zbog studentskih pogodnosti korišten je besplatni F1 plan koji omogućava hostanje aplikacije bez troškova, uz određene resurse i ograničenja. Slika 4.2 prikazuje zaslon u Microsoft Azure gdje su vidljivi detalji o podignutom Web Appu imena “Diplomski-server”.



Slika 4.2. Prikaz Microsoft Azure platforme

4.2. Prikaz programskog rješenja

4.2.1. Upravljanje podatcima

U strukturi web aplikacije gdje je definiran Model, nalazi se direktoriji u kojemu su smještene klase koje opisuju izgled entiteta koji se spremaju u bazu podataka. Svaka od tih klasa odgovara tablici odnosno entitetu u bazi, a relacije entiteta definirane su putem stranih ključeva (engl. *foreign key*).

Za autentifikaciju i autorizaciju korisnika koristio se .NET Identity sustav, koji automatski stvara osnovne tablice, poput AspNetUsers, za pohranu podataka o korisnicima. Također, dodatno je proširen Identity sustav tako da se povezuje s modelima kao što su Doktor, Medicinsko osoblje i Pacijent. Konkretno, svaki od ovih entiteta pohranjuje "ApplicationUserId" iz tablice AspNetUsers, čime se osigurava povezanost između podataka specifičnih za doktore, medicinsko osoblje i pacijente s podacima o njihovim korisničkim računima.

Definirani su i modeli specifični za medicinske podatke koji uključuju laboratorijske nalaze, gdje se bilježi povijest i rezultati svih laboratorijskih testova pacijenata. Napravljeni su modeli za hepatitis B i C testove, koji prate rezultate različitih markera svakog testa od pojedinog pacijenta. Model za propisane lijekove omogućava praćenje lijekova koje je doktor propisao svakom pacijentu, dok je model za dnevnike bilježio dnevni unos lijekova, krvne tlakove i kilažu pacijenta, što je omogućilo sveobuhvatan uvid u stanje pacijenta kroz vrijeme. Informacije o dnevnom unosu lijekova i krvnim tlakovima su zasebni modeli te su povezani s dnevnikom uz pomoć stranog ključa. Dok su ostali medicinski modeli povezani s entitetom Pacijent. Na taj način osigurala se povezanost između različitih medicinskih podataka i

odgovarajućeg pacijenta. Na primjer, svaki laboratorijski nalaz, propisani lijek i dnevnik izravno je povezan s pacijentom putem “PatientId” stranog ključa, čime je postignuta jasna veza između podataka i pacijenta kojem pripadaju.

Korištenje Entity Frameworka je pojednostavilo pristup bazi podataka i manipulaciju podacima kroz LINQ (engl. *Language Integrated Query)* upite, bez potrebe za ručnim pisanjem SQL naredbi.

Slika 4.3. Model baze podataka

```
public class DataContext : IdentityDbContext<IdentityUser>
{
    protected readonly IConfiguration Configuration;

    rokobarbic
    public DataContext(DbContextOptions<DataContext> options, IConfiguration configuration) : base(options)
    {
        Configuration = configuration;
    }

    6 usages
    public DbSet<Doctor> Doctors { get; set; }
    12 usages
    public DbSet<Patient> Patients { get; set; }
    8 usages
    public DbSet<LaboratoryAnalysis> LaboratoryAnalyses { get; set; }
    5 usages
    public DbSet<Medication> Medications { get; set; }
    1 usage
    public DbSet<MedicalStaff> MedicalStaff { get; set; }

    1 usage
    public DbSet<ApplicationUser> ApplicationsUsers { get; set; }
    7 usages
    public DbSet<DailyDiary> DailyDiaries { get; set; }
    4 usages
    public DbSet<MedicationIntake>  MedicationIntakes { get; set; }
    2 usages
    public DbSet<Dialysis> DialysisList { get; set; }
    3 usages
    public DbSet<Suggestions> SuggestionsList { get; set; }
    rokobarbic
```

Slika 4.4. Dio programskog koda DataContext klase

DataContext koristi se za interakciju između aplikacije i baze podataka. Ova klasa nasljeđuje IdentityDbContext<IdentityUser>, što omogućava integraciju ASP.NET Identity sustava za autentifikaciju i autorizaciju korisnika, te upravljanje njihovim podacima kroz standardne tablice poput AspNetUsers, AspNetRoles te ostale povezane tablice. Klasa DataContext definira i DbSet modele koji omogućavaju rad s entitetima u bazi podataka, točnije rečeno omogućuje CRUD (Create, Read, Update, Delete) operacije nad tablicama u bazi.

```
protected override void OnConfiguring(DbContextOptionsBuilder options)

{
    if (Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT") == "Development")
    {
        options.UseNpgsql(Configuration.GetConnectionString(name: "DefaultConnection"));
    }
    else
    {
        options.UseNpgsql(connectionString: Environment.GetEnvironmentVariable("POSTGRESQLCONNSTR_WebApiDatabase"));
    }
}
```

Slika 4.5. Izgled OnConfiguring metode

Metoda *OnConfiguring* koristi se za konfiguraciju opcija povezivanja s bazom podataka. Metoda omogućuje dinamičku promjenu teksta za uspostavu veze (engl. *connection string*), ovisno o okruženju u kojem se aplikacija izvršava. Connection string je tekstulani niz koji sadrži sve potrebne informacije za uspostavljanje veze između aplikacije i baze podataka. Te informacije obično uključuju lokaciju baze podataka, njeno ime, korisničko ime i lozinku te mogu saržavati još neke dodatne parametre.

4.2.2. Prijava i registracija korisnika

Kao što je već spomenuto, korišten je ASP.NET Identity sustav za prijavu i registraciju korisnika. Ovaj sustav je prikladan jer omogućuje upravljanje korisničkim računima, autentifikaciju putem različitih protokola (npr. lozinke, OAuth, tokeni) te autorizaciju pristupa različitim dijelovima aplikacije na temelju korisničkih uloga (engl. *Role*) [19]. Jedna od ključnih prednosti ovog sustava je što se lozinke u bazi podataka enkriptiraju, čime se povećava sigurnost korisničkih podataka.

U web aplikaciji korištene su gotove komponente unutar ASP.NET Identity sustava, koje su zatim prilagođene za specifične potrebe ovog projekta. Ove komponente generirane su putem scaffolding alata, koji automatski stvara potrebne stranice (View), kontrolere i logiku za prijavu, registraciju i upravljanje korisničkim računima. U okviru ovog projekta, web stranica ima mogućnost prijave i registracije za doktore i medicinsko osoblje. Pri registraciji, korisnik mora odabrati određenu ulogu, a na temelju te uloge određuju se prava pristupa različitim dijelovima aplikacije. Prilikom registracije potrebno je unijeti ime, prezime, adresu e-pošte i lozinku. ASP.NET Identity sustav podržava validaciju korisničkih podataka, uključujući minimalne zahtjeve za jačinu lozinke te jedinstvenost korisničkog imena. Nakon uspješne registracije, korisnički podaci se pohranjuju u bazu podataka, a korisnik može pristupiti aplikaciji s vlastitim računom. Slika 4.6 prikazuje metodu na čiji se poziv spremaju podaci o novom korisniku ukoliko su ispravno uneseni.

```
rokobarbic
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");
    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
    if (ModelState.IsValid)
    {
        var user = CreateUser();

        await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
        await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
        var result = await _userManager.CreateAsync(user, Input.Password);

        if (result.Succeeded)
        {
            _logger.LogInformation( message: "User created a new account with password.");
            if (!string.IsNullOrEmpty(Input.Role))
            {
                await _userManager.AddToRoleAsync(user, Input.Role);
            }
            else
            {
                await _userManager.AddToRoleAsync(user, Role.Role_MedicalStaff);
            }
            if (Input.Role == "Doctor")
            {
                var doctorProfile = new Doctor
                {
                    Name = Input.Name,
                    Surname = Input.Surname,
                    ApplicationUserId = user.Id,
                    ApplicationUser = user,
                };
                _context.Doctors.Add(doctorProfile);
                await _context.SaveChangesAsync();
            }
            if (Input.Role == "MedicalStaff")
            {
                var staffProfile = new MedicalStaff
                {
                    Name = Input.Name,
```

Slika 4.6. Dio metode za registraciju doktora i medicinskog osoblja

Mobilna aplikacija s druge strane ne nudi mogućnost registracije novih pacijenata već registraciju obavlja doktor ili medicinsko osoblje putem web aplikacije. Taj proces funkcionira tako da doktor ili medicinsko osoblje unosi podatke novog pacijenta, kao što su ime, prezime, adresa e-pošte, dijagnoza, datum prve hemodijalize te rezultati testova na hepatitis B i C. Nakon uspješne registracije, pacijentu se dodjeljuje automatski generirana lozinka koju može koristiti za prijavu u mobilnu aplikaciju zajedno s njegovom adresom e-pošte. Na slici je 4.7 je prikazana metoda za stvaranja novih pacijenta te njihovo spremanje u bazu podataka ukoliko su uneseni isprvni podaci.

```
[HttpPost]
2 usages  rokobarbic
public async Task<IActionResult> CreatePatient(PatientDTO patientDto)
{
    if (ModelState.IsValid)
    {
        var pwd :IPassword?  = new Password(9).IncludeLowercase().IncludeUppercase().IncludeSpecial().IncludeNumeric();
        var password :string?  = pwd.Next();


        ApplicationUser user = new ApplicationUser
        {
            UserName = patientDto.Name,
            Email = patientDto.Email,
        };

        var resultOfCreation = await _userManager.CreateAsync(user, password);

        if (!resultOfCreation.Succeeded)
        {
            return BadRequest();
        }

        Patient patient = new Patient
        {
            Name = patientDto.Name,
            Surname = patientDto.Surname,
            DateOfBirth = patientDto.DateOfBirth.ToUniversalTime(),
            CityAddress = patientDto.CityAddress,
            StreetAddress = patientDto.StreetAddress,
            Diagnosis = patientDto.Diagnosis,
            FirstHemodialysis = patientDto.FirstHemodialysis.ToUniversalTime(),
            BloodType = patientDto.BloodType,
            RhFactor = patientDto.RhFactor,
            HepatitisBTestResults = patientDto.HepatitisBTestResults,
            HepatitisCTestResults = patientDto.HepatitisCTestResults,
            ApplicationUserId = user.Id
        };
        _context.Patients.Add(patient);
        await _context.SaveChangesAsync();
```

Slika 4.7. Dio metode za registraciju novih pacijenata

Kontrola pristupa ostvaruje se putem atributa [Authorize] koja se dodjeljuje kontroleru, gdje se može definirati koji korisnici imaju pravo pristupa određenim akcijama ili dijelovima aplikacije. Slika 4.8 prikazuje jednu od upotrebu atributa [Authorize] koja kontroleru za upravljanje pacijenatima daje mogućnost samo doktorima i medicinskom osoblju da pristupe određenim magučnostima, poput funkcije sa slike 4.8.

```
[Authorize(Roles ="MedicalStaff,Doctor")]
7 usages  rokobarbic
public class ViewPatientController : Controller
{
```

Slika 4.8. Prikaz dodjele atributa [Authorize]

Na slici 4.9 je prikazan dio koda za autentifikaciju korisnika u mobilnu aplikaciju. Prvi korak u procesu prijave korisnika realiziran je pomoću HTTP POST zahtjeva. Mobilna aplikacija šalje korisničke podatke, korisničko ime i lozinku, na unaprijed definirani URL, gdje se ti podaci prenose u JSON formatu. Za kodiranje podataka koristi se metoda *jsonEncode*().

```
try {
  Response loginResponse = await client.post(
    'https://$baseUrl$loginPath',
    data: jsonEncode(body),
    options: Options(
      headers: {'Content-Type': 'application/json'},
    ),
  );

  if (loginResponse.statusCode == 200) {
    String aplicationPatientId = loginResponse.data;

    final response = await client.get(
      'https://$baseUrl$getPatientInfoPath',
      options: Options(
        headers: {'Content-Type': 'application/json'},
      ),
      queryParameters: {
        'aplicationPatientId': aplicationPatientId,
      },
    );

    if (response.statusCode == 200) {
      Map<String, dynamic> data = response.data;
      Patient patient = Patient.fromJson(data);
      print(patient);
      await _prefs.setBool('isLoggedIn', true);
      await _prefs.setString('aplicationPatientId', aplicationPatientId);

      Navigator.pushReplacement(
        context,
        MaterialPageRoute(
          builder: (context) => MyHomePage(
            key: UniqueKey(),
```

Slika 4.9. Dio funkcije za validiranje korisničkih podataka za prijavu

Ako sustav uspješno validira korisničke podatke, povratni odgovor s API-ja sadrži jedinstveni identifikator korisnika, nazvan “aplicationPatientId”. Ovaj identifikator predstavlja jedinstveni ključ pacijenta u bazi podataka, koji omogućuje aplikaciji da razlikuje korisnike te pristupi specifičnim podacima pacijenta. Statusni kod odgovora mora biti 200, što označava uspješno izvršenje prijave.

