Izrada i testiranje mobilne aplikacije za potporu oboljelima od bolesti štitnjače

Mihačić, Luka

Master's thesis / Diplomski rad

2018

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:200:216789

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij računarstva

IZRADA I TESTIRANJE MOBILNE APLIKACIJE ZA POTPORU OBOLJELIMA OD BOLESTI ŠTITNJAČE

DIPLOMSKI RAD

Luka Mihačić

Osijek, 2018.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 20.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Luka Mihačić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 841 R, 25.09.2017.
OIB studenta:	41891342659
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Krešimir Nenadić
Član Povjerenstva:	Izv. prof. dr. sc. Alfonzo Baumgartner
Naslov diplomskog rada:	Izrada i testiranje mobilne aplikacije za potporu oboljelima od bolesti štitnjače
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U diplomskom radu potrebno je opisati probleme i simptome vezane za bolesti štitnjače, te razraditi model prepoznavanja, praćenja tijeka bolesti i generiranje preporuka oboljelom. Također, treba analizirati i opisati alate i postupke testiranja mobilnih aplikacija prikladnih za Android okolinu, programski ostvariti navedenu mobilnu aplikaciju (sa sučeljem i bazom podataka), te provesti odgovarajuće testove i poboljšanja programskog koda, analizirati ih i opisati.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	20.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 04.10.2018.

Ime i prezime studenta:	Luka Mihačić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 841 R, 25.09.2017.
Ephorus podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada i testiranje mobilne aplikacije za potporu oboljelima od bolesti štitnjače**

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 studenta:

SADRŽAJ

1. UVOD 1
2. PROGRAMSKA OKOLINA 2
2.1. Operacijski sustav Android 2
2.2. Android Studio 4
2.3. Programski jezik Java 4
2.4. Baza podataka Room 5
2.5. Testiranje programske podrške 6
2.5.1. Općenito o testiranju 6
2.5.2. Vrste testova pri testiranju programske podrške 7
2.6. Alati za testiranje mobilnih aplikacija i izazovi 8
2.6.1. Alat za testiranje Espresso 9
3. BOLESTI ŠTITNJAČE I PRIKAZ POSTOJEĆIH RJEŠENJA 10
3.1. Štitnjača 10
3.1.1. Osnovni pojmovi o bolestima štitnjače 10
3.1.2. Bolesti štitnjače 10
3.1.3. Tipovi bolesti 11
3.1.4. Simptomi oboljelih 12
3.1.5. Dijagnoza bolesti 13
3.1.6. Terapija oboljelih 14
3.2. Prikaz postojećih rješenja za potporu liječenju 15
4. MODEL APLIKACIJE 17
4.1. Prikaz cjelovitog sustava za potporu oboljelima 17
4.2. Prikaz informacije o bolesti i dodavanje podsjetnika 18
4.3. Upravljanje korisnicima 19
4.4. Ulazni parametri 19
4.5. Postupak analize odabira simptoma i unosa parametara 21
4.6. Način prikaza rezultata, stvaranja preporuka i upozorenja 22
4.7. Prikaz kompletnog rješenja potpore oboljelima 24
5. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE 28
5.1. Autentifikacija pomoću sustava Firebase 28
5.1.1. Registracija i prijava korisnika 28
5.1.2. Profil korisnika 32
5.2. Navigacije glavnim izbornikom pomoću ladice 35
5.3. Postavljanje baze podataka Room 36

5.3.1. Baza podataka aplikacije 37
5.3.2. Model baze podataka 37
5.3.3. Sučelje My Dao 38
5.4. Podsjetnici korisniku 39
5.4.1. Notifikacije podsjetnika 39
5.5. Programsko rješenje potpore oboljelim korisnicima 42
5.5.1. Unos parametara laboratorijskih testova 42
5.5.2. Implementacija algoritma na osnovi unosa korisnika 43
5.5.3. Prikaz rezultata grafički i generiranih preporuka pomoću algoritma 44
5.5.4. Spremanje prikazanih rezultata u bazu podataka 47
5.6. Programsko rješenje potpore potencijalno oboljelim korisnicima 48
5.6.1. Odabir simptoma korisnika 48
5.6.2. Implementacija algoritma na osnovu odabira korisnika 49
5.6.3. Prikaz rezultata grafički i generiranih preporuka pomoću algoritma 50
5.7. Grafički prikaz svih zaslona aplikacije 52
6. TESTIRANJE MOBILNE APLIKACIJE 53
6.1. Implementacija testnog alata 54
6.2. Pisanje testnih slučajeva 54
6.3. Provođenje testova 55
6.3.1. Snimač testova 55
6.3.2. Pisanje Espresso testova 56
6.3.3. Problemi i prepreke pri izvođenju testova 59
6.3.4. Unaprjeđenje programskog koda 60
7. ZAKLJUČAK 62
LITERATURA 63
SAŽETAK 68
ABSTRACT 69
ŽIVOTOPIS 70
PRILOZI 71

1. UVOD

Poremećaj u radu štitne žlijezde ljudi postaje sve raširenij i veći problem. Unatoč tome što se poremećaji i bolesti mogu vrlo dobro liječiti čestim praćenjem stanja preko rezultata laboratorijskih testova, mnogi ne prepoznaju simptome i ne pridodaju im veliku pažnju. Prisutni se simptomi najčešće pripisuju nekim drugim fizičkim i psihičkim stanjima. Obraćanjem pažnje na simptome i povremena posjeta liječniku bi zasigurno omogućila pravovremeno otkrivanje i liječenje bolesti.

Cilj diplomskog rada je izrada mobilne aplikacije za potporu oboljelima od bolesti štitnjače. Potpora će biti ostvarena preko dvije temeljne opcije aplikacije. Korisnik će preko parametara laboratorijskih testova moći pratiti vlastiti napredak usporedbom trenutnog i prijašnjeg mjerenja. Na temelju usporedbe bit će generirani rezultati i preporuke. Druga mogućnost je odabir grupiranih simptoma poveznim uz određeno fizičko ili psihičko stanje korisnika. Na temelju odabira izvršit će se usporedba sa simptomima bolesti te kao rezultat dati bolest s kojom se simptomi najviše podudaraju. Upravljanje korisnicima će biti ostvareno preko platforme u oblaku, dok će upravljanje podacima biti napravljeno preko baze podataka. Konačno, provest će se automatizirani testovi grafičkog sučelja od kojih se očekuje dodatno poboljšanje stabilnosti aplikacije i uputiti na pogreške u kodu. Od dodatnih mogućnosti korisnik će moći napraviti podsjetnik pomoću notifikacijskog kanala ili pregledati informacije o pojedinoj bolesti unutar aplikacije.

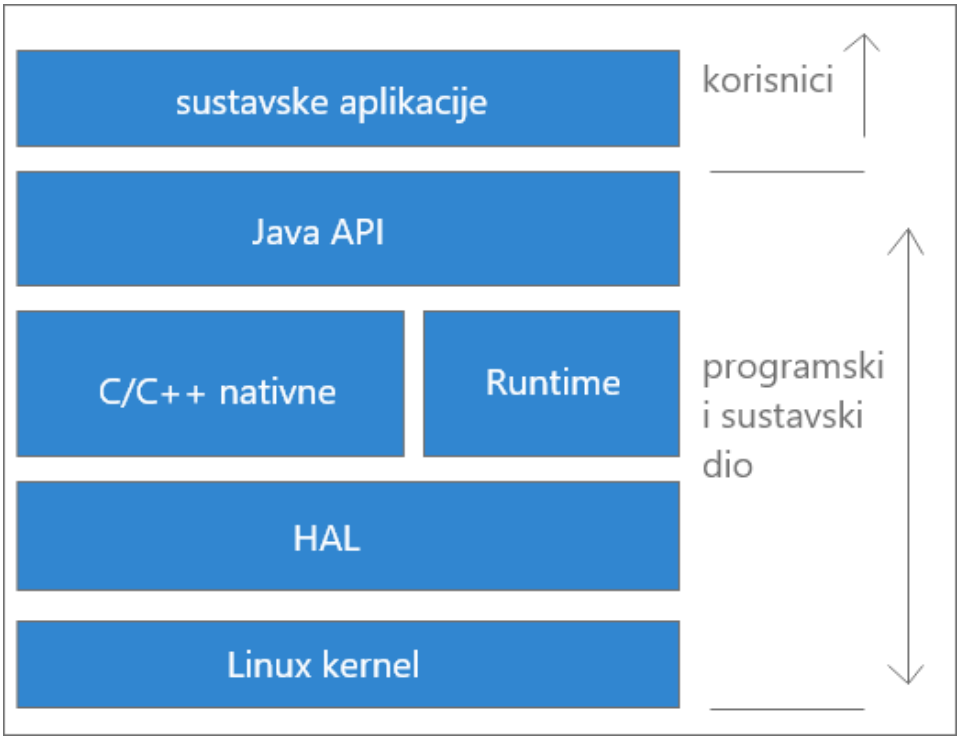
U poglavlju 2 bit će opisane tehnologije i alati koji će se koristiti pri izradi aplikacije. Teorijska osnova u poglavlju 3 pruža informacije koje će poslužiti kao osnova pri generiranju preporuka i rezultata, dok prikaz postojećih rješenja daje predodžbu kako su riješene slične problematike vezane uz temu ovog rada. Poglavlje 4 predstavlja logiku mobilne aplikacije, opisuje postupke i analize podataka koji se u njoj koriste. Poglavlje 5 detaljno opisuje i implementira logiku, postupke i analizu popraćenu programskim kodom. U poglavlju 6 se objašnjavaju postupci testiranja realizirane aplikacije te izvješće o testiranju nakon provedenog testiranja.

2. PROGRAMSKA OKOLINA

U ovom poglavlju opisat će se okolina korištena pri razvoju ove aplikacije u kojoj će se pisati programski kod, upravljati autentifikacijom korisnika, baza podataka te alat za testiranje aplikacije.

2.1. Operacijski sustav Android

Android [1] je operacijski sustav otvorenog koda kojeg je konstruirala tvrtka Android Inc. 2003. godine koju nakon dvije godine preuzima tvrtka Google koja je vlasnik i danas. Platforma je zasnovana na *kernelu* Linuxa. Namijenjen je uređajima sa zaslonom na dodir, kao što su pametni telefon, tablet. Sustav je modularan i prilagodljiv, stoga se sve više koristi u uređajima kao što su televizori, pametni satovi, a sve više su popularni i pametni zvučnici čija je glavna odlika ugrađenost Googleovog pomoćnika (engl. *Google Assistant*) kojim se vrlo lako pristupa glasovno i omogućuje čovjeku današnjice brz pristup informacijama, ali i nekih radnji u vlastitome dome: kontrola svijetla, zvuka ili nekog drugog uređaja u kući. Operacijski sustav je zasnovan na programskim jezicima C i C++, a pokreće ga Linux jezgra inačice 2.6. Arhitektura androida prikazana je na slici 2.1.



Sl. 2.1. Arhitektura Androida

Linux jezgra je temelj Android arhitekture. Sadrži sve *drivere* niže razine za razne komponente koje sam android može podržavati, budući da postoji veliki broj proizvođača programske podrške s različitim komponentama.

HAL (engl. *Hardver Abstraction Layer*) pruža sučelje između komponenti i programske podrške. On se sastoji od više modula biblioteka od kojih svaki pruža sučelje prema specifičnom dijelu sklopovlja. Kada aplikacijski okvir napravi poziv određenog API-ja (*engl. Application Programming Interface*) prema sklopovlju HAL odradi poziv i učita određenu biblioteku koja mu je potreba za komunikaciju s određenom komponentom.

ART (engl. *Android Runtime*) koji se koristi kod aplikacija s minimalnom inačicom Android 5.0 (engl. *Lollipop*) što odgovara API razini 21 koja će se koristiti i u ovom radu kao minimalna inačica Androida. Prije ARTA na inačicama nižim od 21 koristio se Dalvik. Sloj ART je dizajniran s ciljem pokretanja aplikacija u okruženju u kojem su resursi kao što su baterija, procesorske aktivnosti i memorija. Svaka aplikacija ima svoj proces i pokreće se kao zasebni ART virtualni uređaj tako što pokreće DEX (engl. *Dalvik Executable*) datoteke. Glavne značajke ovog sloja su kompiliranje razine višeg programskog jezike AOT (engl. *Ahead Of Time*) i JIT (engl. *Just In Time*), optimiziranije upravljanje nepotrebnim datotekama, bolja podrška za otklanjanje grešaka. U otklanjanje grešaka (engl. debugging) se ubraja detaljna dijagnostika iznimaka i rušenja aplikacije (engl. *crash report*).

C/C++ biblioteke su biblioteke koje se mogu koristiti od strane korisnika i pisane su osnovnim jezikom kojim je pisan i sam operacijski pristup. Pristupom tim bibliotekama korisnik dobiva pristup kameri, *OpenGL ES* koji su služi za manipuliranje 2D i 3D objektima u aplikaciji. Za pristup osnovnim funkcijama platforme koristi se NDK (engl. *Native Development Kit*) koji omogućuju pristup osnovnim funkcijama, odnosno fizičkim komponentama kao što je senzor blizine ili kamera.

Java aplikacijski okvir (engl. *Java API Framework*) je zbirka svih API-ja napisanog u programskom jeziku Javi koji daje razvijateljima programske podrške pristup cjelokupnom repozitoriju značajki sustava. Ovakva kolekcija može se podijeliti u nekoliko dijelova: upravitelj resursima, upravitelj notifikacijama, upravitelj aktivnostima, osiguravatelj sadržaja koji se koristi u drugim aplikacijama sustava. Neke od takvih aplikacija su: e-mail klijent, web preglednik, kalendar, mape. Takva kolekcija se može upotpuniti s aplikacijama iz Google Play [1] trgovine aplikacijama koji je nastao 2008. godine pod nazivom *Android Application Market*.

2.2. Android Studio

Android studio [3] je integrirano razvojno okruženje (engl. *Integrated Development Environment*) korišteno pri razvoju Android aplikacija na osnovi IntelliJ IDEA platforme, moćnog i pametnog alata za uređivanje koda i razvijateljskih alata. Značajke kojima se odlikuje Android Studio su: prilagodljiv sustav zasnovan na Gradleu[1], brz i značajkama bogat emulator, okruženje u kojem se mogu razvijati aplikacije za sve uređaje, brzo pokretanje aplikacije pri malim promjenama, predlošci koda, integracija s Git uslugom, podrška za testiranje i okvirima za testiranje, C++ i NDK podrška, ugrađena Google platforma Firebase koja se koristi za mnoge stvari u oblaku, a neke od značajki su: analitika, mrežne baze podataka, slanje izvještaja o pogrešci u radu aplikaciju, strojno učenje u oblaku, spremanje podataka o korisnicima i mnoge druge opcije.

Sve datoteke su vidljive pod datotekom Gradle skripte i sadrži sljedeće mape: manifest, java (Java datoteke izvornog koda, uključujući JUnit test kod) i resurse (sadrži sve podatke nevezane uz kod kao što su datoteke dizajna, niza znakova, ikonica, animacija itd.) .

Manifest sadrži *AndroidManifest.xml* datoteku koja opisuje ključne informacije o aplikaciji kao i alatima koji se u njoj koriste, dopuštenja koja se koriste kako bi korisnik aplikacije mogao pristupiti određenim resursima i opcijama uređaja. Ovdje su smješteni podaci o sklopovlju i programu koji određuju koji će korisnici moći koristiti aplikaciju ovisno o tim informacijama.

Pri razvoju ove aplikacije korištena je inačica 3.1.2 na sustavu Windows 10 s 8 GB radne memorije s time da je 2 GB ostavljeno na korištenje ugrađenom emulatoru. Minimalna inačica sustava je 5.0 što izravno odgovara API razini 21.

2.3. Programski jezik Java

Java [5] je programski jezik kojeg je razvila tvrtka Sun Microsystems, a sada je vlasnik Oracle. Glavna odlika kojom su se vodili pri stvaranju Jave je WORA (engl. *Write Once, Run Anywhere*). Sintaksa pisanja je vrlo jednostavna, a neka nekih stvari koje zadaju velike probleme kod jezika kao što je C++, a to su eksplicitni pokazivači, preopterećenje operatora i brisanje nereferenciranih objekata. U Javi postoji automatsko brisanje nepotrebnih stvari (engl. *automatic garbage collection*). Objektno je orijentiran, a time i ima sve karakteristike takvog jezika. Interpretacijski jezik što je isto razlog zbog koje je nešto sporiji nego C/C++ kompilirani jezici, što znači da se direktno izvršava na virtualnom uređaju bez direktne pretvorbe izvornog

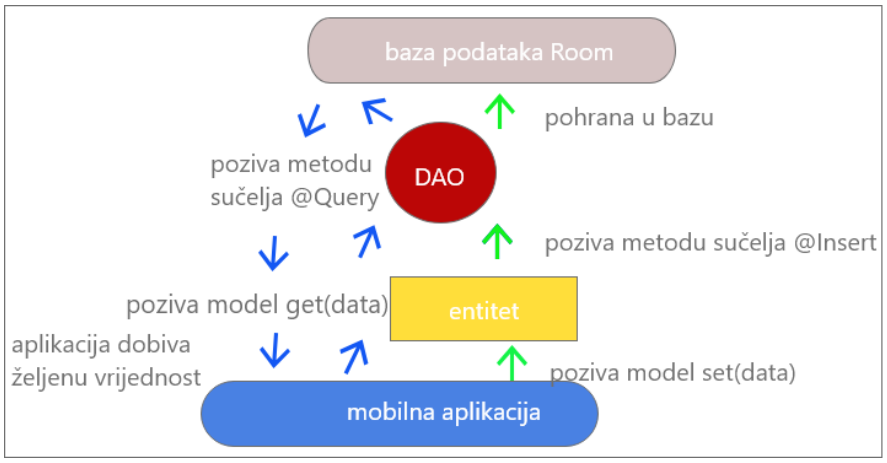
[1] Sustav automatizacije otvorenog koda koji pokreće i određuje i kombinira sve dijelove aplikacije u jedno pri pokretanju

koda u kod računala kao kod kompiliranih programa. Neke od jedinstvenih opcija Jave su: višenitno izvođenje programa, neovisna je o arhitekturi i operacijskom sustavu, sigurna zbog izvođenja unutar virtualnog uređaja koji pruža dodatni zaštitni sloj prema. Neovisnost o platformi je možda i najjača karakteristika Jave kao programskog jezika, pa tako i u ovom slučaju i na Androidu. Datoteka koja sadrži java kod pretvara se java *bytecode* koji može pokrenuti Java virtualni uređaj koji postoji na svakoj platformi. Na operacijskom sustavu Android kod iz Android Studija se kompilira u JVM *bytecode*, koji se zatim pretvara u DVM bycode koji se pretvara u ELF (engl. *Executable and Linkable Format*) datoteku koja se preko virtualnog uređaja pokreće kao zaseban proces na niti vidljivo u izvoru [4]. Najnovija inačica Jave je trenutno Java SE (engl. *Standard Edition*) 8 [4].

2.4. Baza podataka Room

Room je prema [6] baza podataka koju je napravio i održava ju i unaprjeđuje Google. Room pruža apstrakcijski sloj preko standardne SQLite baze podataka kako bi se omogućilo ugodno korištenje za krajnjeg korištenja bez ikakvih poteškoća u korisničkom sučelju i istovremeno zadržavajući sve osobine standardne SQLite baze podataka. Aplikacije koje rukuju sa podacima koji nemaju previše raznoliku strukturu mogu imati velike koristi u performansama ako zadržavaju te podatke lokalno, pa su takvi podaci su dostupni i kada aplikacija nema pristup internetu ili mobilnim podacima te je na takav način također vrlo lagano osvježiti i unaprijediti podatke preko poslužitelja koji će dohvatiti podatke periodički ili nekog drugog ključnog trenutka u trenutku kada je ponovno dostupna neka od internet konekcija, a on je definiran u dokumentaciji aplikacije.

Postoje tri osnovne komponente u Roomu prema [6] su: baza podataka, DAO (engl. *Data Access Objects*), entiteti. Baza podataka sadrži nositelja baze podataka i služi kao glavna točka pristupa za podatke i relacije koja aplikacija zahtjeva. Klasa notacije *@Database* mora biti apstraktna klasa koja nasljeđuje ugrađenu bazu podataka i osnovna je klasa za sve baze u Roomu kojoj se pristupa kroz nasljeđivanje. Entiteti predstavljaju tablicu unutar baze podataka. Načini pristupa podacima u bazi podataka su upit u bazu (engl. *query*), ažuriraj, obriši uz varijacije na među njima, kao i međusobna povezanost. Opisana struktura baze podataka vidljiva je na slici 2.2. U ovom projektu koristit će se Room zbog navedenih značajki.



Sl. 2.2. Prikaz strukture Rooma

2.5. Testiranje programske podrške

U ovom poglavlju objasnit će se teorija testiranja od osnovnih pojmova, preko izazova, pa sve do stvarne primjene pri razvoju aplikacije i izbacivanja aplikacije na tržište.

2.5.1. Općenito o testiranju

Testiranje programske podrške je proces provjere već implementiranog rješenja ili provjeravanje svih obilježja aplikacije koji su definirani dokumentacijom aplikacije. Kada dođe do razlike između očekivanog i stvarnog rezultata onda je na osobi zaduženoj za testiranje dužnost da osobi koja je pisala kod prijavi pogrešku u ponašanju aplikacije. Sve je veće korištenje pametnih telefona, tableta, pametnih satova. Mobilne aplikacije koriste se u razne svrhe kao što je aplikacija za vježbanje i primjerice aplikacija neke banke. Proces testiranja je vrlo zahtjevan zbog velikog broja različitih uređaja, rezolucija zaslona, kapaciteta memorije i drugih. Testeri imaju 48% zasluge u stabilnosti aplikacije koje izađu u trgovini aplikacijama. Prema podacima 48% korisnika [7] neće koristiti aplikaciju ako se aplikacija sruši ili naiđe na neki veći problem, nego će potražiti slično rješenje negdje drugdje.

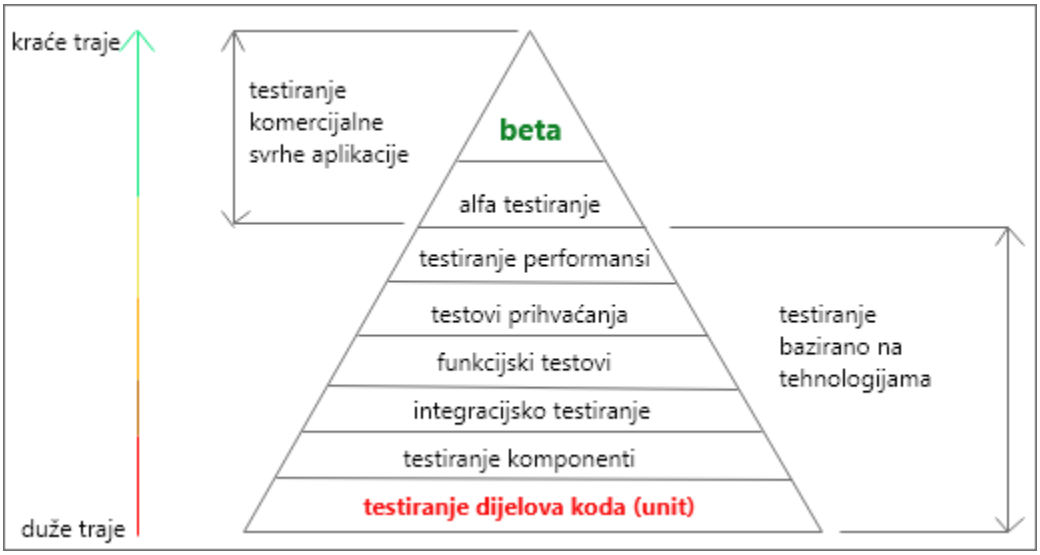
Osnovni tipovi testiranja programske podrške su testovi pod nazivom crna (engl. *black box*) i bijela kutija (engl. *white box*). Prvi tip testiranja provode osobe kojima nije poznat izvorni kod ili struktura toga koda koji testiraju, dok kod testa bijele kutije osobi koja testira vrlo je dobro poznata struktura koda aplikacije koja se testira. Kod prvog testiranja nije potrebno programersko znanje. Drugu vrstu provodi programer kojem je vrlo dobro poznata struktura koda, ovdje se provodi *testovi dijelova koda, integracijski, testovi performansi* i drugi te vrlo je bitno posjedovati programerske vještine, pozivajući se na [8] i [9].

Načini testiranja su ručno (engl. *manual*) ili automatsko testiranje (engl. *automated*). Manualno testiranje [10] je testiranje koje obavlja osoba prolazeći kroz aplikaciju odnosno testirajući osnovne ili nove značajke napravljene od razvojnog tima. Tester provodi testne slučajeve iz repozitorija testnih slučajeva aplikacije koje je prethodno vrlo detaljno razradio.

Automatsko testiranje [11] se zasniva na pokretanju skripte odnosno dokumenta koje se odvija pomoću računala koja je prethodno napisana prema strukturi razvijene aplikacije ili jednog njezinog dijela. Ovakvi testovi jako variraju u složenosti. Pouzdaniji su od ručnog testiranja i ključ su ka kontinuiranoj integraciji gdje se svakom novom dodanom značajkom ažurira skripta koja služi i kao dokumentacija.

2.5.2. Vrste testova pri testiranju programske podrške

U ovom poglavlju redom će se opisati vrste testova pozivajući se na [9], [12] i [13] koji se mogu koristiti pri testiranju programske podrške, a prikazani su slikom 2.3.



Sl. 2.3. Prikaz navedenih testova

Testovi dijelova koda (engl. *unit*) su testovi niske razine, zbog toga što su u razini s izvornim kodom. Sastoji se od testiranja pojedinih metoda, funkcija klasa, komponenti ili modula korištenih u aplikaciji. *Integracijski* testovi provjeravaju događa li se između komponenti međusobno povezanih u aplikaciji neki konflikt, koji može izbiti između inačice biblioteke korištene prije i novije inačice operacijskoj sustava. Isto tako može se provjeriti interakcija s bazom podataka i mikro usluga koji koriste neki podatak iz baze. *Funkcijski* testovi svoj fokus usmjeravaju na poslovne zahtjeve aplikacije. U takvim testovima traži se samo konačni ishod i ne gleda se ostatak aplikacije to jest kako se dolazi do toga rezultata, nego je bitno da se očekivani rezultat dobije. *Cjelokupni* (engl. *end-to-end*) testovi predstavljaju korištenje

aplikacije poput korisnika prolazeći kroz razne usluge aplikacije koje su dostupne samome korisniku, kao prijava, registracija, primanje notifikacije, naplata usluge unutar aplikacije i druge. Ovakvi testovi su korisni, ali su vrlo skupi pogotovo ako su automatizirani. Testovi prihvaćanja (engl. *acceptance*) provode se kada se želi utvrditi konačni poslovni ciljevi, ali sada uz konačni ishod da sve ostalo radi i aplikacija mora zadovoljavati pravila Google Play-a. *Test performansi* provjeravaju stabilnost aplikacije kada se aplikacija nađe pod velikim opterećenjem. *Testovi dima* (engl. *smoke*)[2] provjeravaju osnovne funkcionalnosti aplikacije, ovakav test ne traje dugo. Provjeravaju se osnovne funkcionalnosti i ako je sve u redu tim osiguranja kvalitete (engl. *Quality Assurance*) daje odobrenje da se aplikacija može isporučiti trgovini aplikacijama. Opisan postupak vidljiv je na slici 2.5.



Sl. 2.5. Životni ciklus aplikacije

2.6. Alati za testiranje mobilnih aplikacija i izazovi

Prilikom testiranja mobilnih aplikacija osiguranje kvalitete se susreće sa izazovima opisanim u [7], [14] te [15]. Prvi izazov je *fragmentacija uređaja*, a odnosi zastupljenost najnovije inačice operacijskog sustava, tj. koliki je omjer korištenja starih i novih tehnologija i sustava na mobilnim uređajima. Ovo je česta pojava kod Androida uređaja budući da u prosjeku najnoviju inačicu operacijskog sustava koristi nešto više od 8%. Najučestalije vrste mobilnih aplikacija

[2] Provjerava se prije izlaženja aplikacije na trgovinu i gleda se rade li sve osnovne funkcionalnosti

su [14]: nativne, hibridne. Nadalje, različitost *konekcije* podrazumijeva performanse aplikacije ovisno o korištenju određene mobilne mreže u mrežnom ili izvanmrežnom načinu. Odabir *odgovarajućeg alata* pri testiranju jedna je od ključnih odluka koja se mora donijeti. Potrebno je naći onaj koji najbolje odgovara platformi aplikacije, ima podršku za više platformi, te mnoge druge mogućnosti. Neki od alata prilikom automatskog testiranja mobilnih aplikacija su: Appium, Calabash, Frank, Money Talk, Robotium, Espresso [7]. Budući da će ovaj rad biti razvijen kao nativna Android aplikacija odlučeno je da će se koristiti onaj alat koji najbolje radi s Android nativnom aplikacijom. Izbor se sveo na Robotium i Espresso, ali na kraju je izabran alat Espresso.

Espresso [16] testni alat namijenjen Android nativnim aplikacijama u odnosu na Robotium ima bolju sinkronizaciju jer obično testovi se ne izvrše do kraja kod ažuriranja korisničkog sučelja zbog potrebe za samostalnim ubacivanjem čekanja testova pri izvšenju neke naredbe. Espresso to rješava tako da sinkronizira niti što čini izvođenje testova puno sigurnije i stabilnije.

2.6.1. Alat za testiranje Espresso

Prilikom testiranja koristit će se Espresso alat koji je razvio Google. Namijenjen je testovima grafičkog sučelja aplikacije (engl. *user interface*) koji su kratki, pouzdani i vrlo precizno rade. Alat je namijenjen i osmišljen kako bi i programeri vrlo brzo i lagano mogli provoditi UI testove koji su dio automatskog testiranja i bitan dio programske podrške mobilne aplikacije.

Espresso [17] ima vrlo snažne sinkronizacijske sposobnosti. Kada se u testu pozove metoda prilikom odabiranja elementa na pogled (engl. *onView*) te neka od UI aktivnosti nije trenutno dostupna on će čekati dok se uvjeti sinkronizacije ne ispune. Sadrži razne pakete koji u sebi sadržavaju određene akcije manipulacije sučeljem aplikacije. *Espresso-core* sadrži naredbe vezane za umetanje, podudaranje elemenata, naredbe nad elementima, zatim *web paket* sadrži podršku za prikaz web sadržaja unutar aplikacije, *contrib paket* posjeduje naredbe vezane uz biranje datuma, prilagodljivih lista (engl. RecyclerView*),* ladice (engl. *drawer),* brojanje resursa koji su u stanju pripravnosti.

3. BOLESTI ŠTITNJAČE I PRIKAZ POSTOJEĆIH RJEŠENJA

U ovom poglavlju saznat će informacije o štitnjači općenito, bolestima, simptomima, terapiji, dijagnozi te nekim prijašnjim programskim rješenjima vezanim uz štitnjaču.

3.1. Štitnjača

U ovom poglavlju govorit će se o teoriji štitnjače koja će uvelike poslužiti u daljnjem nastavku razvoja ove aplikacije.

3.1.1. Osnovni pojmovi o bolestima štitnjače

Štitnjača je žlijezda u ljudskom tijelu smještena na vratu ispod grkljana. Oblik štitnjače podsjeća na leptira. Bitna je za mnoge funkcije ljudskog organizma [18]. Štitnjača [19] je odgovorna za regulaciju bazalnog metabolizma (koliko brzo konzumirane kalorije se mogu pretvoriti u energiju potrebnu čovjeku za odvijanje osnovnih životnih funkcija primarno i ostalih aktivnosti sekundarno), regulaciju topline tijela, centra za disanje, rad srca, regulaciju potrošnje energije, apsorpcije vitamina i korištenje istih, zdravlje živčanog sustava, kao i rast i sazrijevanje svih tjelesnih tkiva u ljudskom organizmu.

Hormoni štitnjače [20] odgovorni za bazalni metabolizam su tiroksin T4 i trijodtironin T3. Također postoje i hormoni koji nisu u potpunosti pod utjecajem štitnjače, a to su TSH (engl. *thyroid stimulating hormone*), koji je pod nadzorom hipofize i TRH (engl. *thyrotropin releasing hormone*) pod nadzorom hipotalamusa.

Zdravlje štitnjače može se najbolje očuvati unošenjem određenih količina kemijskog elementa joda koji je u sastavu namirnica. Obična kuhinjska sol koju danas koristimo je jodirana. Jod se nalazi u namirnicama kao što su jaja, jogurt, ribe, sir i mnoge druge. Do problema sa štitnjačom dolazi kada njezin rad više nije pravilan.