Kada je aplikacija primila jedinstveni identifikator, koristi se dodatni HTTP GET zahtjev kako bi se dohvatili svi relevantni podaci o pacijentu. Ovaj zahtjev uključuje “aplicationPatientId” kao parametar, gdje će poslužiteljska strana sustava pretražiti i dohvatiti podatke iz baze

vezanih uz pacijenta čiji indetifikator odgovara predanome. Rezultat ovog zahtjeva vraća podatke u JSON formatu, koje aplikacija potom pretvara u objekt tipa Patient, koristeći unaprijed definiranu metodu *fromJson*(). Slika 4.10 prikazuje automatski generiranu metodu *_$PatientFromJson*.

```
Patient _$PatientFromJson(Map<String, dynamic> json) => Patient(
      id: (json['id'] as num?)?.toInt(),
      name: json['name'] as String?,
      surname: json['surname'] as String?,
      dateOfBirth: json['dateOfBirth'] == null
          ? null
          : DateTime.parse(json['dateOfBirth'] as String),
      cityAddress: json['cityAddress'] as String?,
      streetAddress: json['streetAddress'] as String?,
      diagnosis: json['diagnosis'] as String?,
      firstHemodialysis: json['firstHemodialysis'] == null
          ? null
          : DateTime.parse(json['firstHemodialysis'] as String),
      bloodType: $enumDecodeNullable(_$BloodTypeEnumMap, json['bloodType'],
          unknownValue: JsonKey.nullForUndefinedEnumValue),
      rhFactor: $enumDecodeNullable(_$RhFactorEnumMap, json['rhFactor'],
          unknownValue: JsonKey.nullForUndefinedEnumValue),
      hepatitisBTestResults: (json['hepatitisBTestResults'] as List<dynamic>?)
          ?.map((e) => HepatitisBTestResult.fromJson(e as Map<String, dynamic>))
          .toList(),
      hepatitisCTestResults: (json['hepatitisCTestResults'] as List<dynamic>?)
          ?.map((e) => HepatitisCTestResult.fromJson(e as Map<String, dynamic>))
          .toList(),
      applicationUserId: json['applicationUserId'] as String?,
    );  // Patient
```

Slika 4.10. Metoda fromJson

Metode *fromJson* i *toJson* automatski su generirane za sve klase pomoću Dart paketa json_serializable. Ovaj paket omogućuje automatizirano serijaliziranje i deserializiranje objekata u JSON format [20]. Sve klase modela u mobilnom projektu koriste ovaj paket za automatsko generiranje.

4.2.3. Programsko rješenje na strani poslužitelja

Poslužiteljska strana aplikacije podijeljena je u više različitih kontrolera, pri čemu svaki od njih predstavlja klasu koja nasljeđuje ili ControllerBase ili Controller. Klase koje nasljeđuju ControllerBase koriste se za stvaranje API endpointa, izlažući funkcionalnosti aplikacije vanjskim korisnicima i vraćajući podatke u formatu poput JSON-a. Ovi kontroleri su ključni za mobilnu aplikaciju koja dohvaća podatke preko navedenog RESTful API-ja. S druge strane, klase koje nasljeđuju Controller koriste se unutar same web aplikacije za vraćanje view-ova (korisničkih sučelja) u obliku HTML-a ili drugih formata koji omogućuju prikaz podataka

korisniku. Ovakva podjela omogućava fleksibilnost i skalabilnost sustava, pri čemu se poslužiteljska strana može koristiti za oba tipa klijenata web i mobilne aplikacije.

Slika 4.11 prikazuje dio kontrolera za upravljanje pacijentima. Kontroler, nazvan PatientController, nasljeđuje klasu ControllerBase, što znači da je dizajniran za rukovanje API pozivima, a ne za vraćanje korisničkih sučelja (View). U konstruktoru kontrolera, DataContext i SignInManager se primaju kao parametri kako bi bili dostupni unutar klase. DataContext omogućuje kontroleru pristup podacima iz baze, a SignInManager upravljanje autentifikacijom korisnika.

```
[ApiController]
[Route( template: "[controller]")]
rokobarbic *
public class PatientController : ControllerBase
{
    private readonly DataContext _context;
    private readonly SignInManager<IdentityUser> _signInManager;


    rokobarbic *
    public PatientController(DataContext context, SignInManager<IdentityUser> signInManager)
    {
        _context = context;
        _signInManager = signInManager;
    }

    [HttpGet( template: "getPatientInfo")]
    new *
    public async Task<IActionResult> GetPatientInformation(string aplicationPatientId)
    {
        Patient? patient = await _context.Patients // DbSet<Patient>
            .Include( navigationPropertyPath: a :Patient => a.HepatitisBTestResults) // IIncludableQueryable<Patient,ICollection<...>?>
            .Include( navigationPropertyPath: b :Patient => b.HepatitisCTestResults) // IIncludableQueryable<Patient,ICollection<...>?>
            .FirstOrDefaultAsync(c :Patient => c.ApplicationUserId == aplicationPatientId); // Task<Patient?>

        if (patient == null)
        {
            return NotFound();
        }
        return Ok(patient);
    }
```

Slika 4.11. Prikaz jednog dijela PatientController klase

Također se na slici može vidjeti metoda *GetPatientInformation*, označena atributom HttpGet("getPatientInfo")], predstavlja HTTP GET metodu koja služi za dohvaćanje informacija o pacijentu putem API-ja. Ova metoda prima parametar aplicationPatientId, koji se koristi za pretraživanje pacijenta u bazi podataka. Pomoću LINQ (engl. *Language Integrated Query*), koji omogućuje upite nad podacima u C# kodu, metoda dohvaća pacijenta zajedno s njegovim rezultatima testova za hepatitis B i C. LINQ omogućuje jednostavniji i intuitivniji

način za pisanje upita koji se prevode u SQL i izvršavaju u bazi podataka. Konkretno, *Include*() metode omogućuju dohvaćanje povezanih podataka, čime se osigurava da su podaci o testovima odmah dostupni zajedno s pacijentom. *FirstOrDefaultAsync* vraća prvi rezultat koji odgovara kriteriju ili praznu vrijdnost (engl. null), ako pacijent ne postoji, što se onda koristi za rukovanje potencijalnim greškama. Ako pacijent ne postoji, metoda vraća HTTP status 404 Not Found, dok u suprotnom vraća HTTP status 200 OK, zajedno s podacima o pacijentu u JSON formatu. Na ovaj način isprogramirani su svi kontroleri unutar projekta koji nasljeđuju ControllerBase klasu.

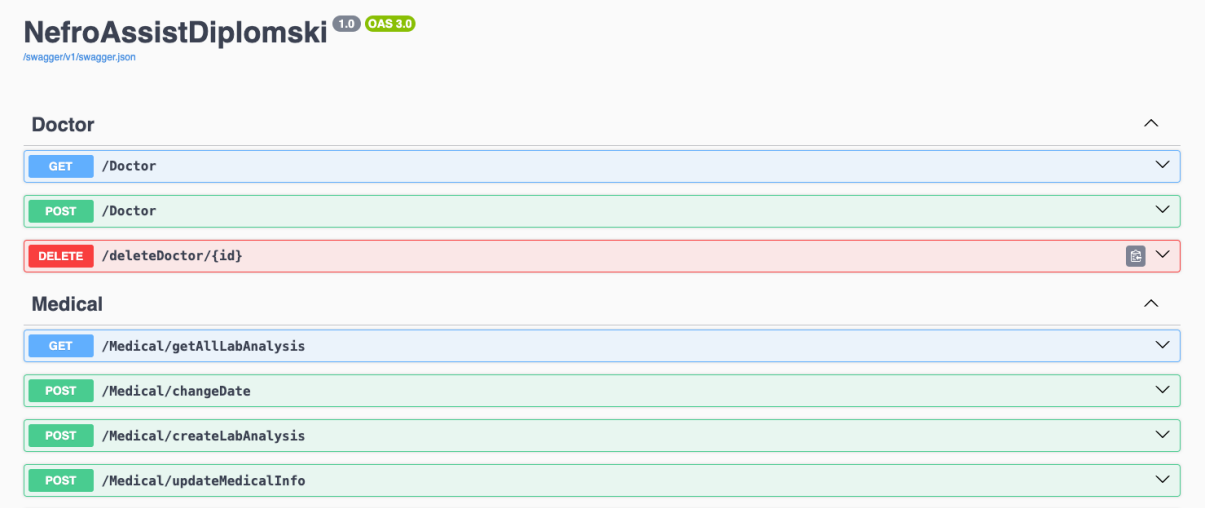
Slika 4.12 prikazuje ViewMedicationController koji nasljeđuje klasu Controller, jer se ona koristi za rad s HTTP zahtjevima i vraćanjem rezultata uključujući vraćanje View-ova. Potrebno je istaknuti također da je ograničen pristup kontroleru uz pomoć atributa [Authorize] koji osigurava da samo korisnici s definiranom ulogom "Role_Doctor" mogu pristupiti metodama ovoga kontrolera. Metoda *PrescribeNewMedication* označena atributom [HttpGet] služi za prikazivanje stranice s formom za unos novog lijeka pacijentu, te se prilikom kreiranja stranice prosljeđuje objekt MedicationDTO. DTO (engl. *Data transfer object*) predstavlja objekte koji služe za prijenos informacija i najčešće se stvaraju kako bi popunili određene zahtjeve za prijenost podataka prema API-ju.

```
[Authorize(Roles =Role.Role_Doctor)]
2 usages  rokobarbic
public class ViewMedicationController : Controller
{
    private readonly DataContext _context;

    rokobarbic
    public ViewMedicationController(DataContext context)
    {
        _context = context;
    }

    [HttpGet]
    2 usages  rokobarbic
    public async Task<IActionResult> PrescribeNewMedication(int patientId)
    {
        MedicationDTO medicationDto = new MedicationDTO { PatientId = patientId };
        return View(medicationDto);
    }

    [HttpPost]
    2 usages  rokobarbic
    public async Task<IActionResult> PrescribeNewMedication(MedicationDTO medicationDto)
    {
        if (ModelState.IsValid)
        {
            var controller = new MedicalController(_context);
            await controller.PrescribeMedication(medicationDto);
            return RedirectToAction("PatientDetailsPage", controllerName: "ViewPatient", routeValues: new { id = medicationDto.PatientId });
        }
        return View(medicationDto);
    }
```

Slika 4.12. Prikaz dijela ViewMedicationControllera klase

Metoda s istim imenom ali drukčijim atributom [HttpPost] koristi se za obradu podataka nakon što je korisnik popunio i poslao formu. Ovjde se provjerava validnost modela putem *ModelState.IsValid*, što osigurava da su svi potrebni podaci u formi ispravno uneseni, ako nisu korisnik se vraća na istu stranicu kako bi mogao ponovno ispuniti formu. Ako je model ispravan, instancira se novi objek MedicalController u kojemu je definirana metoda za dodavanje novog lijeka u bazu podataka. Nakon uspješno izvedenog dodavanja lijeka korisnik se preusmjerava na stranicu s detaljima o pacijentu koristeći metodu *RedirectToAction*. Svi kontroleri i njihove metode su napravljene na sličan način kao što je prikazano na slikama 4.11 i 4.12, te se samo mijenjaju podaci koji se šalju ili stranice koje se prikazuju, ali pricip rada je isti.

Za bolji prikaz, dokumentaciju i testiranje API-ja korišten je Swagger alat, alat koji omogućuje automatsko generiranje i pregled API-ja. Swagger je rješenje otovrenog koda koje se jednostavno integrira u aplikaciju, a njegova konfiguracija vrši se u datoteci *Program.cs*. Prednost korištenja Swaggera je u tome što na jednom mjestu pruža pregled svih API endpointa, zajedno s detaljima poput parametara koje endpointi prihvaćaju i strukture podataka koje vraćaju. Slika 4.13 prikazuje sučelje alata swagger koje prikazuje API endpointe.



Slika 4.13. Prikaz sučelja alata swagger

4.2.4. Programsko rješenje na strani klijenta - mobilna aplikacija

Mobilna aplikacija je napisana po sličnom pricipu poput MVC, tako da su u jednoj datoteci definirane sve metoda za komunikaciju sa poslužiteljem te postoje direktoriji u kojima su definirani modeli i direktoriji u kojemu su definirani zasloni i widgeti. *Patient* klasa je definirana na slici 4.14.

```
part 'patient.g.dart';

@JsonSerializable(explicitToJson: true)
class Patient{
  final int? id;
  final String? name;
  final String? surname;
  final DateTime? dateOfBirth;
  final String? cityAddress;
  final String? streetAddress;
  final String? diagnosis;
  final DateTime? firstHemodialysis;
  @JsonKey(unknownEnumValue: JsonKey.nullForUndefinedEnumValue)
  final BloodType? bloodType;
  @JsonKey(unknownEnumValue: JsonKey.nullForUndefinedEnumValue)
  final RhFactor? rhFactor;
  final List<HepatitisBTestResult>? hepatitisBTestResults;
  final List<HepatitisCTestResult>? hepatitisCTestResults;
  final String? applicationUserId;

  factory Patient.fromJson(Map<String, dynamic> json) => _$PatientFromJson(json);
  Map<String, dynamic> toJson() => _$PatientToJson(this);
```

Slika 4.14. Prikaz klase *Patient*

Jedna od datoteka u direktoriju Modeli je Pacijent koja definira klasu pacijent. Iz ovog primjera može se vidjeti korištenje paketa json_serializable, tako što postoji linija koda koja govoti “part 'patient.g.dart'” i na taj način povezuje klasu s generiranom datotekom “patient.g.dart” u kojoj su definirane funkcije za serijalizaciju i deserijalizaciju. Također se na slici 4.10 može vidjeti kako izgleda dio sadržaja “patient.g.dart” datoteke.

```
Future<Patient?> getPatientInformation(String aplicationPatientId) async {
  Dio client = Dio();

  final response = await client.get(
    'https://$baseUrl$getPatientInfoPath',
    options: Options(
      headers: {'Content-Type': 'application/json'},
    ),
    queryParameters: {
      'aplicationPatientId': aplicationPatientId,
    },
  );

  if (response.statusCode == 200) {
    Map<String, dynamic> data = response.data;
    return Patient.fromJson(data);
  } else {
    print('Error: ${response.statusCode}');
  }
  return null;
}
```

Slika 4.15. Prikaz metode getPatientInformation

GetPatientInformation je jedna od metoda koja se nalazi unutar klase PatientActions u kojoj su definirani pozivi prema RESTful API-ju. Ova metoda dohvaća informacije o pacijentu te poziva ranije navedenu funkciju *fromJson* kako bi stvorila novi objek Patient iz dobivenog responsa koji je u JSON formatu.

Slika 4.16 prikazuje kako PatientInfoWidget nasljeđuje *Stateless* widget, što znači da ne može mijenjati svoje stanje nakon inicijalnog prikaza, osim ako se ne rekreira s novim podacima. Nasuprot tome, postoje i *Stateful* widgeti, koji mogu promijeniti svoje stanje bez potrebe za ponovnim kreiranjem cijelog widgeta. PatientInfoWidget prima instancu klase Patient kao obavezan parametar te je u konstruktoru prosljeđuje varijabli patient. Metoda build određuje kako će se widget prikazati, vraćajući cijelu strukturu korisničkog sučelja. Ovdje metoda kreira Padding widget, koji osigurava da je sadržaj udaljen 16 piksela od rubova zaslona. Unutar Padding widgeta koristi se ListView, koji omogućuje vertikalno pomicanje po zaslonu (engl. *scroll*). ListView sadrži popis widgeta koji su postavljeni jedan ispod drugog, a svaki od tih widgeta ispunjava specifičnu funkciju. Ovaj princip ugnježđivanja widgeta unutar drugih omogućava modularno i fleksibilno kreiranje korisničkog sučelja, dok pojedini widgeti unutar ListView prikazuju informacije vezane za varijablu patient, a funkcije poput *_buildPatientInfo*() vraćaju widget koji će biti prikazani.

Ovakav princip dohvaćanja podataka putem metoda definiranih u datoteci PatientActions te njihovo prosljeđivanje u StatelessWidget ili StatefulWidget, ovisno o potrebama aplikacije, slijedi se kroz cijeli mobilni projekt.

Slika 4.17 prikazuje instanciranje modela pod nazivom "gemini-1.5-flash", uz korištenje API ključa za autentifikaciju. Ovaj model je konfiguriran s visokim razinama filtriranja sadržaja, što uključuje zaštitu od potencijalnog uznemiravanja i govora mržnje. Takve postavke pomažu u zaštiti korisnika od neželjenih ili neprimjerenih rezultata.

Funkcija talkToGemini sa slike 4.18 koristi ovaj model i poziva metodu generateContent, koja prihvaća tekstualni upit kao ulaz. U ovom slučaju, upit traži od modela da generira recept koji odgovara određenoj vrsti hrane, koja se prima kao tekstualni parametar funkcije. U upitu je isto tako definirano da je recept namjenjen za bolesnike sa KBB i visokim krvnim tlakom, kako bi model mogao napraviti zdravi recept. Gemini model će u teksutanom obliku vratiti recept i ta će se vrijednost predati widgetima unutar aplikacije koji su odgovorni za prikaz informacija. Kako bi se uopće mogao koristiti gemini model potrebno je dodati paket google_generative_ai i stvoriti api key na njihovoj web stranici.

```
class PatientInfoWidget extends StatelessWidget {
  final Patient patient;

  PatientInfoWidget({Key? key, required this.patient}) : super(key: key);
  final PatientActions patientActions = PatientActions();

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(16.0),
      child: ListView(
        children: <Widget>[
          const SizedBox(height: 20),
          _buildPatientInfo(),
          const Divider(height: 20, thickness: 2),
          const Text(
            'Hepatitis B Test Results:',
            style: TextStyle(fontWeight: FontWeight.bold),
          ),  // Text
          const SizedBox(height: 10),
          ..._buildHepatitisBTestResults(patient.hepatitisBTestResults),
          const Divider(height: 20, thickness: 2),
          const Text(
            'Hepatitis C Test Results:',
            style: TextStyle(fontWeight: FontWeight.bold),
          ),  // Text
          const SizedBox(height: 10),
          ..._buildHepatitisCTestResults(patient.hepatitisCTestResults),
        ],  // <Widget>[]
      ),  // ListView
    );  // Padding
  }
```

Slika 4.16. Prikaz PatientInfoWidget widgeta

```
geminiModel = GenerativeModel(
    model: 'gemini-1.5-flash',
    apiKey: apiKey,
    safetySettings: [
      SafetySetting(HarmCategory.harassment, HarmBlockThreshold.high),
      SafetySetting(HarmCategory.hateSpeech, HarmBlockThreshold.high),
    ]);  // GenerativeModel
```

Slika 4.17. Prikaz instanciranja geminiModela

```
Future<String> talkToGemini(String? typeOfFood) async {
  final content = [
    Content.text(
        'Give a recipe that will be from this country $typeOfFood or if
  ];
  final response = await geminiModel!.generateContent(content);
  return response.text ?? 'The recipe cant be made, try again later.';
}
```

Slika 4.18. Metoda za slanje propta gemina modelu

4.2.5. Programsko rješenje na strani klijenta, web aplikacija

Klijentska strana web aplikacije je definiran unutar View dijela MVC-a. Generira se na temelju podataka koje šalje Controller. Controller ne mora nužno slati nikakve podatke kada kreira novi prikaz stranice, no u slučaju kad ih šalje onda se ti podatci spremaju u *@model* unutar datoteke View.

U prikazanom primjeru sa slike 4.19, View koristi podatke iz modela definiranog s linijom *@model List<NefroAssistDiplomski.Models.Models.Patient.Patient>*. Ova linija govori kako View prihvaća listu objekata tipa Patient. Lista sadrži sve potrebne informacije o pacijentima koje View treba prikazati. U glavnom dijelu View-a koristi se *foreach* petlja kako bi se prošlo kroz svaki element iz liste pacijenata i prikazali detalji za svakog pacijenta u tablici. Na primjer, prikazuju se ID, ime, prezime i datum rođenja pacijenta unutar HTML strukture tablice (<table>).

Za prikaz tablice koristi se DataTables biblioteka, koja dodaje interaktivne značajke HTML tablicama, poput mogućnosti pretraživanja, sortiranja po različitim stupcima i prikaza određenog broja zapisa unutar tablice. Na slici 4.19, tablica se sastoji od četiri stupca koji prikazuju podatke o pacijentu i jednog stupca koji sadrži akcije koje je moguće izvesti nad svakim pacijentom. Ove akcije uključuju prikaz dodatnih informacija, ažuriranje podataka pacijenta, dodavanje laboratorijskog nalaza ili dodavanje izvješća hemodijalize.