3.1.2. Bolesti štitnjače

Bolesti štitnjače su među najčešćim medicinskih slučajevima, a posebno su izražene kod žena. Simptomi su rijetki, izostaju ili ih je vrlo teško primijetiti. Pošto simptomi nisu uvelike pouzdani, doktori i specijalisti se oslanjaju više na laboratorijsko testiranje pri dijagnozi poremećaja. U 50-im godinama prošlog stoljeća je postojao samo jedan test [21] koji bi otkrio bolesti štitnjače, a to je bilo procjena joda u proteinu povezana s koncentracijom ukupnog seruma tiroksina (T4) koji je pokazivao vrlo lošu procjenu bolesti. Od tada, broj se testova naglo povećao, a testovi su se poboljšali. Prema istraživanju [22] tijekom života 1 od 20 stanovnika Amerike oboljet će od poremećaja štitnjače s time da su sedam puta više obolijevanju sklonije

žene. Rak štitnjače je najučestaliji oblik raka i zauzima dio od 1.5% do 2.1% svi rakova dijagnosticiranih u svijetu.

Proces sinteze hormona štitnjače počinje u predjelu mozga pod imenom hipotalamus koji proizvodi i ispušta hormon TRH koji putuje kroz venski pleksus do malog dijela u mozgu zvan hipofiza. Kao odgovor na TRH hormon hipofiza ispušta drugi hormon TSH u krv. Hormon putuje do štitne žlijezde koja stimulirana hormonom hipofize te proizvodi hormone T3 i T4. Najveću biološku aktivnost ima T3. Većina tiroksina ispuštena u krv se pretvori u aktivni hormon T3 koji je odgovoran za aktivnost metabolizma stanica kako je opisano u [20].

Regulacija sinteze hormona odvija detektiranjem koliko je hormona u krvi. Lučenje prevelikog broja hormona hipofiza uravnoteži proizvodnjom hormona TRH i TSH. Smanjenjem proizvodnje hormona njihova razina će se dovesti na optimalnu razinu. Nepravilnosti u ovom postupku mogu dovesti do poremećaja odnosno bolesti štitne žlijezde. Bolesti možemo klasificirati i navesti prema literaturi [23], [24] te [25]:

- hipertireoza
- hipotireoza (Gravesova bolest)
- tireoiditis (Hashimoto, post porođajni tireoiditis)
- gušavost
- kvržice na štitnjači rak štitnjače

U [27] spominje se i sindrom ne-tireoidalnih bolesti koji je usko vezan s izgladnjivanjem ili se pak primjećuje po nenormalnim rezultatima testa štitnjače koji postoje kod bolesnika s akutnim ili kroničnim bolestima. No ova bolest se neće razmatrati u ovome radu.

3.1.3. Tipovi bolesti

Bolesti štitnjače kod žena utječu na menstrualni ciklus te ga čini nepodnošljivim, neredovnim (može čak izostati i po par mjeseci). Otežava proces začeća zbog utjecaja na ovulaciju. Poteškoće može uzrokovati i tijekom trudnoće djelujući štetno na dijete i na majku. Ponekad se simptomi vezani za štitnjaču vrlo lako predvide kao simptomi menopauze. Nakon menopauze žena je sklonija oboljenju od hipotireoze [26]. Izglednije je da će oboljeti ljudi koji su imali problema sa štitnjačom u prošlosti, operaciju, kemoterapiju koja je utjecala na štitnu žlijezdu ili pak boluju od gušavosti, anemije ili dijabetesa tipa jedan. Navedeni simptomi razvrstat će se po pojedinim bolestima.

Hipotireoza [25] je bolest kada štitna žlijezda ne proizvodi dovoljno T3 i T4 hormona u krvotok. Najveći uzrok hipotireoze u Sjedinjenim Američkim Državama je Hashimoto bolest [26].

Uzrokovati ju može terapija kao što je terapija radioaktivnim jodom, kemoterapija određenog raka ili uklanjanje određenog dijela štitnjače ili cijele štitnjače.

Hipertireoza je bolest kada štitnjača proizvodi previše T3 i T4 hormona u krvotok. Najveći uzrok hipertireoze je Gravesova bolest [26]. Autoimuna bolest kod koje imunološki sustav proizvodi antitijela koja napadaju štitnjaču i kao produkt ona proizvodi previše hormona, a također uzrokuje naticanje i povećanje štitnjače. Od ove autoimune bolesti najčešće obolijevaju žene u dobi između 30 i 60 godina. Vjerojatnost da će žena oboljeti prema [20] i [25] je veća ako ju karakteriziraju sljedeće osobine: oboljeli član obitelji, autoimuna bolest, utjecaj stresa, pušenje.

Tireoiditis [18] je upala štitnjače koju uzrokuju antitijela koja proizvodi imunološki sustav. Uzroci upale štitnjače su autoimune bolesti, reumatični artritis, genetika, virusna ili bakterijska infekcija te određeni tipovi lijekova. Hashimoto [26] bolest javlja se kao oblik upale štitnjače, a pojavljuje se najčešće u 40-im i 60-im godinama života, no mogu oboljeti i mlađe žene. Rizik se povećava ako osoba već ima neku od autoimunih bolesti: reumatični artritis, celijakija, dijabetes tipa jedan, lupus, anemija. Postporođajni tireoiditis prema [20] i [26] se pojavljuje u dva tipa. Prvi tip pojavljuje se unutar prvog i četvrtog mjeseca i žena može imati simptome hipertireoze sljedeća dva mjeseca zbog oštećenja štitnjače pa se hormoni izlijevaju u krvotok. Drugi tip pojavljuje se između četvrtog i osmog mjeseca i može potrajati od pola godine do godinu, a žena ima simptome hipotireoza. Rizik je veći ako oboljela ima povijest ove bolesti u obitelji, već se dogodilo u prijašnjoj trudnoći, ima kronični virusni hepatitis prema.

Gušavost je neobično povećana štitnjača. Uzroci ove bolesti leže u drugim bolestima štitnjače: Hashimoto, Gravesova, kvržice na štitnjači, tireoiditis, rak štitnjače.

Rak štitnjače nastaje kada stanice raka nastanu od stanica štitne žlijezde. Izglednije da će od raka oboljeti žene koje su: između 25 i 65 godina [26], bile ozračene pri snimanju glave ili vrata u mladosti, preboljele gušavost ili se netko u obitelji prije obolio od bolesti raka.

3.1.4. Simptomi oboljelih

Simptom za pojedinu bolest su ujedno i slični i različiti i relativno je teško otkriti pomoću simptoma o kojoj je bolesti uvijek riječ. Simptomi će se navesti iza svake bolesti.

Hipotireoza ima simptome prema [20], [24] i [28]: osjećaj zimogroznosti, neredovita stolica, malaksalost, povećanje tjelesne težine, smanjen apetit, bol u zglobovima ili mišićima, osjećaj

tuge, depresija, umor, blijeda, suha koža, suha i tanka kosa, spori otkucaji srca, manji intenzitet znojenja, glatka koža lica, promukli glas te obilno menstrualno krvarenje.

Simptomi kod Hipertireoze su: gubitak težine, čak i jedenjem iste količine ili veće količine hrane (većina, ali ne svi ljudi izgube težinu), veliki apetit, brzi i nepravilni otkucaji srca, osjećaj nervoze ili tjeskobe, osjećaj razdražljivosti, nesanica, trnci u rukama i prstima, povećan intenzitet znojenja, osjećaj vrućine, slabost mišića, proljev, bol u želucu, kraća i laganija menstrualna razdoblja od normalnog, zatim promjene u očima koje mogu uključivati izbočenje očiju, crvenilo ili iritaciju [20] i [24]. Gravesova bolest [26] posjeduje sljedeće simptome: izbočene oči, debljina i crvenilo kože, osobito u listovima i zglobovima stopala (samo u Gravesovoj bolesti), razdražljivost ili nervoza, umor i slabost mišića, osjetljivost na toplinu, problemi sa spavanjem, ruke se tresu, brzi i nepravilni otkucaji srca, proljev, gubitak težine bez posebnog režima prehrane i gušavost.

Gušavost se prepoznaje po povećanoj štitnjači, otežanom gutanju, otežanom disanju. Kvržice na štitnjači ima simptome vizualnih kvržica vidljivih na vratu na mjestu gdje se nalazi štitnjača ili čak može indicirati na rak štitnjače [22].

3.1.5. Dijagnoza bolesti

Ponekad je teško dijagnosticirati bolest samo na temelju simptoma pacijenta iz razloga što su slični simptomima koje imaju mnoge druge bolesti. Liječnik [29] će u slučaju sumnje na bolest štitnjače prvo pitati povijesti bolesti štitnjače u obitelji. Prvi korak je podizanje glave i pregledavanje postoji li možda izraslina na vratu.

Krve pretrage su obično druga razina testiranja na bolest. Takvi testovi sastoje se od testiranja razine TSH hormona u krvi pacijenta, kako bi se otkrilo proizvodi li štitnjača previše ili premalo hormona štitnjače. Test će se možda ponoviti, a nekim slučajevima provjerava se razina oba hormona u krvi. Prema [18] pri testovima se koriste: TSH, T4 i fT4, T3 i fT3, gdje slovo *f* ispred naziva označava broj slobodnih hormona, a normalan ukupni broj hormona. Provjerava se i razina sljedećih antitijela: TPO, TG (tireoglobulin) te TSI (antitijela TSH hormona). O spomenutim parametrima više će se reći u potpoglavlju 4.4.

Ispitivanje uzimanjem radioaktivnog joda [21] provodi se tako da pacijent proguta tekućinu ili kapsulu s malom dozom radioaktivnog joda koji se veže na štitnjaču. Visoka razina radioaktivnog joda pokazuje da je štitnjača preaktivna, odnosno mala razina pokazuje kako je štitnjača smanjene aktivnosti, nego što bi ona to trebala biti.

Scintigrafija prema [26] je slikanje štitnjače provodi se s istim radioaktivnim jodom u obliku kapsule ili tekućine. Posebnom kamerom pravi slika štitnjače na zaslonu računala. Postoje tri vrste takvih slika. „Vruće" kvržice prikazuju se na zaslonu nešto svjetlije od normalnih štitnih izraslina, budući da se veže više radioaktivnog joda na štitnjaču što označava veću aktivnost štitnjače. Manje od 1% posto takvih kvržica je kancerogeno. „Tople" kvržice vežu na štitnjaču normalnu količinu radioaktivnog joda i samo 5% do 8% je kancerogeno [30]. "Hladne" kvržice vežu na štitnjaču vrlo malo radioaktivnog joda i ne štitnjača ne proizvodi hormone, do 15% posto ovakvih kvržica je kancerogeno.

Ultrasonografija, ultrazvuk štitnjače [28] koristi zvučne valove kako bi stvorio sliku štitnjače na zaslonu računala. Ovakav test može pomoći liječniku vidjeti koji tip kvržice pacijent ima i koliko je velika. Potrebno je više pregleda ultrazvukom kako bi se procijenilo raste li kvržica ili se pak smanjuje. Može pomoći u pronalaženju raka štitnjače, ali ne i u njenom dijagnosticiranju [29].

Citopunkcija ili biopsija štitnjače otkriva nalaze li se u štitnjači zdrave stanice. Liječnik potom izvadi stanice i tekućinu iz štitnjače dovoljne za mikroskopski uzorak. Pacijent može bolovati od raka štitnjače [26] i [30] u slučaju da stanice nisu zdrave.

Test na antitijela mogu otkriti postojanje autoimune bolesti kao što je Hashimoto bolest štitnjače. Otkrivanje antitijela otkriva se preko pretraga krvi. Postojanje određenih vrsta i količine antitijela u krvotoku ne mora nužno značiti da čovjek ima autoimunu bolest, čak oko 10 % [26] ljudi u svom krvotoku ima antitijela i normalan rad štitne žlijezde.

3.1.6. Terapija oboljelih

Za svaku bolest postoji određena terapija kako bi se ublažili simptomi bolesti i loš utjecaj na tijelo oboljelih. Hipotireoza se liječi lijekovima [26] koje prepiše liječnik. Lijek djeluje tako da vraća štitnjaču u normalan rad, normalnu proizvodnju hormona i regulaciju hormona prema potrebi. Takvi lijekovi nerijetko se uzimaju cijeloga života.

Hipertireoza se liječi na više načina što ovisi o simptomima i uzroku hipertireoze. Lijekovima na dva načina. Gravesova bolest [24] i [26] liječi se lijekovima MMI (Methimazole) koji je štetan za bebu u trudnoći i PTU (Propylthiouracil). Ova dva lijeka sprječavaju štitnjaču da proizvodi previše hormona, a obično se daju pacijentima prije operacije ili nekog drugog tretmana. Ovi lijekovi dugoročno neće proizvesti nikakva oštećenja štitnjače. Beta blokeri blokiraju utjecaj hormona na ljudski organizam te mogu pomoći kratkoročno pri smanjivanju krvnog tlaka, dok se čeka na nekakvu drugu terapiju. Problem je što oni ne blokiraju broj

hormona koji je prevelik i utječe na tijelo i druge funkcije ljudskog tijela. Tretman radioaktivnim jodom ubija stanice štitnjače i kao takav može prouzročiti hipotireozu vrlo brzo nakon tretmana. Hipotireozu [25] i [28] može prouzročiti i operacija kojom se uklanja dio ili cijela štitnjača.

Tireoiditis [26] se može podijeliti na dvije autoimune bolesti: Hashimoto bolest i postporođajni tireoiditis. Hashimoto bolest se liječi uspješno lijekovima koji sadrže T4 hormon. Tijelu treba neko vrijeme da reagira na takve lijekove, pa je odlazak liječniku vrlo poželjan, dok se razina TSH hormona ne dovede u ravnotežu. Nakon toga posjeti liječniku mogu biti sve rjeđi. Postporođajni tireoiditis se liječi ovisno o fazi u kojoj se žena nalazi. U pravilu rad štitnjače se vraća u normalnu kroz 12 do 18 mjeseci, dok žene s povijesti o takvoj bolesti mogu razviti trajnu hipertireoza unutar 5 do 10 godina, pozivajući se na [23], [26] te [30] .

Gušavost se može liječiti lijekovima koji će vratiti štitnjaču u njeno normalno stanje ili čak operacijom u kojoj će liječnik morati odstraniti dio štitnjače.

Kvržice na štitnjači se liječe ovisno o vrsti kvržice. Prvi korak, ako kvržica nije kancerogena, je praćenje njezinog stanja, u toj fazi se odvijaju pregledi, krvne pretrage ili čak ultrazvuk štitnjače. Operacija nastupa kada je kvržica kancerogena ili je prevelika tako da otežava disanje ili gutanje. Tretman radioaktivnim jodom se koristi kada kvržice vežu na sebe previše takvog joda. Radioaktivni jod pomaže da se kvržice smanje i uzrokuje da štitnjača općenito proizvodi manje hormona prema [23], [25] i [26]. Naravno takva terapije može imati i neugodne posljedice, kao bubrenje tkiva koji su kao ostaci ostali u području vrata , suha usta, gubitak osjeta i okusa u ustima. Zanimljivo je kako žvakaća guma može znatno umanjiti zračenje prije nego radioaktivna tvar dođe u mjehur tako što kupe tvari na sebe te mnoge druge [22] i [25].

Rak štitnjače se primarno liječi operacijom [25], tako se odstrani dio koji je kancerogen ili što manji dio ako je to moguće. Najbolje se liječi ako se rak nije proširio i na vrijeme je otkriven. Nakon operacije liječnik će koristiti radioaktivni jod kako bi uništio sve stanice raka koje nisu uklonjene i osigurati da se ne prošire na ostatak tijela. Za svaku fazu raka postoji drugačiji tretman.

3.2. Prikaz postojećih rješenja za potporu liječenju

U ovom poglavlju bit će dane osnovne informacije o već postojećim projektima, načinima generiranja podataka pri otkrivanju ili dijagnosticiranju bolesti štitnjače

Predviđanje dijagnoze bolesti štitnjače pomoću umjetne inteligencije imunološkog sustava [31] prikazana je mogućnost određivanja tipa bolesti na osnovu tri testa koji se obično provodi ne bi li se otkrila bolest štitnjače, a to su: razina T3 u krvi, razina T4, te razina TSH hormona. Prvi korak pri pravljenu ovakvog sustava otkrivanja bolesti je naći dovoljnu i reprezentativnu količinu podataka koja je preuzeta s UCI repozitorija koji je jedan od najvećih repozitorija podataka koji se koriste pri strojnom učenju (engl. *machine learning*). U strojnom učenju je potrebno konstruirani model predviđanja na temelju danih podataka tako da se podaci mapiraju u grupe (engl. *cluster*) koje komuniciraju sa najbližim susjedom do centra grupe gdje se klasificiraju. Opisani način je postupak koji je jednak onom u imunološkom sustavu čovjeka. Metafore korištene u ovome modelu su: vezanje antigena, sazrijevanje afiniteta, pravljenje memorije u imunološkom sustavu, konkurencije resursa te proces odabira kloniranja. Ovim načinom postigla se točnost s više od 94%. Navedeno je programsko rješenje pomoglo pri realizaciji vlastitog programskog rješenja u pogledu odabira parametara štitnjače.