DataTables se koristio na svim mjestima u projektu gdje je bilo potrebno ispisati veći broj podataka istog tipa.

```
@model List<NefroAssistDiplomski.Models.Models.Patient.Patient>

@{
    ViewBag.Title = "Patients List";
    Layout = "_Layout";
}

<h2>@ViewBag.Title</h2>

<!-- Table of Patients -->
<table class="table table-bordered table-striped" id="patientTable">
    <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Surname</th>
            <th>Date of Birth</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var patient in Model)
        {
            <tr>
                <td>@patient.Id</td>
                <td>@patient.Name</td>
                <td>@patient.Surname</td>
                <td>@patient.DateOfBirth.ToString(format: "yyyy-MM-dd")</td>
                <td style="...">
                    <a asp-controller="ViewPatient" asp-action="PatientDetailsPage" asp-route-id="@patient.Id" class="btn btn-info btn-sm">More Info</a>
                    <a asp-controller="ViewPatient" asp-action="" asp-route-id="@patient.Id" class="btn btn-warning btn-sm">Update</a>
                    <a asp-controller="ViewLabAnalysis" asp-action="AddNewLabAnalysis" asp-route-patientId="@patient.Id" class="btn btn-success btn-sm" style="background-color: #
                    <a asp-controller="ViewDialysis" asp-action="AddNewDialysis" asp-route-patientId="@patient.Id" class="btn btn-success btn-sm" style="background-color: #3CB371
                </td>
            </tr>
        }
    </tbody>
</table>
```

Slika 4.19. Dio koda za prikazivanje tablice pacijenata

Korištenje formi unutar HTML-a služi za unos i slanje podataka na poslužitelja, a primjer jedne forme prikazan je na slici 4.20. Kako bi podaci poslani putem forme mogli biti obrađeni, potrebno je definirati kontroler i njegovu metodu, što se postiže atributima *asp-controller* i *asp-action.* Također je potrebno odrediti tip metode, npr. method="post", što znači da će se podaci slati poslužitelju u tijelu zahtjeva, za razliku od metode GET, gdje se podaci šalju putem URL-a. Komunikacija i slanje podataka prema kontrolerima kroz cijeli web projekt ostvaruje se pomoću formi.

```
<div class="col-md-4">
    <div class="card-body">
        <form asp-controller="Medical" asp-action="UpdateMedicalInfo" method="post">
            <input type="hidden" name="Id" value="@Model.LaboratoryAnalysis.Id" />

            <div class="form-group">
                <label for="PresentIllnesses"><strong>Present Illnesses:</strong></label>
                <textarea id="PresentIllnesses" name="PresentIllnesses" class="form-control" rows="3">@Model.LaboratoryAnalysis.PresentIllnesses</textarea>
            </div>

            <div class="form-group">
                <label for="PhysicalExamination"><strong>Physical Examination:</strong></label>
                <textarea id="PhysicalExamination" name="PhysicalExamination" class="form-control" rows="3">@Model.LaboratoryAnalysis.PhysicalExamination</textarea>
            </div>

            <button type="submit" class="btn btn-primary mt-3" style="background-color: #8FBC8F; border-color: white">Save Medical Info</button>
        </form>
    </div>
</div>
```

Slika 4.20. Prikaz jednostavne forme

Slika 4.21 prikazuje korištenje JavaScripta unutar View-a za pregled laboratorijskih nalaza. Funkcija prikazana na slici provjerava je li određena vrijednost unutar zadanog raspona te vizualno označava polja koja odstupaju od tog raspona, a uz to nudi skup savjeta koje liječnik može poslati pacijentu. Kako bi ovakva funkcionalnost bila izvediva, bilo je potrebno definirati granične vrijednosti određenih parametara, poput hemoglobina, čije bi vrijednosti trebale biti između 100 i 120 g/L. Na slici 4.22 prikazano je postavljanje provjere za ovaj parametar. Vrijednosti izvan navedenih granica mogu ukazivati na potencijalne kardiovaskularne rizike. Kako bi se razvoj takvih događaja usporio ili spriječio, liječniku se prikazuju već gotovi savjeti vezani uz pojedine parametre iz nalaza. Ovi savjeti omogućuju pacijentima bolje razumijevanje njihovog zdravstvenog stanja i pomažu u poduzimanju potrebnih mjera opreza. Ovaj način korištenja JavaScripta u projektu dodaje interaktivnost korisničkom sučelju, istovremeno smanjuje potrebu za slanjem dodatnih zahtjeva poslužitelju. Funkcija je dio šire skripte koja pomaže liječniku u interpretaciji laboratorijskih nalaza te pomaže u pružanju preporuka na temelju rezultata.

```
function checkRange(id, min, max, suggestionIndex) {
    let element = document.getElementById('Model_LaboratoryAnalysis_' + id);
    let suggestionOption = document.querySelector(`#Suggestions option[value="${suggestionIndex}"]`);

    if (element && suggestionOption) {
        let value = parseFloat(element.textContent);

        if (value < min || value > max) {
            element.style.border = '2px solid red';
            suggestionOption.style.display = '';
        }
    }
}
```

Slika 4.21. Prikaz funkcije u JavaScriptu

```
checkRange('hemoglobin', 100, 120, 1);
```

Slika 4.22. Prikaz poziva funcije *checkRange*

Slika 4.23 prikazuje dio koda koji stvara graf. Prvo se dodaju x i y osi koje definiraju kako će se podaci prikazivati duž osi vremena (x) i vrijednosti (y). Linija se crta pomoću D3-ove metode *line*(), koja povezuje točke na temelju podataka. Osim linije, na svaku točku se dodaju crvene kružnice koje predstavljaju pojedinačne vrijednosti laboratorijskih nalaza. Klikom na kružnicu prikazuje se dodatne informacije o toj točki, poput datuma i odgovarajuće vrijednosti.

```
// Add the x-axis
svg.append("g")
    .attr("transform", `translate(0,${height})`)
    .call(d3.axisBottom(x));

// Add the y-axis
svg.append("g")
    .call(d3.axisLeft(y));

// Add the line
svg.append("path")
    .datum(data)
    .attr("fill", "none")
    .attr("stroke", "steelblue")
    .attr("stroke-width", 2)
    .attr("d", d3.line()
                .x(d => x(d.date))
                .y(d => y(d.value))
    );

// Add points to the line
svg.selectAll(".dot")
    .data(data)
    .enter().append("circle")
    .attr("class", "dot")
    .attr("cx", d => x(d.date))
    .attr("cy", d => y(d.value))
    .attr("r", 5)
    .attr("fill", "red")
    .on("click", function(event, d) {
        isTooltipVisible = !isTooltipVisible;
            d3.select(".tooltip").style("opacity", isTooltipVisible ? .9 : 0)
                .html(`Date: ${formatDate(new Date(d.date))}<br>Value: ${d.value}`)
                .style("left", (event.pageX + 5) + "px")
                .style("top", (event.pageY - 28) + "px");
        });
```

Slika 4.23. Korištenja D3.js za vizualizaciju

Za vizualizaciju grafa koji prikazuje vrijednosti parametara iz laboratorijskih nalaza kroz vrijeme, korišten je D3.js. D3.js (engl. *Data-Driven Documents*) je JavaScript biblioteka koja omogućuje stvaranje dinamičkih i interaktivnih vizualizacija podataka. Korištenjem SVG-a (engl. *Scalable Vector Graphics*) moguće je kreirati grafove koji se mogu skalirati i prilagoditi različitim veličinama prikaza bez gubitka kvalitete.

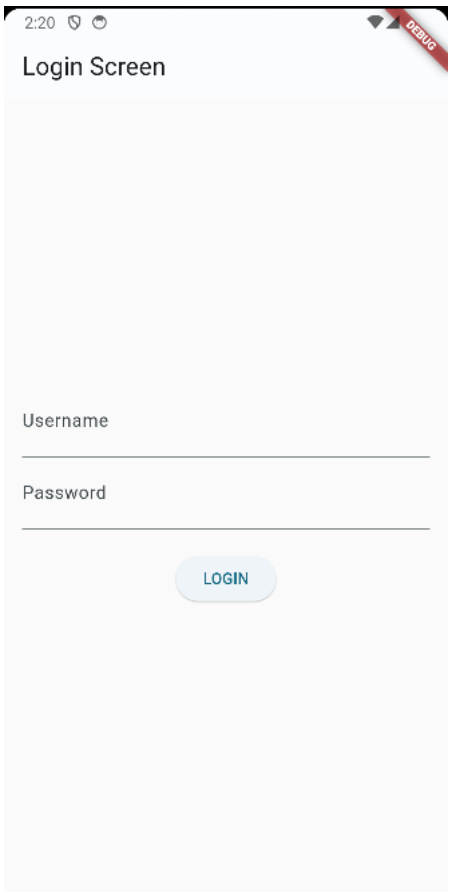
Ovakav način vizualizacije pruža korisnicima, liječnicima, mogućnost da intuitivno pregledavaju laboratorijske podatke, identificiraju trendove i donose odluke na temelju tih podataka.

5. PRIKAZ I ISPITIVANJE RADA APLIKACIJE

U ovom poglavlju detaljno je prikazan rad web i mobilne aplikacije, pri čemu su opisane sve ključne funkcionalnosti i mogućnosti koje ona nudi korisnicima.

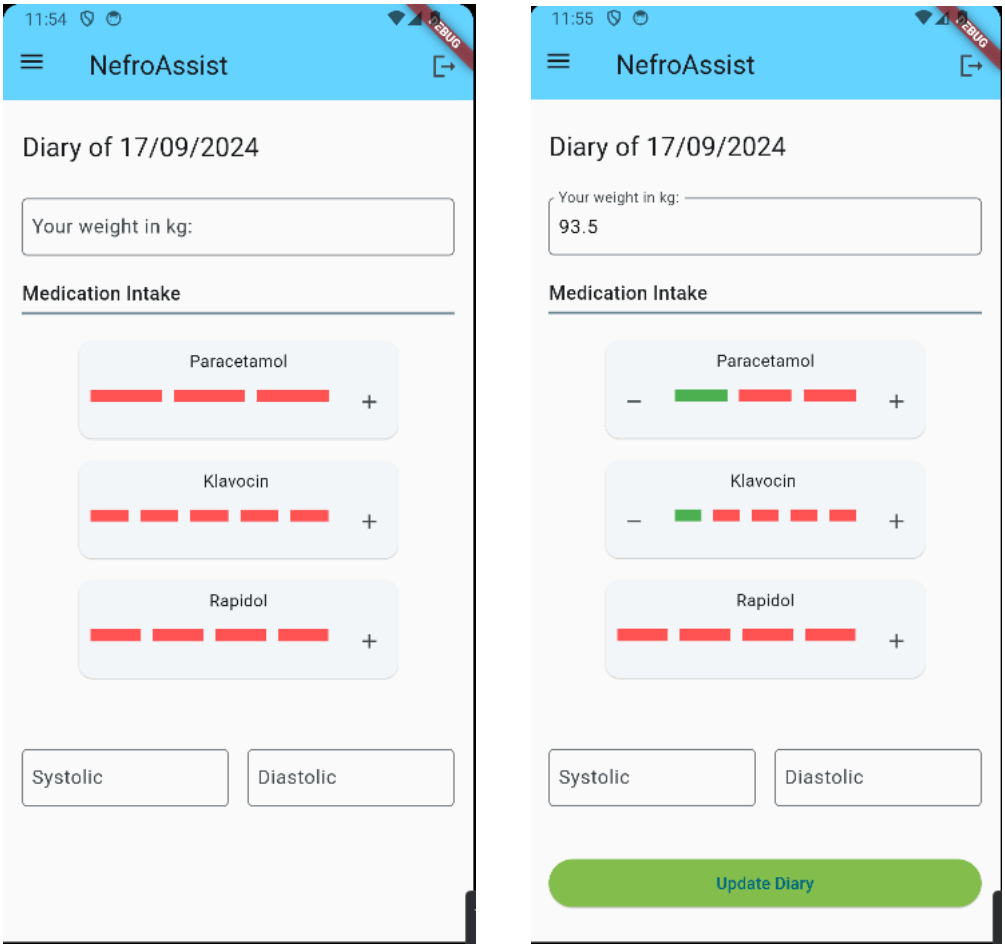
5.1. Mobilna aplikacija

Mobilna aplikacija je višeplatformska, ali u ispitivanju rada aprilikacije korišten je Android emulator. Pokretanja aplikacije prikazuje se zaslon za prijavu ukoliko već korisnik nije prijavljen u aplikaciju. Zaslon za prijavu je prikazan na slici 5.1.