Segmentiranje kvržica štitnjače pomoću *K-Means* algoritma na mobilnim uređajima [32] je drugi primjer koji će se razmotriti. Smisao ove aplikacije je obrada slike i uloga mobilnog uređaja kao klijenta. Slijed je sljedeći: tehničar za snimanje štitnjače pomoću magnetske rezonancije šalje snimljenu sliku na poslužitelj. Korisnik uzme mobilni uređaj preko kojeg može dobiti originalnu sliku i odabrati opciju analiziraj. Slika se šalje na obradu na poslužitelj preko alata *MatLab* te se pomoću algoritma K-Means obradi prikazuje rezultat područja na kojima se nalaze kvržice kompletno zanemarujući sve ostale dijelove s naglaskom na dio gdje se ona nalazi. Procedura algoritma je ukratko postavljanje točaka u prostor oko objekata koji se skupljaju oko glavnih točaka ponovnom procjenom udaljenosti sve dok se svi objekti ne pridruže danim točkama. Proces dobivanja rezultata sastoji se iz predprocesiranja, algoritma, postprocesiranja, ZSI (indeks točnosti) te rezultata. U zaključku ZSI uspoređuje metode koje se koriste u ovome radu i druge metode istreniranog modela. Ako je vrijednost iznad 0.7 u rasponu od 0 do 1 onda je to dobro. Točnost ovog rada procijenjena je na 90% što je vrlo dobar rezultat.

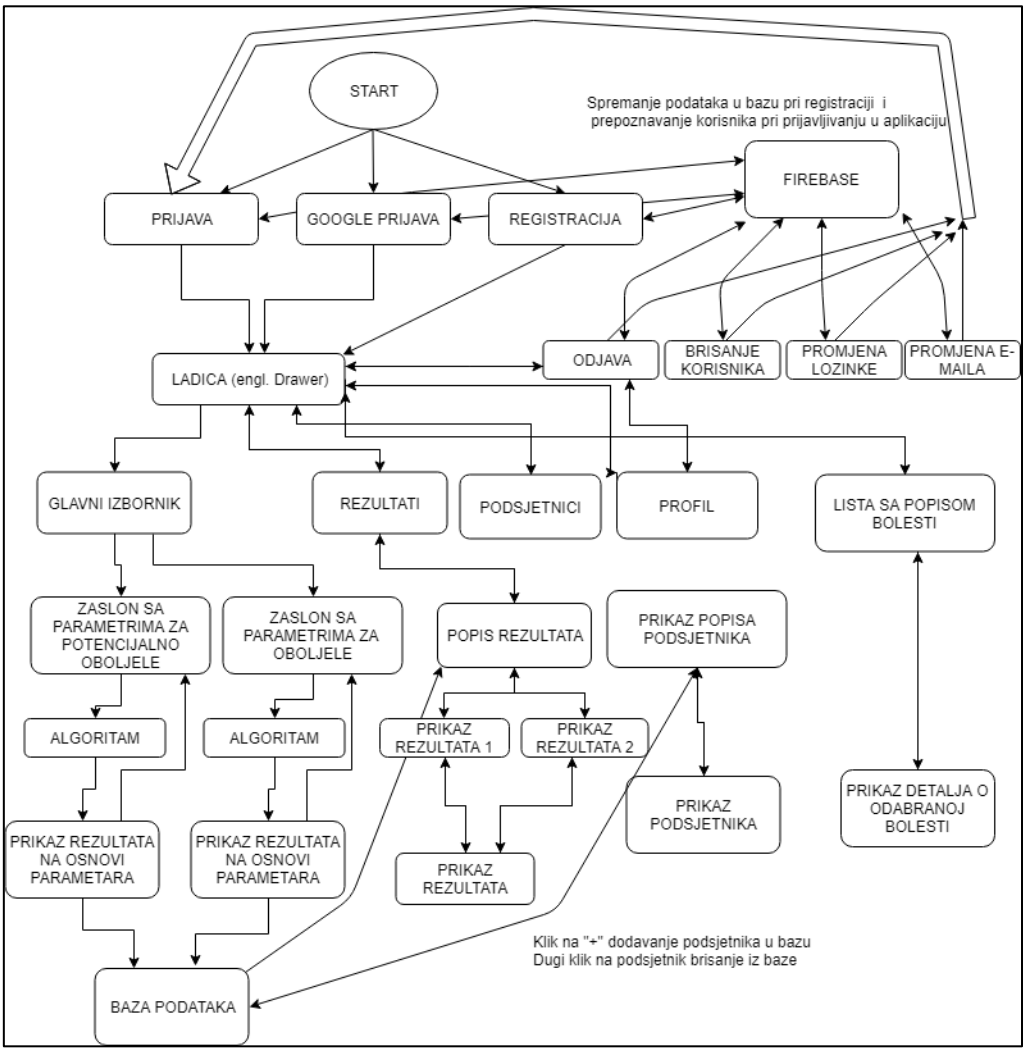
Postoji još radova koji se bave ovom problematikom otkrivanja bolesti štitnjače, ali i drugih bolesti. Svi oni se svode na umjetnu inteligenciju i strojno učenje ili njihova fuzija [33] sudeći prema radovima i godinama u kojima su objavljeni. Takav način pristupa je budućnost i dobro istrenirani modeli preko velike količine dostupnih podataka (engl. *Big data*), bilo da se radi o donošenju odluke na osnovu podatka ili filtriranja slika pomoću mnogih algoritama kojim se kao rezultat iz slike dobije točno određeni dio koji je zanimljiv koji je trebalo pronaći na temelju parametara.

4. MODEL APLIKACIJE

U ovom poglavlju na osnovu prethodnih poglavlja uspostavit će se konačni model aplikacije sa svim njezinim osobinama, preko informacija o bolestima, upravljanja korisnima te krajnjim prikazom i algoritmom generiranja rezultata i preporuka.

4.1. Prikaz cjelovitog sustava za potporu oboljelima

Na slici 4.1 dan je osnovni prikaz modela aplikacije s osnovnim tijekom i ponašanjem. U nastavku ovaj model će se objasniti u detalje segment po segment te će na kraju biti prikazan nadopunjeni model koji će ujedno i tvoriti onakav model koji će biti izveden i prikazan u programskom rješenju. Gledajući dijagram tijeka od gore prema dolje, može se vidjeti upravljanje korisnikom koji je potreban za pristup svim mogućnostima aplikacije, spremanje i rezultata analiza i podsjetnika odvija se u interakciji s bazom dok upravljanje korisnika preko platforme Firebase.



Sl. 4.1. Osnovni prikaz cjelovitog rješenja aplikacije

4.2. Prikaz informacije o bolesti i dodavanje podsjetnika

Informacije o bolesti štitnjače bit će pružene svim korisnicima aplikacije. Odabirom opcije iz izbornika ladice, otvorit će se lista s nazivima bolesti. Ovaj popis bolesti biti će dostupan čak i kada nema ništa u bazi podataka jer će ona biti ugrađena u aplikaciju preko polja niza znakova koji će preuzeti i resursa aplikacije u kojima se nalazi .xml datoteka s popisom svih značajki vezanih uz svaku bolest, a općenito tamo se nalaze svi nizovi znakova koji se koriste u aplikaciji. Odabirom elementa liste dolazimo u detaljan prikaz svake bolesti, a jednostavnom navigacijom kroz aplikaciju odabirom gumba za nazad može se vrlo lako prebacivati iz detaljnoj prikaza jedne bolesti u detaljan prikaz druge bolesti. Informacije tih parametara preuzete su iz poglavlja 3.1 te će pravo ti podaci pružit će informacije korisniku o pojedinoj bolesti, bilo da se želi informirani, podsjetiti ili nešto novo naučiti o bolestima i samoj štitnjači općenito.

Zaslonu gdje se nalaze podsjetnici može se pristupiti preko ladice pritiskom na ikonu s tri vodoravne linije. Ondje je omogućeno dodavanje podsjetnika pritiskom na gumb dodavanja. U tome zaslonu se dodaje ime, tekst te datum i vrijeme kada korisnik želi da ga se podsjeti na određenu radnju koju treba napraviti. Slikom 4.2 dana je potencijalna struktura u bazi podataka koja će se na sličan način implementirati i za druge stvari, poput spremanja rezultata i pristupanje istim. Dugim pritiskom na element liste već dodanih podsjetnika može se izvršiti akcija brisanja preko dijaloga koji se pojavi i koji korisnika upozorava kako će podsjetnik biti izbrisan ako se pritisne na gumb s potvrdnim odgovorom.

```
@Entity(tableName = "podsjetnici")
public class reminder {

    @PrimaryKey
    @ColumnInfo(name = "podsjetnik_id")
    private String id;

    @ColumnInfo(name = "ime")
    private String name;

    @ColumnInfo(name = "tekst")
    private String text;

    @ColumnInfo(name = "vrijeme")
    private String time;

    @ColumnInfo(name = "datum")
    private String date;
}
```

Sl. 4.2. Potencijalna struktura podsjetnika u bazi podataka

4.3. Upravljanje korisnicima

U aplikaciji postoji par opcija kako se prijaviti u aplikaciju. Postoje tri opcije: nova registracija korisnika, prijava, prijava putem Google računa. Sve ove tri opcije rade preko Google-ove platforme Firebase koja je već ranije spomenuta u podnaslovu Android Studio. Prateći upute u Android Studiju implementacija ne bi trebala biti problem. U nju se spremaju podaci o korisniku pri registraciji koji su kasnije dostupni u aplikaciji i lako ih je izvući, a svi podaci su dostupni za pregled preko lijepog sučelja. Kao što je već navedeno osim ove vrste baze podataka pri prijavi i registriranju koristi se i Room kao baza podataka, no koja je korist i kako njih dvije zapravo funkcioniraju zajedno će biti navedeno u tekstu koji slijedi.

Firebase se koristi za registraciju, login i podatke koji su vezani uz konekciju na Internet kao što je mijenjanje slike profila korisnika, isto tako ova platforma služi za povlačenje podataka specifičnih za određenog korisnika prilikom odjave jednog korisnika i prijave drugog korisnika. U ovom slučaju je to slika profila koja može biti različita za svakoga ukoliko ju korisnik promijeni unutar aplikacije i spremi. U trenutku kada je novi korisnik prijavljen automatski se događa sinkronizacija s podacima i oni se skidaju kako bi bili vidljivi unutar aplikacije, a ta sinkronizacija se može napraviti i dodatno na neke druge načine kao što je dodatni gumb za osvježavanje podataka ili jednostavno čišćenjem aplikacije iz memorije ili stavljanjem u pozadinu i ponovnim pokretanjem. Room baza podataka služi za spremanje lokalnih podataka kao što su podsjetnici, rezultati laboratorijskih mjerenja ili otkrivanje potencijalnih bolesti pomoću određenih kategorija u koju su klasificirani simptomi.

4.4. Ulazni parametri

Glavni dio aplikacije u kojem registrirani korisnik kroz dvije opcije glavnog izbornika dobiva informacije o mogućem oboljenju ili napretku, odnosno pogoršanju ovisno o odabranoj opciji. Odabir simptoma razvrstanih u kategorije je opcija rezervirana za korisnike koji sumnjaju na bolest štitnjače po promjenama vlastitog tijela, odnosno registriranim simptomima. Nakon odabira opcije dolazi se na zaslon za četiri kategorije parametara tako da se u svakom parametru nalazi određena skupina simptoma u obliku padajućeg izbornika. Simptomi su prikazani tablicom 4.1. Simptomi za pojedinu skupinu parametara preuzeti su iz potpoglavlja 3.1.5. Simptomi oboljelih, koji su zatim po vlastitoj logici uz razmatranje [34] i [36] grupirani po smislenim parametrima kako bi algoritam koji će se koristiti, a o kojem će biti riječi u sljedećim poglavljima, imao što lakši zadatak za podudaranje s jednom od ranije spomenutih bolesti.

Tab. 4.1. Prikaz parametara i pojedinih simptoma po kategoriji

Parametar	Simptomi
Područje glave	Blijeda koža, Tanka kosa, Promukli glas, Crvenilo očiju, Iritacija očiju, Izbočene oči, Povećanje štitnjače, Otežano disanje i gutanje, Kvržice na vratu
Promjene tijela	Povećanje težine, Usporen rad srca, Manji intenzitet znojenja, Povećano menstrualno krvarenje, Gubitak težine, Brzi i nepravilni rad srca, Proljev, Smanjeno menstrualno krvarenje, Povećani intenzitet znojenja, Bol u želucu
Promjene u udovima	Bol u mišićima i zglobovima, Slabost mišića, Deblja koža oko listova i zglobova, Ruke se tresu
Promjene ponašanja	Zimogroznost, Malaksalost, Tuga, Depresija, Umor, Veliki apetit, Nervoza i tjeskoba, Razdražljivost. Nesanica, Osjetljivost na toplinu, Osjećaj vrućine
Druge bolesti	Autoimuna, Reumatični artritis, Virusna ili bakterijska infekcija, Anemija, Celijakija, Dijabetes Tip 1, Druge bolesti štitnjače, Gušavost

Druga opcija glavnog izbornika je praćenje napretka već oboljelih korisnika unosom rezultata testiranja, a takav rezultat se odnosi na rezultate koji se odnose na hormone i antitijela prema [34] i [38], barem u ovom slučaju. U tablici 4.2 dani su osnovni pojmovi vezani uz parametre koje će korisnik trebati unijeti kako bi se oni obradili te na kraju dali pravovaljani rezultat. Razine hormona i antitijela preuzete su iz [40] u kombinaciji sa [38].

Tab. 4.2. Objašnjenja osnovnih parametara vezanih uz rezultate testiranja

Tip	Objašnjenje	Specifičnosti
TSH	Provjera razine TSH iz testova krvi	Odgovoran za lučenje T3 i T4 hormona, ako razina nije normalna potrebno je napraviti dodatni test (T3, T4), pa čak i kada je normalna
FT4	Provjera slobodnih T4 hormona u krvi	Visoka ili niska razina ne mora značiti poremećaj, kod trudnica je T4 veći, a bolesnika koji se liječe kortikosteroidima manji, takvi poremećaji uzrokuju da se protein iz krvi veže za T4 hormone, koji služe kao pričuva), preporuča se napraviti i T3 test [34]
FT3	Provjera razine slobodnih T3 u krvi	Najčešće zadnji test koji se provodi u slijedu testova
TPO	Napadaju žlijezdu kao posljedica autoimunog poremećaja, provjera razine TPO antitijela	Vremenom obično uzrokuje uništenje tkiva štitnjače, povećane razine uzrokuju najčešće Hashimoto bolest, no i u normalnim okvirima također može uzrokovati spomenutu bolest
TSI	Test mjeri razinu antitijela u tijelu	Antitijela koja uzrokuju izlučivanje povećane razine hormona, uzrokujući hipertireozu, odnosno Gravesovu bolest
TG	Ovaj test mjeri razinu antitijela u krvotoku	Antitijela koja uništavaju protein tiroglobin koji je znak postoji li aktivnost štitne žlijezde, može se naći kod pacijenata koji potencijalno boluju od raka štitnjače ili ostalih autoimunih bolesti

Kako bi se zapravo odredio napredak vrlo je važno definirati podatke za pojedine hormone štitnjače te antitijela koja na nju utječu određivanjem granice za razine određenog hormona ili

antitijela koji ulaze u normalne razine, koji u dozvoljene i one ispod razine dozvoljene. Ovakva spomenuta klasifikacija podataka prema [38] i [40] dana je tablicom 4.3 i predstavlja temelj druge opcije glavnog izbornika i orijentacija oboljelima koji žele pratit svoj napredak i saznati sljedeći korak koji je potrebno napraviti u toj promjeni, bez obzira pogoršava se situacija ili poboljšava u odnosu na prijašnju. Korisnik će moći unijeti razine hormona i antitijela na temelju rezultata provedenih laboratorijskih testova.