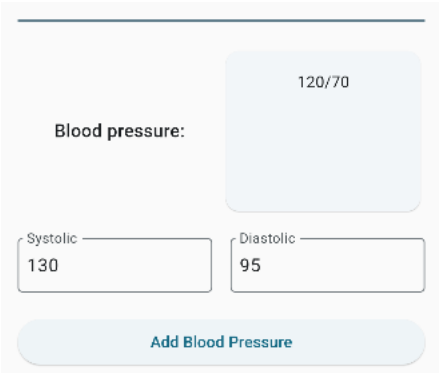
Slika 5.1. Zaslon za prijavu korisnika

Nakon uspješne prijave, otvara se zaslon koji prikazuje dnevnik za taj dan, s mogućnošću unosa kilaže, terapije i krvnog tlaka. Prilikom unosa novih vrijednosti, korisniku se prikazuje gumb za spremanje i ažuriranje dnevnika. Dio zaslona koji se odnosi na unos lijekova je pomičan, što omogućuje korisniku pregled i unos svih njegovih lijekova. Slika 5.2 prikazuje dva zaslona jedan je onaj kada su vrijednosti u dnevniku nepromjenjene, a drugi je nakon unosa izmjerene kilaže i lijekova.



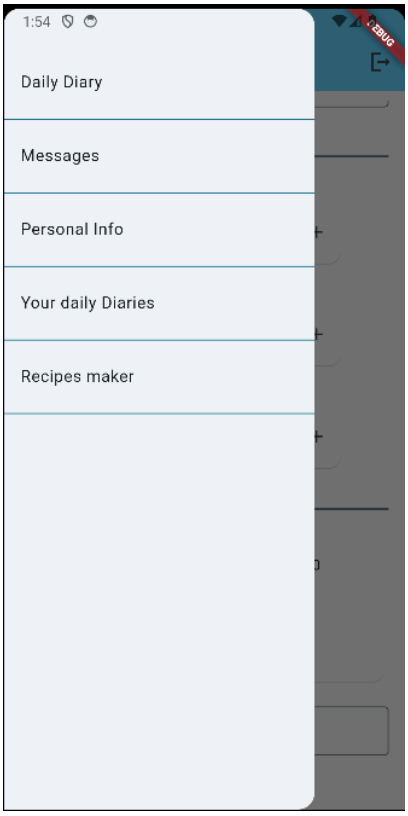
Slika 5.2. Prikaz dnevnika

Slika 5.3 prikazuje navigaciju u aplikaciji, gdje pacijent može vidjeti sve dostupne opcije. Opcija „DailyDiary“ vodi korisnika na početni zaslon, gdje može upravljati dnevnikom za taj dan.

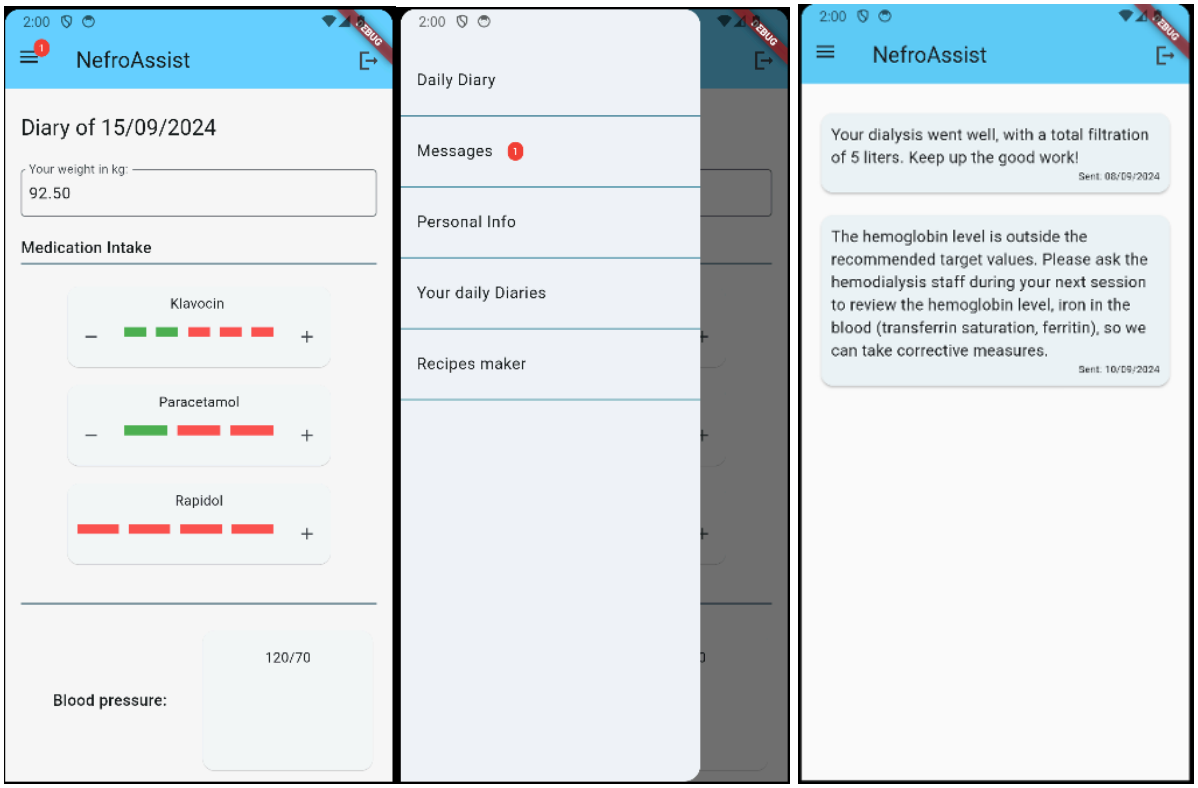


Slika 5.3. Prikaz unosa krvnog tlaka u dnenvik

Nakon unosa krvnog tlaka pojavljuje se widget koji prikazuje sve unesene vrijednosti krvnog tlaka, ovaj widget je vidljiv samo ako postoje spremljeni tlakovi. Također, omogućeno je pomicanje po zaslonu kako bi se prikazali svi uneseni tlakovi.



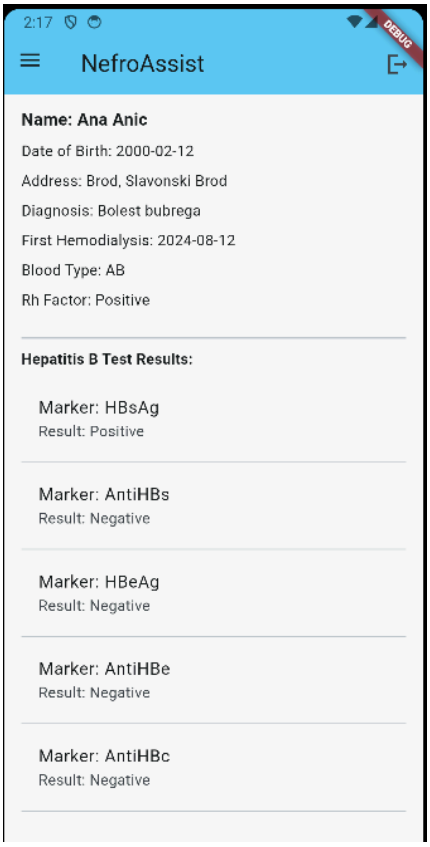
Slika 5.4. Prikaz widget drawer-a



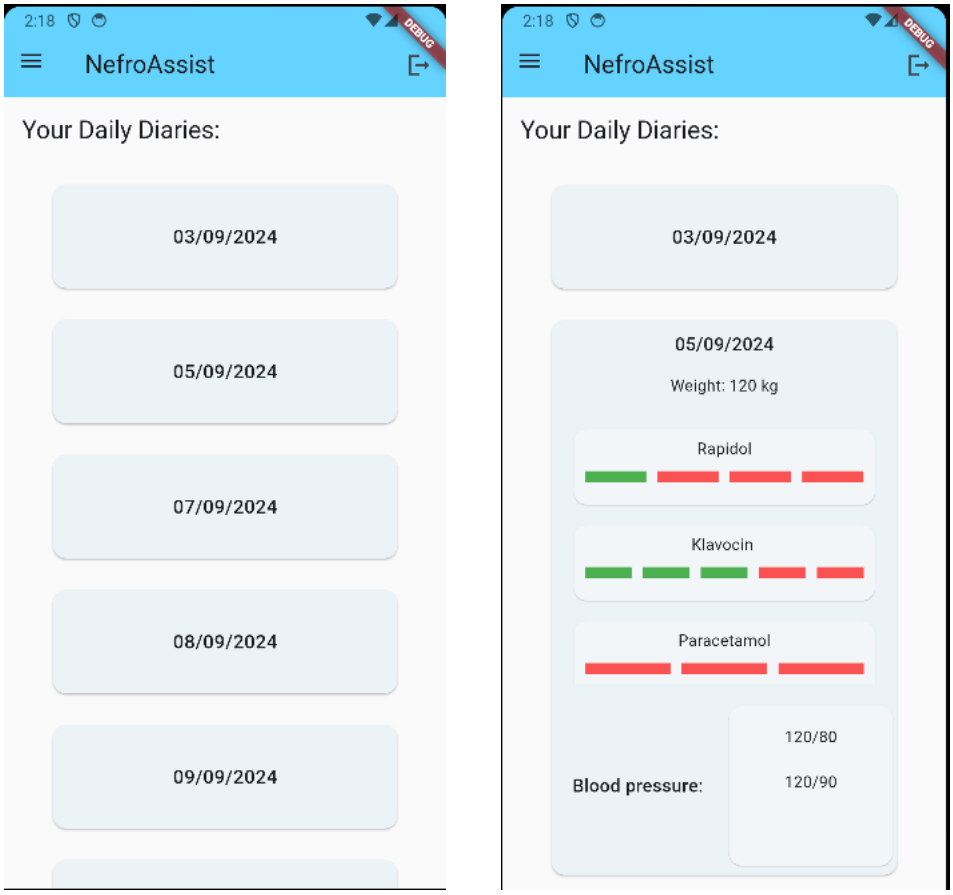
Slika 5.5. Prikaz obavjesti o novoj poruci i zaslon sa porukama

Na zaslonima sa slike 5.5 vidljivo je kako pacijent dobiva obavijesti o nepročitanim porukama i savjetima od doktora. Kada se odabere opcija za navigaciju na zaslon s porukama i savjetima, prikazat će se sve poruke, s najnovijom porukom smještenom na dnu.

Slika 5.6 prikazuje zaslon s osobnim podacima, dok su zaslon sa svim dnevnicima i njihovi detalji također prikazani na slici 5.7 Korisniku su prikazani datumi dnevnika, a dodirivanjem jednog od datuma otvaraju se detalji tog dnevnika. Također, omogućeno je pomicanje po zaslonu kako bi svi dnevnici bili vidljivi.