Tab. 4.3. Klasifikacija podataka

Razina hormona/antitijela	Naziv hormona/antitijela	Iznos rezultata testa
Normalna	TSH (mU/L)	0.4 – 6.0
Ispod dozvoljene		< 0.4
Iznad dozvoljene		> 6.0
Normalna	FT4 (pmol/L)	9.0 – 25.0
Ispod dozvoljene		< 9.0
Iznad dozvoljene		> 25.0
Normalna	FT3 (pmol/L)	3.5 – 7.5
Ispod dozvoljene		< 3.5
Iznad dozvoljene		> 7.5
Normalna	TPO (IU/ml)	0 – 35
Iznad dozvoljene		> 35
Normalna	TSI (IU/ml)	0 – 1.3
Iznad dozvoljene		> 1.3
Normalna	FG (IU/ml)	0 – 4.0
Iznad dozvoljene		> 4.0

4.5. Postupak analize odabira simptoma i unosa parametara

Analiza podataka odvijat će se posebno za svaku grupu korisnika ovisno odabire li korisnik simptome ili unosi parametre testova. Način pronalaženja postotka podudaranja s bolesti ili razinom hormona i antitijela s spomenutim rasponima ovdje će biti detaljno razjašnjen.

Odabiranjem simptoma stvara se polje niza znakova. Zatim, postoji sedam polja bolesti s isto tako nizom znakova simptoma. Rezultat će se izvršiti pomoću algoritma uspoređivanja polja. Polje odabira znakova uspoređuje se sa svakom bolesti posebno. Za svaki simptom s kojim se podudara povećat će brojač za jedan. U konačnici postotak se dobiva dijeljenjem vrijednosti brojača s ukupnim brojem simptoma u polju simptoma bolesti te množenjem broja sa stotinu. Rezultat se sprema u varijablu. Postupak se ponavlja za sve bolesti. U polju vrijednosti vrši se algoritam sortiranja koji nalazi najveći postotak u polju koji je potrebno prema mjestu u polju povezati s određenom bolesti kako bi se mogla prikazati. Uvjetnim grananjem odlučuje se treba li se korisnik obratit liječniku na temelju postotka podudaranja ili to ipak nije potrebno. Sve varijable se spremaju u polje brojeva koji se ugrađuju u graf rezultata. Imena bolesti i vrijednosti

spremani su istim redoslijedom pa se podudaraju u grafičkom prikazu. Dodatna filtriranja oko preporuka odvijat će se pomoću *if* petlje postavljanjem raznih uvjeta

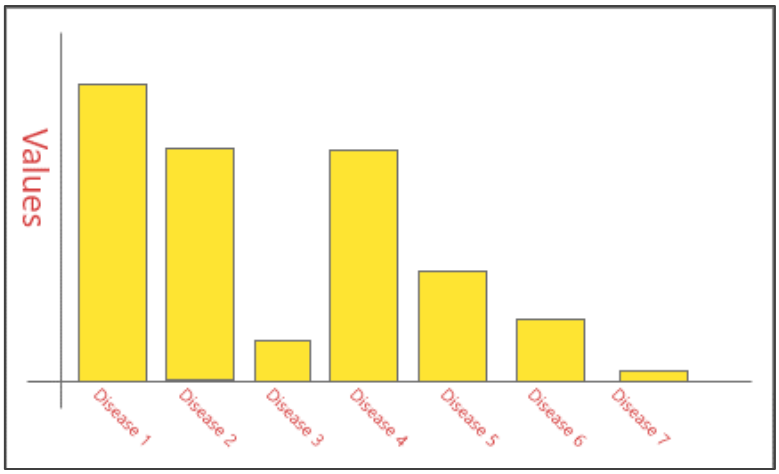
Uneseni parametri od strane korisnika se spremaju u polje brojeva koji se šalju i ugrađuju u graf. Prema definiranim konstantama razina parametara za normalnu, povećanu i smanjenu razinu količine hormona i parametara izvršit će se algoritam odlučivanja tako da će se rezultat prikazivat ovisno nalazi li se unesena vrijednost ispod dozvoljene ili iznad dozvoljene i u tom kontekstu prikazivati preporuke. Uvjetnim grananjem u parovima dva parametra hormona ili antitijela odlučivat će se boluje li korisnik od pojedine bolesti li ne. Važno je naglasiti kako će u grafu o kojem će biti više riječi kasnije, uvijek pokazivati uvijek zadnju vrijednost i trenutno unesenu vrijednost. Prvi puta bit će prikazana samo jedna skupina vrijednosti.

4.6. Način prikaza rezultata, stvaranja preporuka i upozorenja

Nakon što je algoritam obradio podatke te rezultate je potrebno prikazati. Za prvu grupu registriranih korisnika koji sumnjaju na bolest prikaz rezultata će imati sljedeći raspored: bolest i postotak, preporuke i prikaz podataka grafom. Bolest se prikazuje na temelju najvećeg postotka podudaranja sa simptomima određene bolesti čiji je postupak dan u prijašnjem poglavlju. Generiranje preporuka odvija se na temelju visine također postotka u posljednjem primjeru, dok graf prikazuje postotak podudaranja s ostalim potencijalnim bolestima. Postupak odlučivanja, koja će se preporuka prikazati usko je vezan uz postotak podudaranja bolesti sa simptomima, dan je tablicom 4.4 prema [34] i [35] te kombiniranjem podataka iz potpoglavlja 3.1.4. Prikaz podataka grafom napravljen je pomoću biblioteke za Android uređaje koja je otvorenog koda [37], a osnovni prototip grafa je prikazan slikom 4.3.

Tab. 4.4. Postupak odlučivanja prikaza bolesti za potencijalno oboljele korisnike

Razina	Preporuka
0-35%	Nema potrebe za brigom, probajte se više odmarati, ako se simptomi pogoršavaju ili se ponavljaju učestalo svakako potražite savjet liječnika
36-70%	Većina simptoma usmjeravaju ka poremećaju rada štitnjače, pomno pratite simptome te se savjetujte sa svojim liječnikom ili specijalistom
71-100%	Velika je vjerojatnost da postoji problem u normalnom radu štitnjače i njenom utjecaju na tijelo, odmah dogovorite test krvi kako bi se vidjele određene razine hormona koji su odgovorni za normalan rad štitnjače



Sl. 4.3. Prikaz rezultata grafom prve skupine korisnika

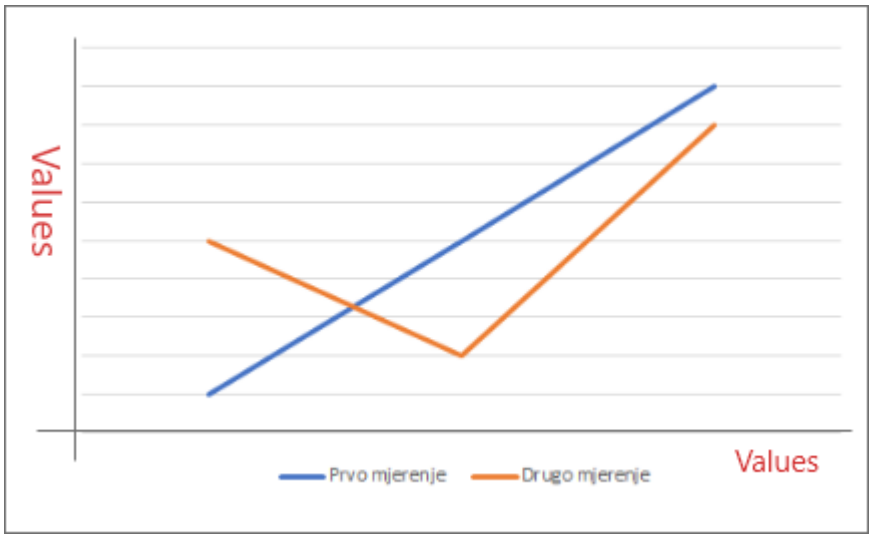
Prikaz rezultata kod već oboljelih korisnika je vrlo sličan. Nekoliko je razlika. Prva kategorija je prikaz napretka u odnosu na prethodno mjerenje, proveden test. Ako je korisniku to prvi podatak bit će prikazan tekst s porukom koja će ga obavijestiti kako je to prvo mjerenje te se napredak trenutno ne može vidjeti. Kada postoji barem jedno mjerenje aplikacija će prikazati postotak odstupanja od prijašnje vrijednosti postotkom. Preporuke koje su generirane prema literaturi [34] i [36] prikazati će se na zaslonu rezultata prema pravilima tablice 4.5 i 4.6. U zadnjoj kategoriji, grafu, će se najbolje vidjeti razlika između dva mjerenja, na način da će korisnik vidjeti pomak vrijednosti prema dolje što će značiti napredak, dok pad vrijednosti ispod prijašnje vrijednosti će značiti pogoršanje, za bolji prikaz i shvaćanje korisniku će na raspolaganju biti linijski graf koji se može vidjeti na slici 4.4.

Tab. 4.5. Postupak odlučivanja prikaza bolesti za potencijalno oboljele korisnike na temelju rezultata hormona

Postotak	TSH	FT4	FT3
Normalna	Sve je u redu, bilo bi dobro provesti i FT4 test	Sve je u redu, možete provesti i FT3 test, no nije nužno [31]	Sve je u redu, vaša štitnjača radi normalno ako su prethodno testovi TSH i FT4 bili normalni
Ispod dozvoljene	Prema razini hormona bolujete od hipotireoze, svakako provedite i FT4 test	Prema razini hormona bolujete od hipotireoze, zbog vlastite sigurnosti provedite i FT3 test	Prema razini hormona bolujete od hipotireoze, ako su već provedeni TSH i FT4
Iznad dozvoljene	Prema razini hormona bolujete od hipertireoze, savjetuje se provođenje FT4	Prema razini hormona bolujete od hipertireoze, zbog vlastite sigurnosti provedite i FT3 test	Prema razini hormona bolujete od hipotireoze, ako su prethodno već provedeni TSH i FT4

Tab. 4.6. Postupak odlučivanja prikaza bolesti za potencijalno oboljele korisnike na temelju rezultata antitijela

Postotak	TPO	TSI	TG
Normalna	Razina antitijela je u normalnom rasponu, ali to ne isključuje neku od autoimunih bolesti, najčešće Hashimoto bolest	Razina TSI je u normalnom rasponu, postoji mogućnost da bolujete od Gravesove bolesti, iako je sve u normalnom rasponu	Sve je u redu, vaša štitnjača radi normalno ako su prethodno testovi TSH i FT4 bili normalni TG antitijela su u normalnom rasponu, s ovom kategorijom antitijela valja biti uvijek na oprezu i pratiti njihov raspon što češće budući da su rasponi najviše zabilježeni kod ljudi s rakom štitne žlijezde
Iznad dozvoljene	Razina antitijela je povećana što upozorava na upalu žlijezde odnosno da je autoimuna bolest napredovala	Postoji povećana razina TSI antitijela, što ukazuje u velikoj mjeri na Gravesovu bolest, koja je uzrokovala hipertireozu svojim djelovanjem	Povećana je razina TG antitijela, što može upućivati na mnoge bolesti kao što su rak (dio ili cijela štitna žlijezda je odstranjena), Gravesova, a najveća je mogućnost za Hashimoto bolesti

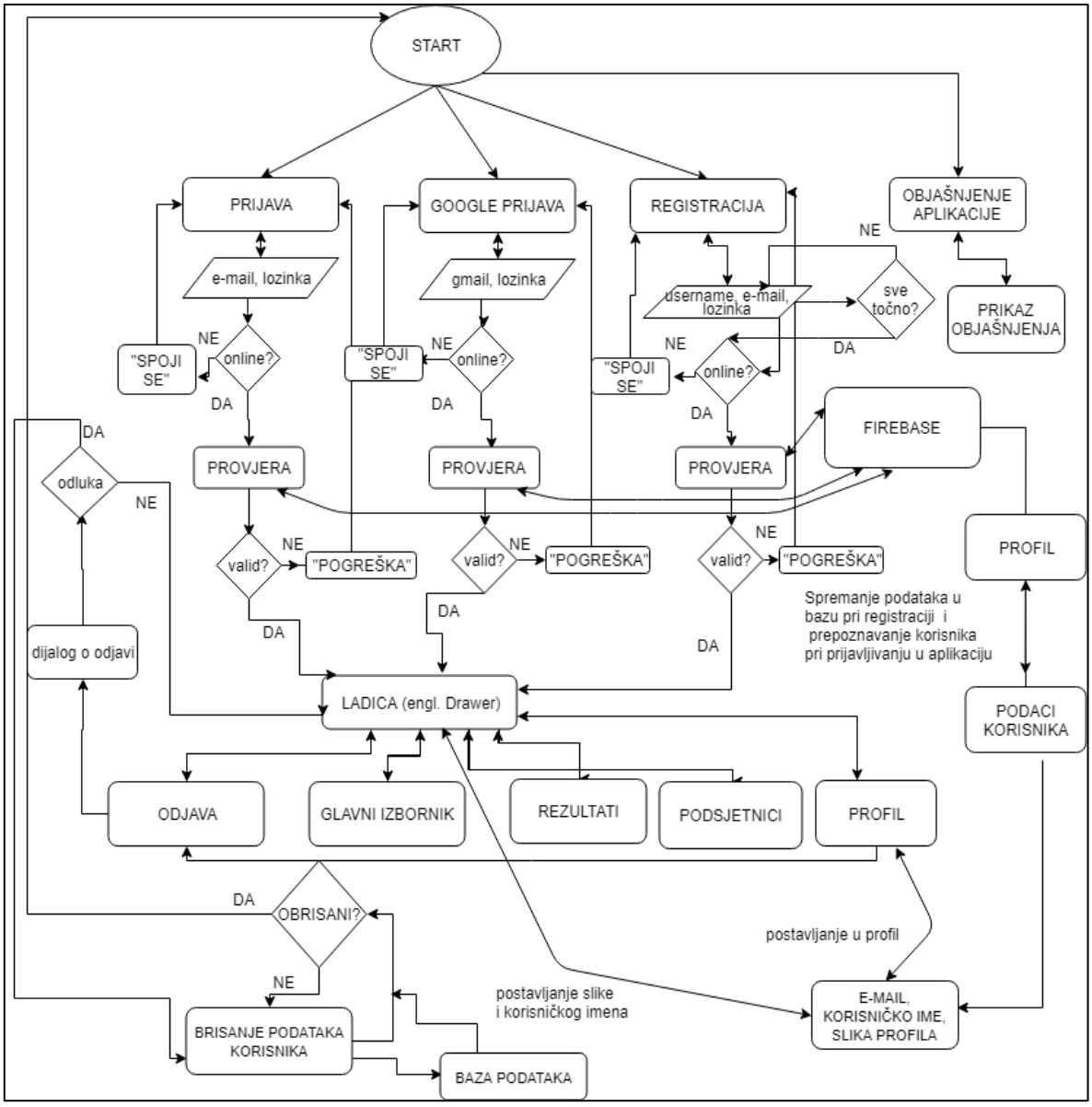


Sl. 4.4. Prikaz rezultata grafom druge skupine korisnika

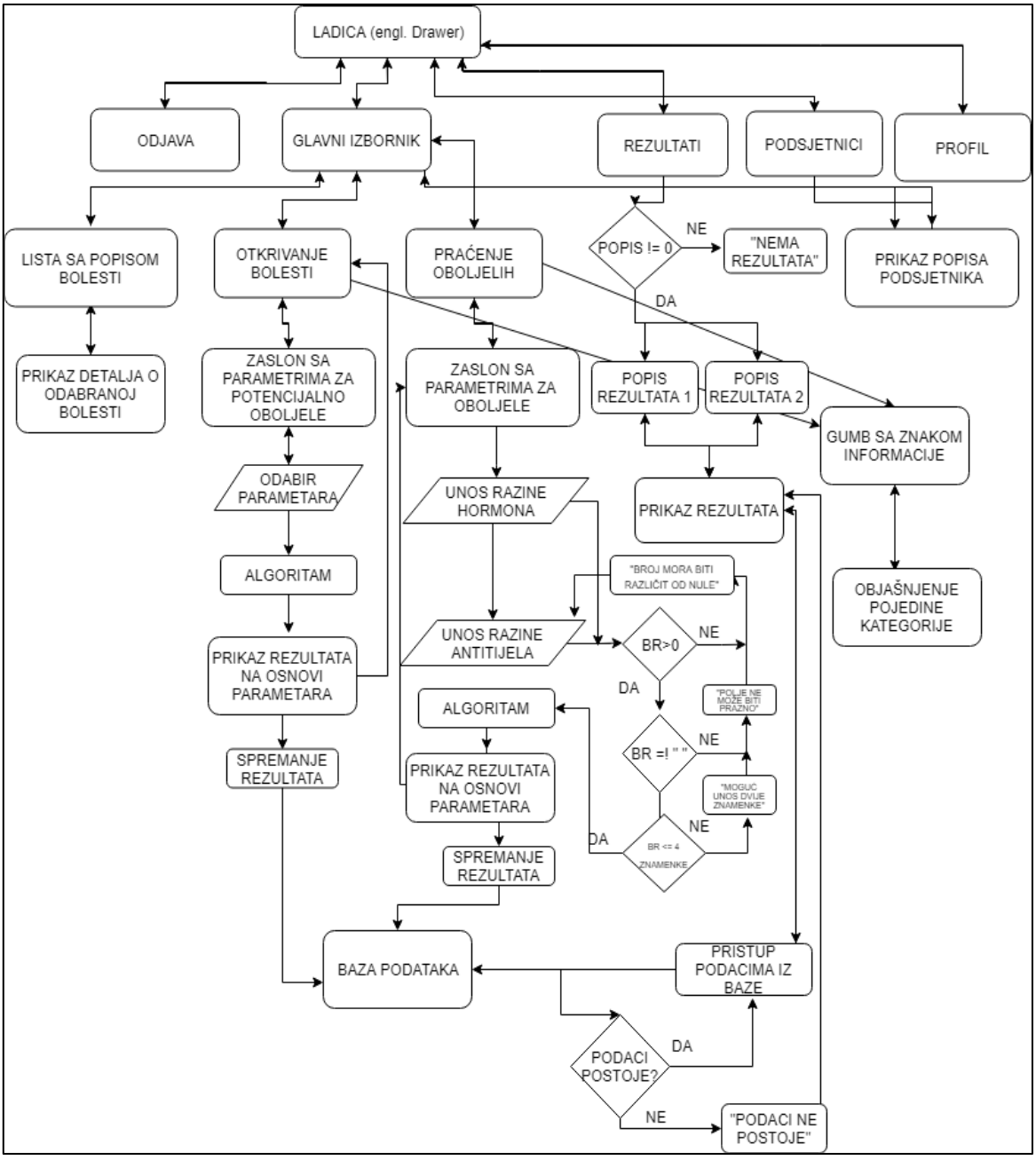
4.7. Prikaz kompletnog rješenja potpore oboljelima

Sustav kreiran u poglavlju 4.1. Prikaz cjelovitog sustava će se u ovom poglavlju podijeliti u nekoliko segmenata te će svaki biti prikazan odgovarajućom slikom povezanom s detaljiziranim sustavom aplikacije netom prije početka programskog rješenja kako bi se dobio uvid u navigaciju, ali i sve moguće opcija unutar aplikacije. Slikom 4.5. prikazan je dio vezan uz prijavu i registriranje i općenito upravljanje korisnikom. Prikazan je pristup korisnika aplikacije preko unosa korisničkog imena, lozinke koje prolaze provjeru preko platforme Firebase i ulazak korisnika u aplikaciju koja dobiva osobine korisnika prikazujući njegovu sliku i korisničke

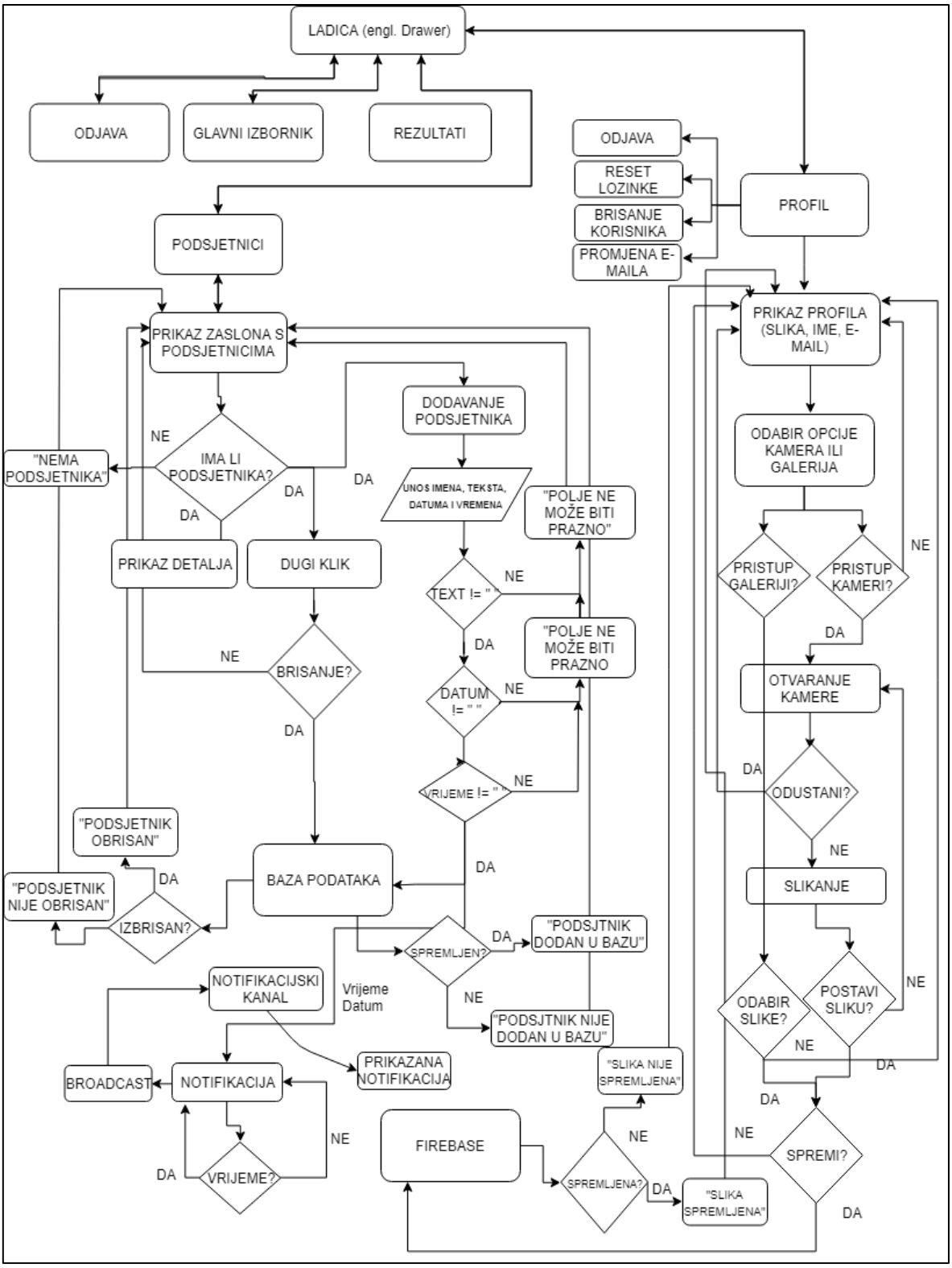
podatke. Dio oko liste bolesti i dvije kategorije korisnika oboljelih i oni koji sumnjaju na to je dan slikom 4.6. Ona u detalje prikazuje tijek unos parametara korisnika preko provjere i spremanja u bazu, kao i pristup podacima iz baze, ako oni postoje. Na kraju će biti prikazan slikom 4.7. dijagram tijeka vezan uz podsjetnike te informacije o korisniku odnosnu profilu korisnika. Tijek spremanja podsjetnika i biranja vremena i datuma za koje korisnik želi biti obaviješten prikazan je detaljno, kao i proces ažuriranja slike korisnika preko platforme Firebase.



Sl. 4.5. Prikaz dijagrama tijeka svi opcija korisnika pri prijavi, registraciji, odjavi, promjeni lozinke, e-mail i brisanje korisnika



Sl. 4.6. Prikaz dijagrama tijeka vezanog uz oboljele i potencijalno oboljele korisnike



Sl. 4.7. Prikaz dijagrama tijeka spremanja podsjetnika i obavijesti korisnika te ažuriranje slike profila korisnika

5. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE

Prikaz cjelokupnog razrađenog modela u obliku programskog koda s usputnim objašnjenjem specifičnih dijelova kao što je manipuliranje korisnicima, analiza unosa i odabira i njihov prikaz na zaslonu rezultata.

5.1. Autentifikacija pomoću sustava Firebase

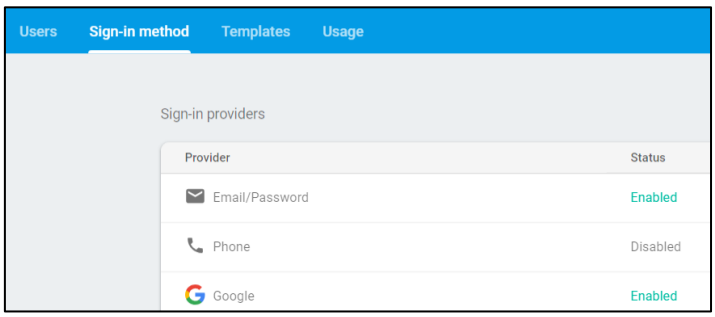
Pomoću platforme Firebase o kojoj je bilo riječi ranije u teorijskoj podlozi, točnije poglavlju 2.2. Pristupom web stranici poslužitelja i kreiranju računa za ovu aplikaciju bilo je potrebno osigurati ključ *SHA1* koji se generira tako što se otvori Gradle i iz izbornika se izabere opcija potpisivanje izvještaja (engl. *signingReport*) opcija pomoću koje aplikacija generira ključ koji se kopira u sustav Firebase. Na taj način su aplikacija i poslužitelj povezani. Kako bi aplikacija mogla koristiti puni potencijal i usluge platforme potrebno je unutar Gradle-a skripti implementirati ovisnosti (engl. *dependecies*) koje su dane slikom 5.1, a preuzete iz izvora [3] pod istoimenim pojmom Firebase.

```
implementation 'com.google.firebase:firebase-core:16.0.1'
implementation 'com.google.firebase:firebase-auth:16.0.1'
implementation 'com.google.android.gms:play-services-auth:15.0.1'
implementation 'com.google.firebase:firebase-storage:16.0.1'
implementation 'com.google.firebase:firebase-database:16.0.1'
```

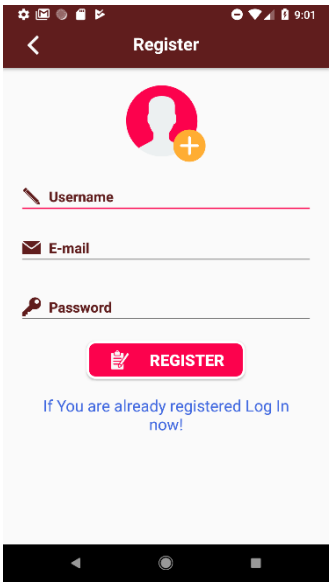
Sl. 5.1. Prikaz ovisnosti potrebnih za platformu Firebase

5.1.1. Registracija i prijava korisnika

Kada je sve postavljeno, potrebno je registrirati korisnika. Na Firebase je potrebno dopustiti registraciju određenim korisnicima. Korisnici kojima je dozvoljeno korištenje aplikacije pokazano je slikom 5.2. U aplikaciji su napravljene forme u kojima korisnik popunjava tri stvari. To su redom korisničko ime, e-mail adresa te na kraju lozinka koji uvijek moraju biti popunjeni na što se korisnika upozoriti provjera ako nije ispunjen jedan od uvjeta (slika 5.3).



Sl. 5.2. Dopuštanje registracije određenim korisnicima



Sl. 5.3. Zaslon registracije korisnika

Ovisno o uspješnosti pokazuje se poruka i preusmjerava korisnika dalje u aplikaciju na glavni zaslon. Registracija je korisnika vidljiva na slici 5.4 te ovisno o ishodu na temelju metode koja osluškuje (engl. *task complete listener*). Po uspješnoj registraciji korisnik je vidljiv u sustavu i njime se može manipulirati akcijama kao što su brisanje, blokiranje i uređivanje. U istom trenutku je napravljen i model korisnika u sustavu i bit će spomenut u sljedećem potpoglavlju.

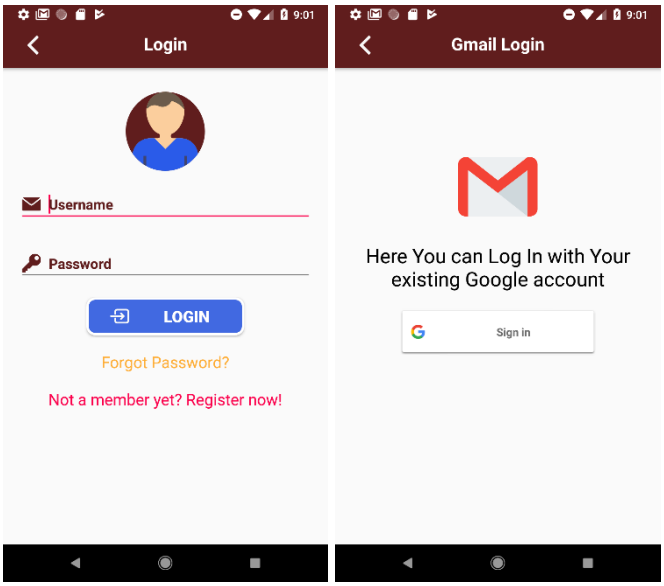
```
progressBar.setVisibility(View.VISIBLE);
    mAuth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(RegisterActivity.this, new
OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    progressBar.setVisibility(View.GONE);
                    if (!task.isSuccessful()) {
                        Toasty.error(RegisterActivity.this, "Authentication
failed" + task.getException(),
                                Toast.LENGTH_SHORT).show();

                    } else {
                        Toasty.success(RegisterActivity.this, "User is
Registered", Toast.LENGTH_SHORT).show();
                        saveUserInfomation();
                        startActivity(new Intent(RegisterActivity.this,
MainMenuActivity.class));
                        finish();
                    }
                }
            });
}
```

Sl. 5.4. Dodavanje novog korisnika s ispunjenim podacima

Nakon što je korisnik uspješno kreiran potrebno je moći iskoristiti te podatke za svaku buduću prijavu. Na zaslonu za prijavu nalaze se standardne stvari (Slika 5.5) kao što je e-mail adresa i lozinka ne bi li korisnik produžio na glavni zaslon što se vidi u kodu na slici 5.6. Postoji i opcija

resetiranje lozinke i na ovom zaslonu, ali nešto više o tome u sljedećem poglavlju. Važno je naglasiti kako aplikacija pamti postoji li već prijavljen korisnik pri svakom pokretanju korisniku i pamti sesiju, tako da nema potrebe za prijavljivanjem korisnika iznova svaki puta (Slika 5.7). U slučaju da je korisnik prijavljen aplikacija će samo produžiti na glavni zaslon.