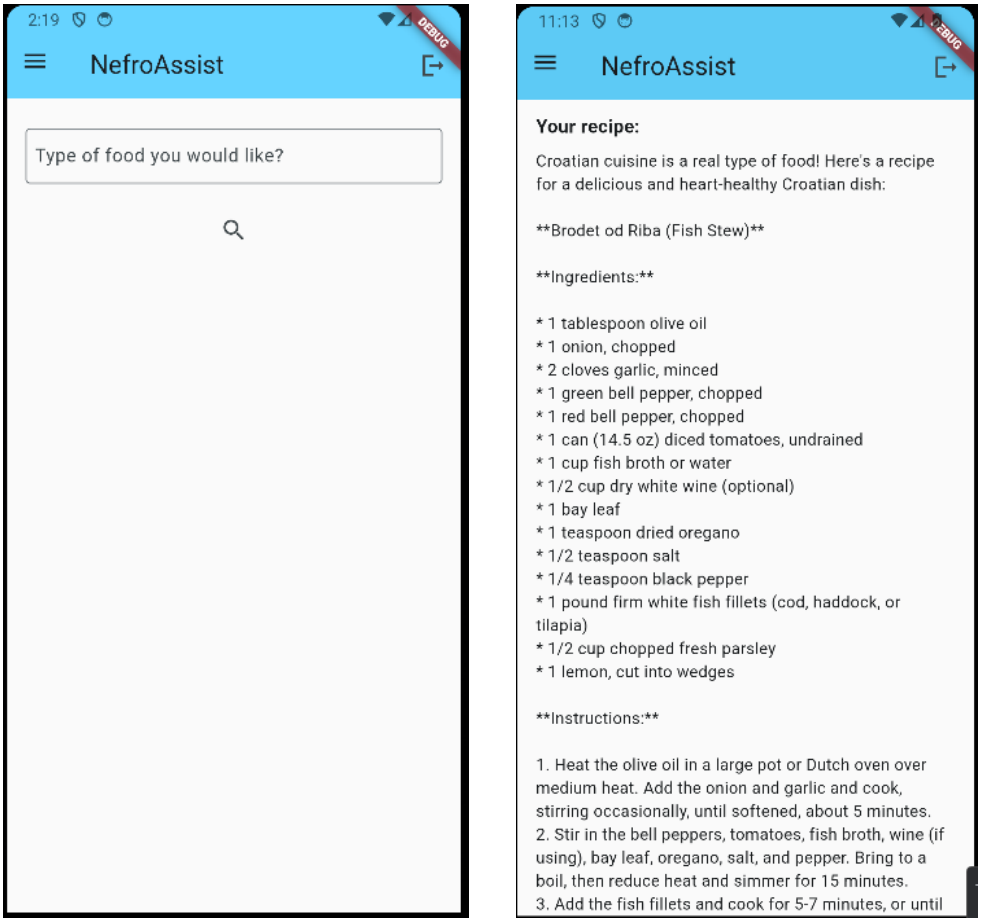


Slika 5.6. Prikaz zaslona s osobnim podatcima



Slika 5.7. Prikaz zaslona sa svim dnevnicima

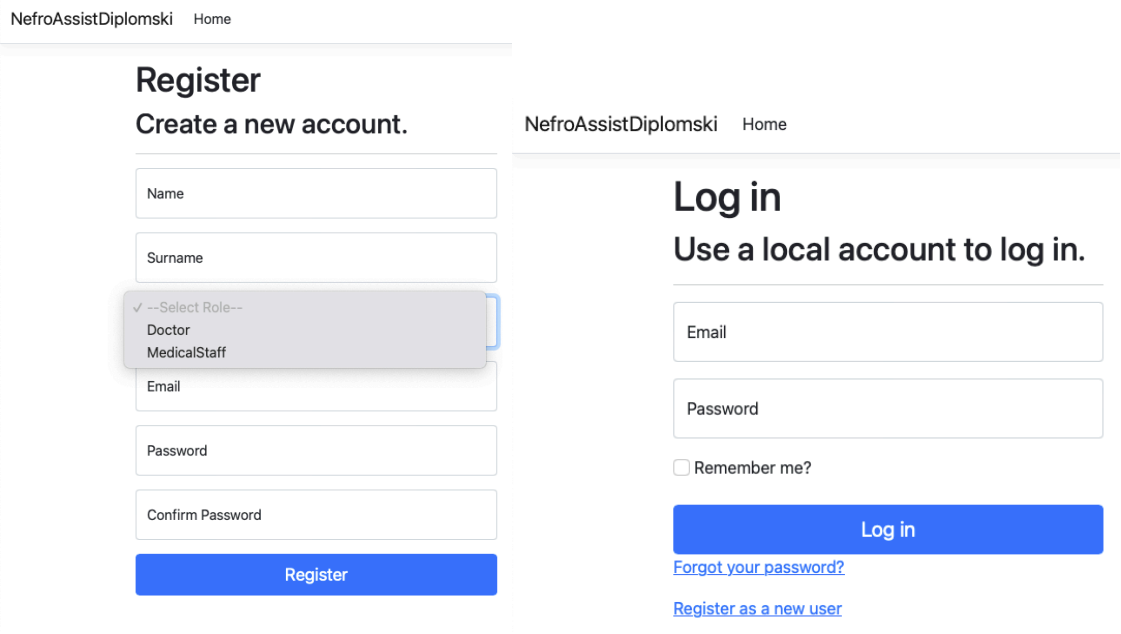
Na zaslonu za stvaranje recepta koji je vidljiv na slici 5.8 korisnik unosi vrstu hrane koju želi pripremiti. Pritiskom na ikonu tražilice započinje proces stvaranja recepta. Kada AI sustav Gemini generira recept, prikazat će se na zaslonu. Korisnik u gornjem desnom uglu aplikacije uvijek ima mogućnost odjave iz aplikacije.



Slika 5.8. Prikaz korištenja pametne kuharice

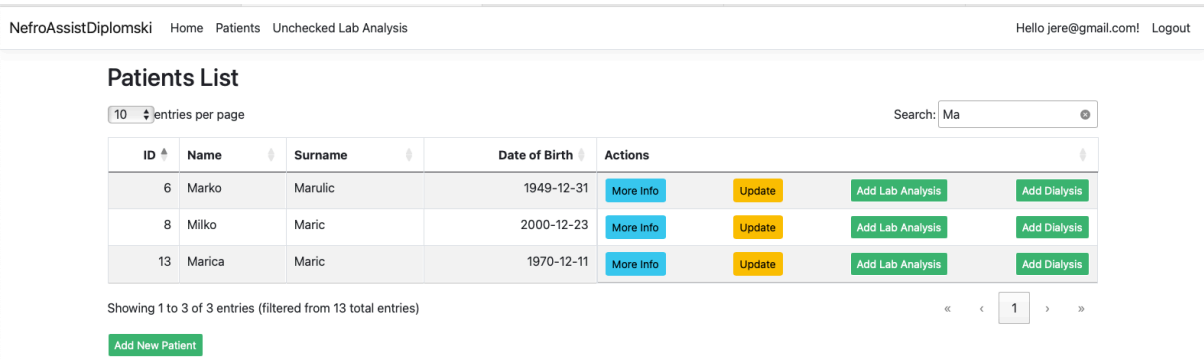
5.2. Web aplikacija

Prilikom pokretanja aplikacije korisnik se nalazi na početnom prozoru koji prikazuje općenite informacije o ovom projektu. Na navigacijskoj traci korisnik ima mogućnost odabira između prijave i registracije. Na slici 5.9 prikazani su prozori za prijavu i registraciju. Također, vidljivo je da korisnik pri registraciji može odabrati vrstu korisnika, bilo da je doktor ili medicinsko osoblje.



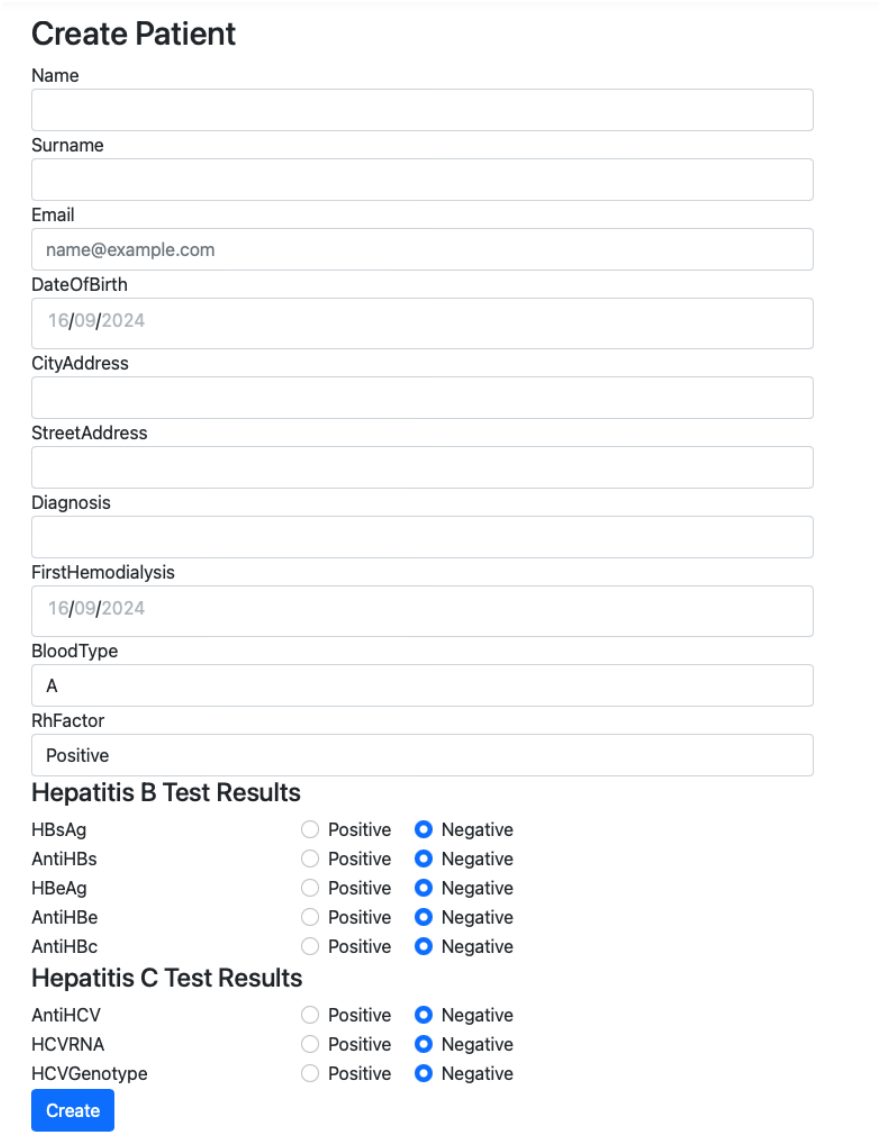
Slika 5.9. Prikaz registracije i prijave korisnika

Nakon uspješne prijave u aplika ciju korisnik može pregledati sve pacijente. Odabirom opcije „Pacijenti" na navigacijskoj traci, otvara se zaslon prikazan na slici 5.10, koji sadrži tablicu svih pacijenata. Korisnik ima mogućnost pretraživanja pacijenata unutar te tablice, što je vidljivo na slici 5.10. jer se pretražuju pacijenti sa početnim slovima "Ma".