Sl. 5.5. Prijava korisnika (lijevo) i korisnika Gmail-a (desno)

```
progressBar.setVisibility(View.VISIBLE);
    //authenticate user
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(LoginActivity.this, new
OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    progressBar.setVisibility(View.GONE);

                    if (!task.isSuccessful()) {
                        Toasty.error(LoginActivity.this, "Authentication
failed",
                                Toast.LENGTH_SHORT).show();
                    } else {
                        Toasty.success(LoginActivity.this, "User is Logged
in", Toast.LENGTH_SHORT).show();
                        Intent intent = new Intent(LoginActivity.this,
MainMenuActivity.class);
                        startActivity(intent);
                        finish();
                    }
                }
            });
}
```

Sl. 5.6. Prijava korisnika s postojećim podacima

```
@Override
public void onStart() {
    super.onStart();
    if (mAuth.getCurrentUser() != null) {
        startActivity(new Intent(RegisterActivity.this,
                MainMenuActivity.class));
        finish();
    }
}
```

Sl. 5.7. Kod koji osluškuje postoji li sesija korisnika

Kako je navedeno dodatno postoji opcija prijave korisnika koji već ima korisnički račun kreiran preko Google-a. U ovoj implementaciji postoji samo prijava korisnika, a ona je napravljena kao na slici 5.8. Pozovemo metodu također postojeću u Firebase-u tako da se definira i pozove funkcija (*GoogleAuthProvider*) koja omogućuje će pri prijavi provjeriti postoji li korisnik već u Google-ovoj bazi korisnika, ostatak se odvija kao i za standardnu već opisanu prijavu.

```
private void firebaseAuthWithGoogle(GoogleSignInAccount account) {
    AuthCredential credential =
GoogleAuthProvider.getCredential(account.getIdToken(), null);
    mAuth.signInWithCredential(credential)
            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        Log.d(TAG, "signInWithCredential:success");
                        spinnerProgress.setVisibility(View.GONE);
                        Toasty.success(GmailLoginActivity.this, "Google Sign
In success", Toast.LENGTH_SHORT).show();
                        Intent intent = new Intent(GmailLoginActivity.this,
MainMenuActivity.class);
                        startActivity(intent);

                    } else {
                        Log.w(TAG, "signInWithCredential:failure",
task.getException());
                        spinnerProgress.setVisibility(View.GONE);

                        Toasty.error(GmailLoginActivity.this, "Authentication
failed",
                                Toast.LENGTH_SHORT).show();
                    }

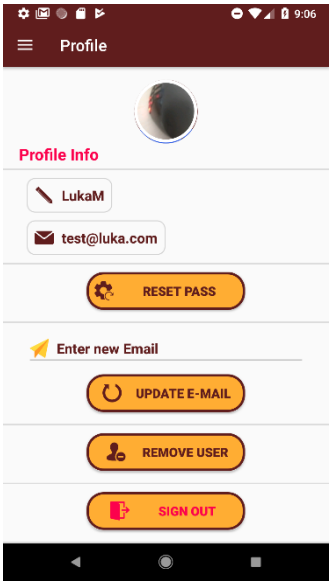
                }
            });
}
```

Sl. 5.8. Prijava putem Gmail-a

5.1.2. Profil korisnika

Profil korisnika sastoji se od osnovnih informacija: podaci o računu (e-mail, korisničko ime), slike korisnika koja se također nalazi u navigacijskoj ladici, promjene lozinke, promjene e-mail adrese, brisanja korisnika, kao i njegova odjava koje ćemo redom spomenuti, a prikazani su slikom 5.9. Nešto detaljnije će se opisati mijenjanje slike profile budući da je za to potrebno dosta stvari. Pomoću dugog pritiska izbacuje se dijalog koji traži od korisnika izbor jedne od opcija kamera ili galerija slika. Pri bilo kojem izboru korisnik mora prihvatiti dopuštenje da aplikacija može koristiti navedene resurse u protivnome neće moći izvršiti akciju. Nakon odabira slike bilo kojim načinom slika se sprema u kružni okvir za koji je korištena biblioteka [41] i [42]. Slika je vidljiva u okviru.

U istom trenutku kada je slika umetnuta u okviru u pozadini se slika sprema u postojeći model korisnika koji je kreiran pri registraciji korisnika, a sada će samo biti dodana još jedna stavka. Model je prikazan na slici 5.10 u Firebase-u i programskom okruženju. Ovakav model će poslužiti u sljedećim koracima kreiranja programskog rješenja kada bude trebalo dohvaćati podatke pojedinog korisnika u segmentima aplikacije. Nacrt spremanja korisnika u bazu podataka platforme dan je slikom 5.11.



Sl. 5.9. Prikaz profila korisnika

```java
public class UserInformation {
    private String username;
    private String email;
    private String photoURL;

    public UserInformation(String
username,
    String  email, String photoURL) {
        this.username = username;
        this.email = email;
        this.photoURL = photoURL;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String
username) {
        this.username = username;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPhotoURL() {
        return photoURL;
    }
    public void setPhotoURL(String
photoURL) {
        this.photoURL = photoURL;
    }
}
```



Sl. 5.10. Prikaz modela u Android Studiju (lijevo) i Firebase-u (desno)

```java
try {
    mAuth = FirebaseAuth.getInstance();
    FirebaseUser currentUser = mAuth.getCurrentUser();
    Context applicationContext = MainMenuActivity.getContextOfApplication();
    applicationContext.getContentResolver();
    InputStream inputStream =
applicationContext.getContentResolver().openInputStream(Objects.requireNonNull(
uri));
    Bitmap bitmap = BitmapFactory.decodeStream(inputStream);
    String filename = String.valueOf(System.currentTimeMillis());
    String path = saveProfilePhoto(getContext(), "images" + filename +
currentUser.getUid(), bitmap).getAbsolutePath();
    updateUserData("images", filename, bitmap);

    String username = tvUsername.getText().toString().trim();
    String email = tvEmail.getText().toString().trim();
    UserInformation userInformation = new UserInformation(username, email,
path);

    mDatabase.child(currentUser.getUid()).setValue(userInformation);
    Glide.with(getActivity())
            .load(path)
            .into(ivProfilePic);

} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Sl. 5.11. Spremanje slike u model postojećeg korisnika Firebase-a

Pri svakoj prijavi korisnika potrebno je dohvatiti specifične podatke za svakog korisnika. To je napravljeno pomoću funkcija koje osluškuju svaku promjenu u aplikaciji (engl. *add value listener*) i na temelju toga dohvaćaju korisnika i postavljaju njegove informacije na zadana mjesta. Dohvaćanje podataka karakterističnih za trenutno prijavljenog korisnika prikazan je na slici 5.12. Provjerava se koji tip korisnika je prijavljen jer metode nisu jednake za oba tipa korisnika, provjera je dana slikom 5.13. Moglo bi doći do preklapanja u sesiji običnog korisnika i već registriranog putem Google-a. Iz tog razloga je ubačen dio koda koji provjerava postoji li sesija Google-ovog računa.

```
mDatabase.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            String name =
Objects.requireNonNull(dataSnapshot.child(user.getUid()).child("username").
getValue()).toString();
            String email =
Objects.requireNonNull(dataSnapshot.child(user.getUid()).child("email").get
Value()).toString();
            tvUsername.setText(name);
            tvEmail.setText(email);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
        }
    });
    StorageReference storageReference =
FirebaseStorage.getInstance().getReference();
    storageReference.child("images" +
user.getUid()).getDownloadUrl().addOnSuccessListener(new
OnSuccessListener<Uri>() {
        @Override
        public void onSuccess(Uri uri) {
            String photo = uri.toString();

            Glide.with(getActivity()).load(photo)
                    .crossFade()
                    .thumbnail(0.5f)
                    .bitmapTransform(new CircleTransform(getActivity()))
                    .diskCacheStrategy(DiskCacheStrategy.ALL)
                    .into(ivProfilePic);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Glide.with(getActivity()).load(urlProfileImg)
                    .crossFade()
                    .thumbnail(0.5f)
                    .bitmapTransform(new CircleTransform(getActivity()))
                    .diskCacheStrategy(DiskCacheStrategy.ALL)
                    .into(ivProfilePic);
        }
    });
}
```

```
GoogleSignInAccount acct =
GoogleSignIn.getLastSignedInAccount(Objects.requireNonNull(getActivity()));
if (acct != null) {
    String personName = acct.getDisplayName();
    String personEmail = acct.getEmail();
    Uri personPhoto = acct.getPhotoUrl();

    tvUsername.setText(personName);
    tvEmail.setText(personEmail);
    // Loading profile image
    Glide.with(this).load(personPhoto)
            .crossFade()
            .thumbnail(0.5f)
            .bitmapTransform(new CircleTransform(getActivity()))
            .diskCacheStrategy(DiskCacheStrategy.ALL)
            .into(ivProfilePic);
}
```

Sl. 5.12. Dohvaćanje podataka za korisnika (gore) i korisnika Gmail-a (dolje)

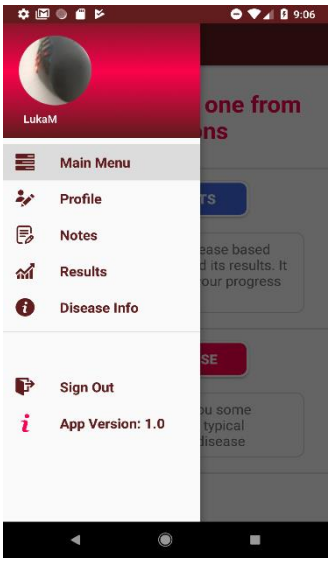
```
GoogleSignInAccount lastSignedInAccount =
GoogleSignIn.getLastSignedInAccount(Objects.requireNonNull(getActivity()));
if (lastSignedInAccount != null) {
    loadUserGmail();
} else {
    loadUserInfo();
}
```

Sl. 5.13. Provjera koji je korisnik prijavljen i koji će se podaci učitati

Opcije vezane uz manipulaciju korisničkim računom neće se detaljno opisivati. Za resetiranje lozinke dovoljno je upisati e-mail te će instrukcije biti poslane na taj isti račun, dok kod ažuriranja e-mail adrese upisuje se nova adresa nakon čega je korisnik odjavljen i može se ponovno prijaviti s istom lozinkom i novim računom. Brisanje korisnika znači odjavu i brisanje svih relevantnih podataka iz aplikacije, odnosno baze podataka o kojoj će se govoriti više u poglavlju 5.3. Na sličan način radi i odjava korisnika, ali je moguća ponovna prijava u bilo kojem trenutku.

5.2. Navigacije glavnim izbornikom pomoću ladice

Nakon što je korisnik ulogiran dolazi na glavni zaslon, glavna navigacija u aplikaciji izvodi se preko navigacijske ladice (slika 5.14) koja se otvori preko izbornika u gornjem desnom kutu. Iz izbornika ladice svaki odabir će označavati otvaranje jednog novog fragmenta unutar glavnog *actvityja*. Primjer otvaranja jednog od fragmenata dan je kod na slici 5.15. Kao zadani fragment postavljen je fragment imena glavni izbornik. Ladica se automatski zatvara kako korisnik ne bi morao svaki puta zatvarati ladicu pri otvaranju novog prikaza.



Sl. 5.14. Prikaz navigacijske ladice

```
  @Override
public boolean onNavigationItemSelected(@NonNull MenuItem item) {
   switch (item.getItemId()) {

       case R.id.nav_MainMenu:
           getSupportFragmentManager().beginTransaction().replace(R.id.frame,
new MainMenuFragment()).commit();

Objects.requireNonNull(getSupportActionBar()).setTitle(fragmentTitles[0]);
           break;

       case R.id.nav_Profile:
           if(profileFragment == null) {
               profileFragment = new ProfileFragment();
           }
           getSupportFragmentManager().beginTransaction().replace(R.id.frame,
profileFragment).commit();

Objects.requireNonNull(getSupportActionBar()).setTitle(fragmentTitles[1]);
           break;
```

Sl. 5.15. Navigacija između fragmenata i ladice

U navigacijskoj se ladici nalazi i slika profila trenutnog korisnika i njegovo korisničko ime, koje je dohvaćeno na isti način kako je opisano u potpoglavlju 5.1.2. Odjava korisnika također je preuzeta iz profila korisnika.

5.3. Postavljanje baze podataka Room

U ovome poglavlju će se opisati osnovne korištene komponente Room baze podataka s njezinim bitnim dijelovima i implementacija koja je korištena za potrebe ove aplikacije. Na isti način koji je napravljen za platformu Firebase potrebno je implementirati ovisnosti koda koji će omogućiti sve metode baze podataka.

5.3.1. Baza podataka aplikacije

Baza podataka predstavlja strukturu u koju će se spremati podaci kako bi se mogli u svakom trenutku mogli iz nje dohvatiti. U ovoj klasi potrebno je definirati i navesti sve modele koji će se koristiti, odnosno kako je navedeno u definiciji entiteti. Ovdje se još stvara sučelje (engl. *interface*) objekta koji pristupa i manipulira podacima za bazu podataka (DAO). Ovakva struktura dana je slikom 5.16.

```
@Database(entities = {Note.class, Result.class, ResultD.class,
LabResults.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public abstract MyDao myDao();
}
```

Sl. 5.16. Prikaz strukture baze podataka

Pri definiranju baze podataka bitno je naglasiti kako je potrebno dopustiti i omogućiti programski odvijanje akcija na glavnoj niti procesa aplikacije vezanih uz bazu podataka, kao što su spremanje, dohvaćanje i brisanje (Slika 5.17).

```
appDatabase = Room.databaseBuilder(getApplicationContext(),
        AppDatabase.class, "database").allowMainThreadQueries().build();
```

Sl. 5.17. Dopuštanje odvijanja akcija na glavnoj niti procesa

5.3.2. Model baze podataka

Modeli baze podataka gotovo su identični po nacrtu strukture ranije navedenom modelu korisnika. Razlika je u tome što ovdje postoji drugačija notacija i zapravo cijeli model je jedan entitet baze podataka dok se podaci u njemu navode kao stupci tog entiteta odnosno tablice kako je i navedeno slikom 5.18. Za dohvaćanje tih podataka koriste se standardno metode koje dohvaćaju i postavljaju podatke, također sve se izvršava preko sučelja DAO.

```
@Entity (tableName = "lab_results")
public class LabResults {

    @PrimaryKey (autoGenerate =
false)
    private int id;

    @ColumnInfo (name = "tsh")
    private float tsh;

    @ColumnInfo (name = "ft4")
    private float ft4;

    @ColumnInfo (name = "ft3")
    private float ft3;

    @ColumnInfo (name = "tpo")
    private float tpo;

    @ColumnInfo (name = "tsi")
    private float tsi;

    @ColumnInfo (name = "tg")
    private float tg;
```

```
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public float getTsh() {
    return tsh;
}

public void setTsh(float tsh) {
    this.tsh = tsh;
}

public float getFt4() {
    return ft4;
}

public void setFt4(float ft4) {
    this.ft4 = ft4;
}

public float getFt3() {
    return ft3;
}
```

Sl. 5.18. Prikaz entiteta baze podataka

5.3.3. Sučelje My Dao

Ovo sučelje, definirano u klasi baze podataka, sadrži sve potrebne metode za manipulaciju podataka u bazi podataka. Metode dohvaćanja i postavljanja se odvijaju preko ovog sučelja u kojemu se mogu definirati razne metode od postavljanja podataka modela u bazu, do njezinog brisanja, ažuriranja i još mnoge druge opcije. U slučaju ove aplikacije koristile su se metode dodavanja podataka, brisanja i dohvaćanja već spremljenih podataka. Prikaz definiranih metoda manipulacije podataka prikazan je na slici 5.19, dok je način dodavanja i dohvaćanja dan slikom 5.20 na primjeru dodavanja podsjetnika u bazu podataka budući da se taj dio neće detaljno opisivati. Isti način koristi se i za brisanje podataka.

```
@Dao
public interface MyDao {

    @Insert
     void addNote (Note note);
    @Insert
    void addResult (Result result);
    @Insert
    void addResultD (ResultD resultD);
    @Insert (onConflict = OnConflictStrategy.REPLACE)
    void addLabRes (LabResults labResults);
    @Query("select * from notes")
     List<Note> getNote();
```

Sl. 5.19. Prikaz nekih od metoda manipulacije podacima

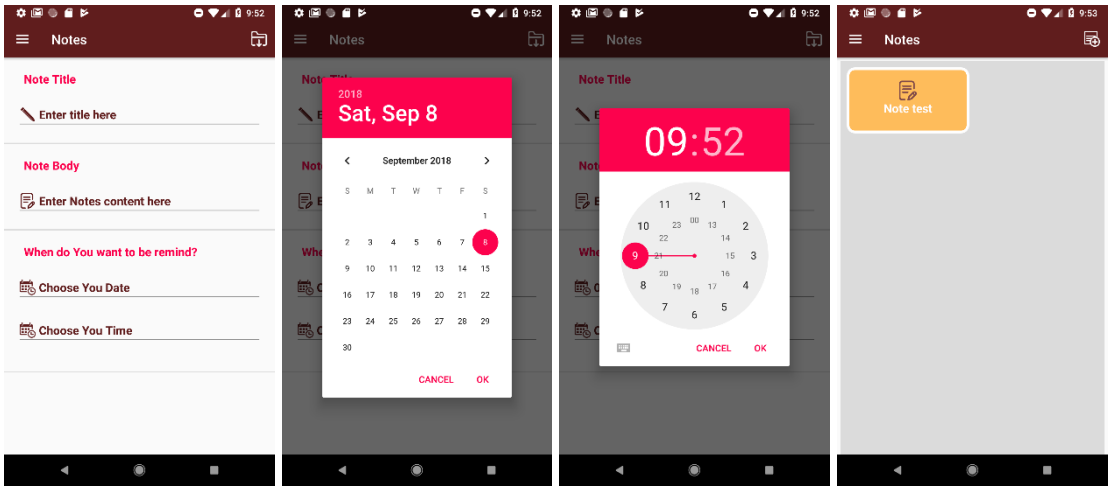
```
Note note = new Note();
note.setTitle(title);
note.setContent(content);
note.setDate(date);
note.setTime(time);

MainMenuActivity.appDatabase.myDao().addNote(note);
```

Sl. 5.20. Osnovni primjer dodavanja podataka u bazu

5.4. Podsjetnici korisniku

Podsjetnici su zamišljeni kao vrsta spremljenih tekstualnih poruka koji bi podsjetili korisnika kako mora obaviti neku aktivnost, kao što je na primjer u ovom slučaju prikladno reći: popiti lijekove ili slično. Izgled podsjetnika dan je na slici 5.21 kao i prikaz spremljenih podsjetnika iz baze podataka. Sastoji se od naslova, tijela gdje se upisuje tekst i zapravo svrha podsjetnika. Preostala dva polja osigurana su za datum i vrijeme koji su realizirani u obliku fragmenta, gdje korisnik odabire datum kada želi da ga se podsjeti i u koje vrijeme što se opisati u sljedećem poglavlju. Spremanje ispunjenih podataka prikazano je slikom 5.20, a naravno odabirom jednog od spremljenih podsjetnika prikazat će se detalji, odnosno isti oni podaci koji su prethodno uneseni. Postoji i opcija brisanja dugim pritiskom na jedan od spremljenih podsjetnika.



Sl. 5.21. Prikaz zaslona bilješki

5.4.1. Notifikacije podsjetnika

Notifikacije su obavijesti samog sustava Android koje u određenom zadanom trenutku prikazu korisniku obavijest, upozorenje i slično. Kako je razina API-ja za ovu aplikaciju 27 što odgovara operacijskom sustavu 8.1, imena Oreo, tako je za ovu verziju sustava potrebno napraviti nešto što do ove verzije nije trebalo raditi, a zove se – notifikacijski kanal (engl. *notification channel*). Kanal predstavlja posebni dio notifikacija koji se tiče samo te aplikacije za koju je kanal napravljen. Tako da se u postavkama može vrlo lako upravljati notifikacija i

izabrati vlastite izbore upravljanja istim. Kreiranje notifikacijskog kanala prikazan je na slici 5.22 zajedno s prikazom kanala u postavkama samog sustava.