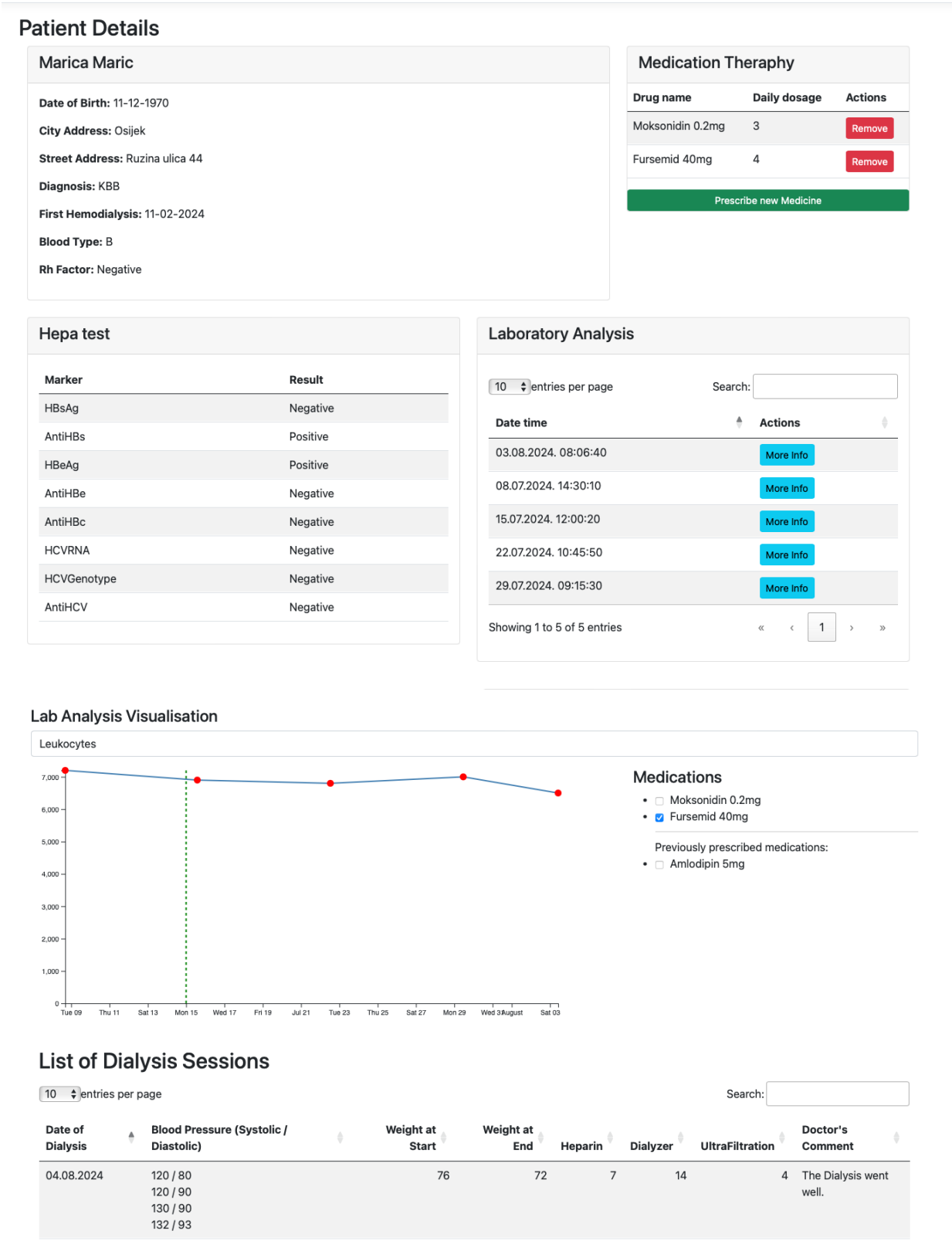
Slika 5.10. Prikaz stranice za pregled pacijenata

Pritiskom na gumb za dodavanje novog pacijenta, korisnik će biti odveden na stranicu sa slike 5.11, gdje će biti potrebno ispuniti podatke o pacijentu.



Slika 5.11. Prikaz dodavanja novog pacijenta

Nakon stvaranja pacijenta, moguće je pregledati detalje o pacijentu. Stranica s detaljima pacijenta, prikazana na slici 5.12, omogućuje doktoru različite funkcionalnosti. Doktor može propisati novi lijek ili ukloniti postojeći iz terapije. Također, može pregledavati laboratorijske nalaze i izvještaje s dijalize. Na stranici je dostupna i vizualizacija vrijednosti pojedinih parametara iz nalaza kroz vrijeme, što pomaže u praćenju promjena. Uz to, doktor može vidjeti trenutne i prethodne terapije kako bi lakše razumio učinak lijekova.



Slika 5.12. Prikaz stranice s detaljima o pacijentu

Ako je korisnik aplikacije doktor, može odabrati nepregledane nalaze na navigacijskoj traci. Bit će preusmjeren na stranicu s tablicom svih laboratorijskih nalaza koji su uneseni u sustav

od strane medicinskog osoblja, ali još nisu pregledani od strane doktora. Navedena stranica je prikazana na slici 5.13.

Laboratory Analysis that have to be checked

10 entries per page Search:

Date time	Name	Surname	Actions
03.08.2024. 08:06:40	Marica	Maric	More Info
08.07.2024. 14:30:10	Marica	Maric	More Info
15.07.2024. 12:00:20	Marica	Maric	More Info
22.07.2024. 10:45:50	Marica	Maric	More Info
29.07.2024. 09:15:30	Marica	Maric	More Info
29.08.2024. 09:15:30	Mirko	Filipovic	More Info
29.08.2024. 09:15:30	Ana	Anic	More Info
30.08.2024. 08:06:40	Ana	Anic	More Info

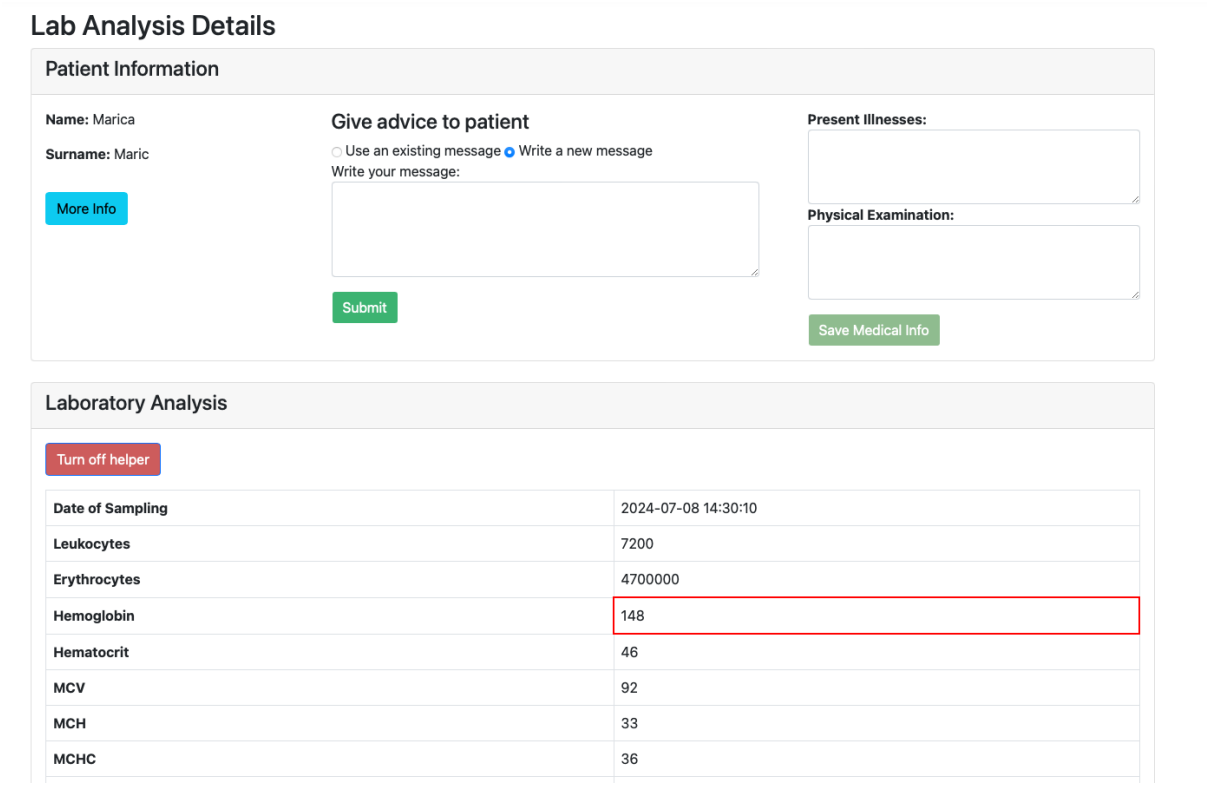
Showing 1 to 8 of 8 entries « ‹ 1 › »

Slika 5.13. Stranica sa nepregledanim nalazima

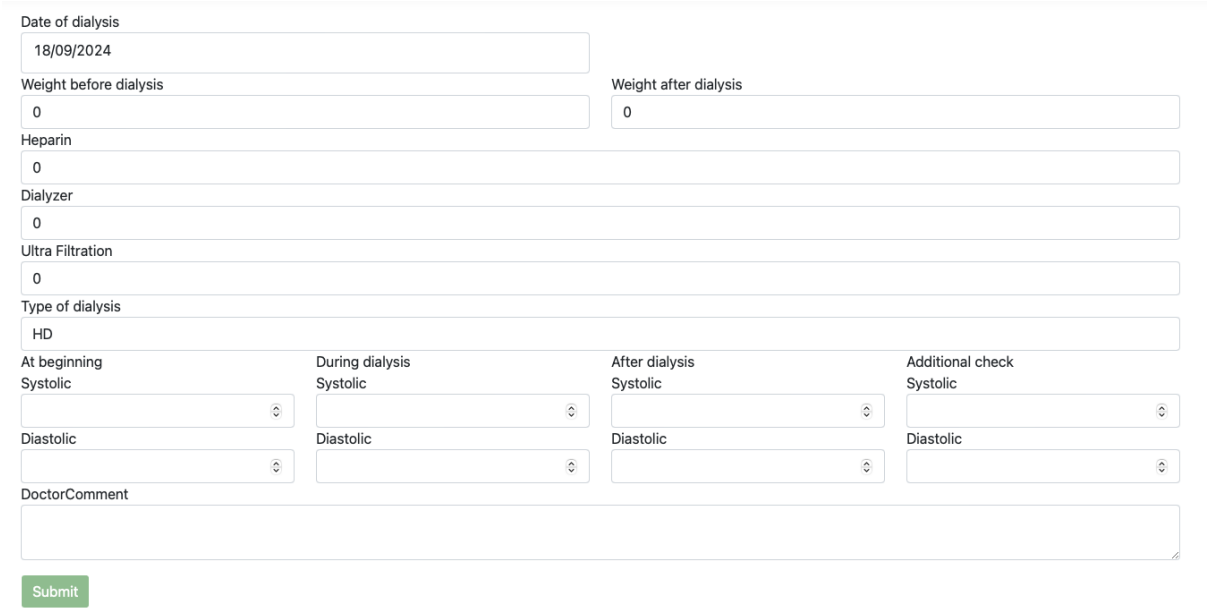
Klikom na gumb za više informacija, doktor će biti preusmjeren na stranicu s detaljima o određenom nalazu, prikazanu na slici 5.14.

Na stranici s detaljima o nalazu, doktor treba ispuniti tekstualna polja za trenutno stanje bolesti pacijenta i opis njegovog fizičkog pregleda. Tek nakon unosa tih podataka, nalaz će biti uklonjen s liste nepregledanih nalaza. Također, stranica omogućuje stvaranje sugestija doktoru, ali samo ako on to odabere, sugestije funkcioniraju na način da polja s parametrima koji su izvan željenih vrijednosti budu označena crvenom bojom.

Na istoj stranici, doktor može napisati poruku i savjet pacijentu ili odabrati već gotove preporuke. Ako je uključena pomoć za doktora, prikazat će mu se samo one sugestije koje odgovaraju parametrima koji su izvan željenih vrijednosti. Dok slika 5.15 prikazuje zaslon za unos dijalize.



Slika 5.14. Stranica s detaljima o nalazu



Slika 5.15. Stranica za unos dijalize

5.3. Ispitivanje i analiza

Postoje dva korisnička slučaja za web aplikaciju i jedan za mobilnu aplikaciju. Korisnički slučajevi za web aplikaciju ovise o tome koristi li aplikaciju liječnik ili medicinsko osoblje. Ako aplikaciju koristi medicinsko osoblje, poput medicinske sestre, ono ima mogućnost unosa

novih laboratorijskih nalaza i dnevnika dijalize, kao i pregled postojećih pacijenata. U slučaju da aplikaciju koristi liječnik, on ima iste mogućnosti kao medicinsko osoblje, ali s dodatnim funkcionalnostima. Liječnik može pregledavati laboratorijske nalaze te ostavljati komentare, upravljati terapijama pacijenata i slati poruke sa savjetima pacijentima. Sučelje web aplikacije je jednostavno i intuitivno za korištenje.

Mobilnu aplikaciju koriste pacijenti koji imaju mogućnost zabilježbe unosa lijekova, izmjerenog krvnog tlaka i tjelesne težine, što se sprema u dnevnik. Također, pacijenti mogu pregledavati svoje dnevnike, izvješća o dijalizi i poruke liječnika. Jedna od dodatnih funkcionalnosti je stvaranje zdravih recepata. Mobilna aplikacija je višeplatformska kako bi je mogao koristiti veći broj pacijenata. Korisničko iskustvo aplikacije je osmišljeno tako da bude lako razumljivo i korisnicima pristupačno.

6. ZAKLJUČAK

U ovom diplomskom radu razvijen je sveobuhvatan sustav koji pruža podršku pacijentima u liječenju kronične bubrežne bolesti, s naglaskom na poboljšanje komunikacije između pacijenata i liječnika te praćenje zdravstvenih parametara. Sustav se sastoji od dva dijela. To su web aplikacije za medicinsko osoblje i mobilna aplikacija za pacijente. Web aplikacija, razvijena u .NET 8 razvojnom okruženju, omogućuje liječnicima i medicinskom osoblju unos i praćenje važnih podataka o pacijentima, kao što su laboratorijski nalazi, izvješća o dijalizi te propisane terapije. Ova aplikacija pomaže liječnicima u vizualizaciji prikupljenih podataka, olakšava analizu i donošenje informiranih odluka o daljnjem tijeku liječenja. S druge strane, mobilna aplikacija, razvijena u Flutteru, namijenjena je pacijentima i osmišljena je kako bi im pomogla u praćenju dnevnog unosa terapije, tjelesne težine i krvnog tlaka. Aplikacija omogućuje pacijentima vođenje dnevnika u kojem bilježe izmjerene vrijednosti i praćenje svojih zdravstvenih parametara. Također, pacijenti putem aplikacije dobivaju personalizirane obavijesti i preporuke od strane liječnika, što doprinosi boljoj prilagodbi životnih navika i poboljšanju kvalitete života. Obje aplikacije dijele zajedničku bazu podataka koja je implementirana u PostgreSQL-u. Također, obje koriste isti poslužitelj, koji je povezan s bazom podataka putem Entity Frameworka.

Ispitivanje rada web aplikacije na korisničkim slučajevima doktor i medicinsko osoblje pokazuju mogućnost upravljanja i vizualizacije podataka vezanih uz pacijente. Višeplatformska mobilna aplikacija je ispitana u korisničkom slučaju pacijent te potvrđuje mogućnost dodavanja i pregleda dnevnika, pregleda korisnih savjeta i generiranja zdravih recepta. Moguća unaprjeđenja ovog sustava mogu biti u vidu sigurnosti cijelog sustava i većeg broja opcija za komunikaciju između liječnika i pacijenata.

LITERATURA

[1] T.H. Jafar, P.C. Stark, C.H. Schmid, et al. Progression of Chronic Kidney Disease: the Role of Blood Pressure Control, Proteinuria, and Angiotensin-Converting Enzyme Inhibition: a Patient-Level Meta-Analysis. Ann Intern Med, Vol. 139(4), 2003., pp. 244-252

[2] B. Devčić, B. Vujičić, I. Bubić, L. Orlić, D. V. Šimac, S. Živčić Ćosić, S. Rački, Strukturirani predijalizni edukacijski program – desetogodišnje iskustvo KBC Rijeka, Acta Med Croatica, Vol. 73, 2019., pp. 267-276

[3] Usporavanje napretka KBB, https://poliklinika-aviva.hr/en/advice/hitna-intervencija-kljucna-bitka-za-ocuvanje-bubreznog-zdravlja/ [10.9.2024.]

[4] Informacije o broju bolesnika u RH, https://www.plivazdravlje.hr/vijesti/clanak/37037/Kronicna-bubrezna-bolest-u-Hrvatskoj.html [12.9.2024.]

[5] Statistika mHealth app, https://www.statista.com/statistics/779910/health-apps-available-ios-worldwide/ [7.6.2024]

[6] N.S. Thorat, R.V. Kulkarni, A Review- Role of Mobile Application for Medical Services, Posebno izdanje-FIIIIPM2019, 2019., pp. 43-45

[7] Y. Shi, S. Pu, H. Peng, J. Zhang, Y. Li, X. Huang, C. Song, and Y. Luo, Impact of Mobile Application and Outpatient Follow-Up on Renal Endpoints and Physiological Indices in Patients with Chronic Kidney Disease: A Retrospective Cohort Study in Southwest China, BMC Medical Informatics and Decision Making, 2024., pp. 24-163

[8] NefroCare MyCompanion, https://www.nephrocare.co.uk/mycompanion [10.9.2024.]

[9] P. Kasilingam, A. D. Kaivelikkal, V. Iyer, A Mobile Application for Chronic Kidney Disease (CKD) Diagnosis, Conference: ICMHI 2022: 2022 6th International Conference on Medical and Health Informatic 2022.

[10] Mi Kidney aplikacija, https://www.ehealthireland.ie/case-studies/mi-kidney-app/ [17.9.2024]

[11] Portal zdravlja, https://gov.hr/hr/portal-zdravlja-dostupan-i-na-mobilnim-uredjajima/2340?lang=hr [10.9.2024.]

[12] D.P. Pop, A. Altar, Designing an MVC Model for Rapid Web Application Development, Procedia Engineering, 2013., pp. 69

[13] REST, https://radixweb.com/blog/graphql-vs-rest [10.9.2024.]

[14] RESTful API, https://www.techtarget.com/searchapparchitecture/definition/RESTful-API, [10.9.2024.]

[15] Arhitektura slika, https://sites.psu.edu/annaarsiriy/2019/02/10/restful-api/ [18.9.2024.]

[16] B. Cvijić, P. Ranilović, From .NET Core to .NET 8: A Comprehensive Analysis of Performance, Features, and Migration Pathways, JITA, Vol.14, 2024.

[17] Flutter SDK, https://www.altexsoft.com/blog/pros-and-cons-of-flutter-app-development/, [10.9.2024.]

[18] Render, https://docs.render.com/free, [16.9.2024.]

[19] ASP Identity, https://korzh.com/blog/aspnet-identity-store-user-data-in-claims/ [18.9.2024]

[20] Json serializable paket, https://pub.dev/packages/json_serializable [10.9.2024.]

ŽIVOTOPIS

Roko Barbić rođen je 24. studenog 2000. godine u Osijeku. Pohađao je osnovnu školu Antuna Mihanovića u Osijeku. Nakon toga upisuje Prirodoslovnu matematičku gimnaziju u Osijeku koju završava 2019. godine te nakon toga upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer Računarstvo. Akademske godine 2022./2023. Upisao je prvu godinu diplomskog sveučilišnog studija računarstva, smjer Programsko inženjerstvo. U studenom mjesecu 2023. godine započeo je raditi u Intesa Sanpaolo International Value Services kao student na poziciji Flutter developera.

SAŽETAK

Višeplatformski razvoj mobilne aplikacije za praćenje zdravstvenog stanja nefroloških bolesnika i stvaranje preporuka

U ovom diplomskom radu je opis izazova u nefrologiji i osobinama nefroloških bolesnika, uz naglasak na metode liječenja i usporavanja progresije bolesti. Također, dan je pregled postojećih programskih rješenja za mobilne aplikacije koje pomažu u liječenju nefroloških pacijenata. Predstavljeno je idejno rješenje vlastite višeplatformske aplikacije, te opisana arhitektura i dizajn aplikacije, kao i njezini funkcionalni i nefunkcionalni zahtjevi. Mobilna aplikacija razvijena je u razvojnom okruženju Flutter, a njena svrha je pomoći pacijentima u svakodnevnom praćenju zdravstvenog stanja, olakšanju komunikacije s liječnikom te poboljšanju prehrambenih navika. Web aplikacija je razvijena u .NET 8 okruženju, s ciljem unosa medicinskih podataka od strane liječnika i pružanja podrške u liječenju pacijenata. Obje aplikacije koriste isti poslužitelj i PostgreSQL bazu podataka. Prikaz načina rada i testiranje aplikacije pokazali su da su implementirane funkcionalnosti, te da aplikacija uspješno ispunjava svoje ciljeve u pružanju podrške pacijentima i liječnicima kroz efikasnu razmjenu informacija i praćenje zdravstvenog stanja.

Ključne riječi: višeplatformska aplikacija, stvaranje preporuka, nefrološki bolesnici, Flutter, .Net 8

ABSTRACT

Cross-platform development of a mobile application for monitoring the health of nephrology patients and generating recommendations

This thesis describes the challenges in nephrology and the characteristics of nephrological patients, with an emphasis on methods of treatment and slowing disease progression. It also provides an overview of existing software solutions for mobile applications that aid in the treatment of patients with Chronic kidney disease. A conceptual solution for a cross-platform application is presented, along with the architecture and design of the application, as well as its functional and non-functional requirements. The mobile application was developed in the Flutter development environment, and its purpose is to assist patients in monitoring their health status, facilitating communication with doctors, and improving dietary habits. The web application was developed in the .NET 8 environment, aimed at enabling doctors to enter medical data and support patient treatment. Both applications use the same server and PostgreSQL database. The demonstration and testing of the application have shown that the implemented functionalities are effective and that the application successfully meets its goals of providing support to patients and doctors through efficient information exchange and health monitoring.

Keywords: cross-platform application, generating recommendations, nephrology patients, Flutter, .Net 8

PRILOZI

Prilog 1. Diplomski rad u datoteci .docx

Prilog 2. Diplomski rad u datoteci .pdf

Prilog 3. Programski kod aplikacije