```
public class NotesReminder extends Application {

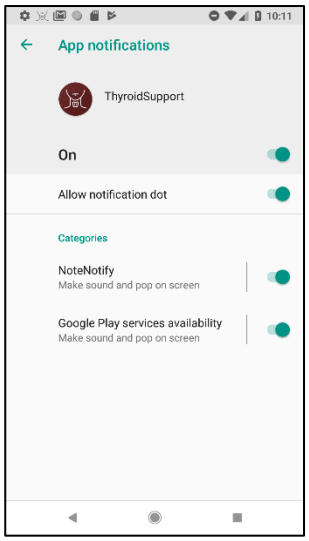
    public static final String CHANNEL_NOTE =
"Notification";

    @Override
    public void onCreate() {
        super.onCreate();
        createNotificationChannels();
    }

    private void createNotificationChannels() {
        if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.O) {
            NotificationChannel noteNotify = new
NotificationChannel(
                    CHANNEL_NOTE,
                    "NoteNotify",
                    NotificationManager.IMPORTANCE_HIGH
            );
            noteNotify.setDescription("NoteNotify");

            NotificationManager manager =
getSystemService(NotificationManager.class);

Objects.requireNonNull(manager).createNotificationChanne
l(noteNotify);
        }
    }
}
```



Sl. 5.22. Notifikacijski kanal kod (lijevo) i postavke sustava (desno)

Po obavljenom prvom koraku potrebno je napraviti klasu koja će primiti podatke o obavijesti i poslati notifikacijskom kanalu. Klasa će biti tipa prijemnik emitiranja (engl. *broadcast receiver*) što znači čim dođe vrijeme za obavijest podatak se šalje toj klasi koja aktivira notifikacije, a dio koda te klase prikazan je slikom 5.23. Ova klasa prima preko dodatnih vrijednosti (engl. *get extra*) obavijest s jedinstvenim identifikatorom kako bi se znalo o kojoj se obavijesti točno radu.

```
public class NotificationReceiver extends BroadcastReceiver {

    public static String NOTIFICATION_ID = "notification-id";
    public static String NOTIFICATION = "notification";

    @Override
    public void onReceive(Context context, Intent intent) {
            NotificationManager notificationManager =
(NotificationManager)context.getSystemService(Context.NOTIFICATION_SERVICE);

            Notification notification =
intent.getParcelableExtra(NOTIFICATION);
            int id = intent.getIntExtra(NOTIFICATION_ID, 1);
            Objects.requireNonNull(notificationManager).notify(id,
notification);
    }
}
```

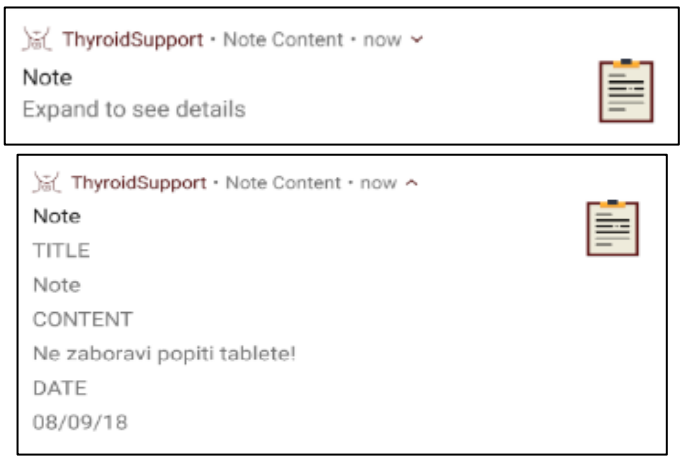
Sl. 5.23. Prijemnik emitiranja notifikacije

Preostaje samo odrediti vrijeme kada se korisnik biti obaviješten o pojedinom podsjetniku preko menadžera alarma ugrađenog u sustavu (engl. *Alarm Manager*). Preuzet će se vrijednosti iz polja gdje su odabrani vrijeme i datum te pomoću Java ugrađene funkcije pretvoriti u vrijeme koje će predstavljati kao vrijeme odgode budući da upravo funkcija notifikacija prihvaća takav tip, što će odgoditi vrijeme dolaska notifikacije do odabranog trenutka. Također svakoj notifikaciji potrebno je dodijeliti jedinstveni identifikacijski broj, ako postoji više notifikacija. Dio opisanog koda može se vidjeti na slici 5.24 kao izgled notifikacije koja na pritisak otvara aplikaciju na glavni zaslon.

```
private void sendOnChannel1 (int delay) {
    titleNote = etTitle.getText().toString();
    contentNote = etContent.getText().toString();
    dateNote = etDate.getText().toString();
    timeNote = etTime.getText().toString();

    Intent broadcastIntent = new Intent(getActivity(),
NotificationReceiver.class);
    broadcastIntent.putExtra(NotificationReceiver.NOTIFICATION_ID,
getUniqueId());
    broadcastIntent.putExtra(NotificationReceiver.NOTIFICATION,
getNotification());
    PendingIntent actionIntent = PendingIntent.getBroadcast(getActivity(), 0,
broadcastIntent, PendingIntent.FLAG_UPDATE_CURRENT);

AlarmManager alarmManager = (AlarmManager)
Objects.requireNonNull(getActivity()).getSystemService(Context.ALARM_SERVICE);
Objects.requireNonNull(alarmManager).set(AlarmManager.RTC_WAKEUP,
futureInMillis, actionIntent);
```



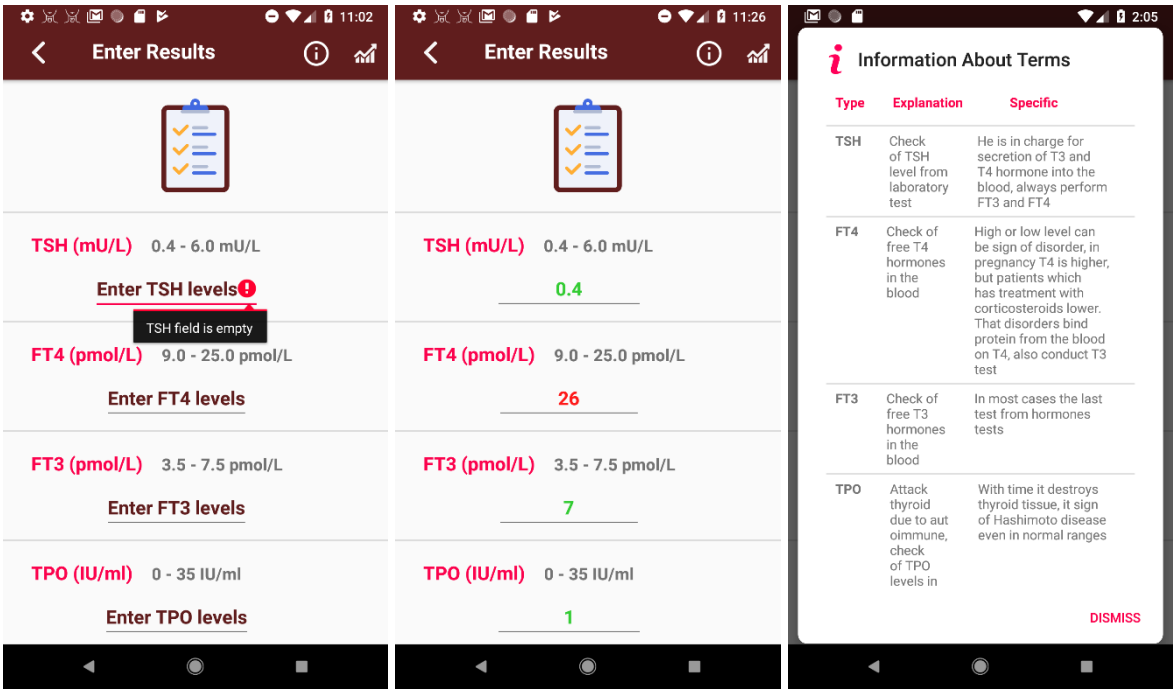
Sl. 5.23. Prikaz metode slanja i izgled notifikacije

5.5. Programsko rješenje potpore oboljelim korisnicima

U ovom poglavlju opisat će se kompletno programsko rješenje pri unosu parametara oboljelih osoba od bolesti štitnjače, preko grafičkog rješenja, generiranja preporuka, spremanja rezultata te pristupanje spremljenim rezultatima.

5.5.1. Unos parametara laboratorijskih testova

Parametri provedenih testova oboljelih korisnika unose se u odgovarajuća polja u obliku brojčanih vrijednosti preciznosti na dvije decimale. Postoji šest parametara, tri se odnose na parametre hormona, a preostala tri na antitijela. Provjera unesenih parametara korisniku ne dopušta prelazak na sljedeći korak, ako prethodno nije unio sve parametre. Ako je vrijednost koja je unesena unutar normalne razine hormona ili antitijela broj će promijeniti boju u zelenu, a u protivnom u crvenu. Vizualni prikaz navedenih značajki prikazan je slikom 5.24, dok je provjera i mijenjanje boje vrijednosti dan slikom 5.25. Vrijednosti pojedinih razina parametara su definirane kao konstantne vrijednosti koje služe kao reference pri svakom daljnjem uspoređivanju pri generiranju rezultata kao što je navedeno u potpoglavlju 4.4.



Sl. 5.24. Prikaz zaslona unosa oboljelih

```
private void checkIsEmpty(){
    if (etParam1.getText().toString().isEmpty()) {
        etParam1.setError("TSH field is empty");
        return;
    }
    else if (etParam2.getText().toString().isEmpty()) {
        etParam2.setError("FT4 field is empty");
        return;
    }
    else if (etParam3.getText().toString().isEmpty()) {
        etParam3.setError("FT3 field is empty");
        return;


@RequiresApi(api = Build.VERSION_CODES.M)
    @Override
    public void afterTextChanged(Editable s) {
        if (etParam1.getText().toString().isEmpty()) {
            return;
        } else {
            if (Double.parseDouble(etParam1.getText().toString()) <
TSH_NORMAL_LEVEL_1 ||
                    Double.parseDouble(etParam1.getText().toString()) >
TSH_NORMAL_LEVEL_2) {
                etParam1.setTextColor(getColor(R.color.red));
            } else {
                etParam1.setTextColor(getColor(R.color.green));
            }
        }
    }
});
```

Sl. 5.25. Prikaz provjere unosa (gore) i postavljanje boje unosa parametara (dolje)

Odabirom ikonice u gornjem desnom kutu otvara dijalog koji sadrži osnovne informacije o parametrima preuzetih iz poglavlja 4.4.

5.5.2. Implementacija algoritma na osnovi unosa korisnika

Kako se grafički rezultat sastoji od brojevnih vrijednosti koje korisnik unosi, unesene vrijednosti se moraju nekako poslati sljedećem zaslonu, a to je učinjeno na način da su se vrijednosti iz svih polja spremile u varijable. Varijable će se spremiti u novokreirano polje koje prima podatke kakvi su uneseni od strane korisnika, decimalni brojevi (Slika 5.26).

```
float [] param = new
float[]{(Float.parseFloat(etParam1.getText().toString())),
        (Float.parseFloat(etParam2.getText().toString())),
        (Float.parseFloat(etParam3.getText().toString())),
        (Float.parseFloat(etParam4.getText().toString())),
        (Float.parseFloat(etParam5.getText().toString())),
        (Float.parseFloat(etParam6.getText().toString()))};

Intent intent = new Intent (PatientInputActivity.this,
PatientResultActivity.class);
intent.putExtra(EXTRA1, param);
```

Sl. 5.26. Spremanje parametara u polje i slanje na drugi zaslon

Nadalje, potrebno je generirati preporuke. U metodama se izvršava algoritam koji kao rezultat daje polje niza znakova koje se šalje u sljedeći korak koji prikazuje rezultate. Svaki uneseni

parametar se uspoređuje s njegovim konstantama i ovisno nalazi li se u optimalnoj , povećanoj ili premaloj razini, varijabli se pridjeljuje niz znakova iz resursa niza znakova, gdje su prethodno definirane. Sve varijable se spremaju u kreirano polje. Ovaj postupak vidljiv je na slici 5.27.

```
private void checkSuggestion() {
   checkTSH();
   checkFT4();
   checkFT3();
   checkTPO();
   checkTSI();
   checkTG();
}

private void checkFT3() {
    if (Double.parseDouble(etParam3.getText().toString()) >
FT_3_NORMAL_LEVEL_2) {
        ft3Res = getResources().getString(R.string.above_normal_ft3);
    } else if (Double.parseDouble(etParam3.getText().toString()) <
FT_3_NORMAL_LEVEL_1) {
        ft3Res = getResources().getString(R.string.below_normal_ft3);
    } else if (Double.parseDouble(etParam3.getText().toString()) >=
FT_3_NORMAL_LEVEL_1 ||
            Double.parseDouble(etParam3.getText().toString()) <=
FT_3_NORMAL_LEVEL_2) {
        ft3Res = getResources().getString(R.string.normal_ft3);
    }
}
checkSuggestion();
String [] suggest = new String[]{tshRes, ft4Res, ft3Res, tpoRes, tsiRes,
tgRes};
```

Sl. 5.27. Generiranje preporuka i spremanje u polje niza znakova

5.5.3. Prikaz rezultata grafički i generiranih preporuka pomoću algoritma

Sada kada su vrijednosti poslane, potrebno ih je pravilno preuzeti i iskoristi kako bi tvorile pravi rezultat. Pri grafičkom prikazu koristi se biblioteka *MP Android Chart*. Grafički prikaz koji će se koristiti je linijski graf. On pomoću polja decimalnih brojeva koji se predaju listi generira graf koji međusobno spaja točke iz iste liste i na taj način se dobije linijski prikaz. Upravo toj listi predaju parametre koje je korisnik unio u prethodnom koraku (Slika 5.28). Tako uspješno predane vrijednosti, na grafu izgledaju kao na slici 5.29.

```
private void addData(){
    List<Entry> new_record = new ArrayList<>();
    List<Entry> previous_record = new
ArrayList<>();
    param =
getIntent().getFloatArrayExtra(EXTRA1);

    Entry tsh_new = new Entry(0f, param[0]);
    new_record.add(tsh_new);
    Entry ft4_new = new Entry(1f, param[1]);
    new_record.add(ft4_new);
    Entry ft3_new = new Entry(2f,param[2]);
    new_record.add(ft3_new);
    Entry tpo_new = new Entry(3f, param[3]);
    new_record.add(tpo_new);
    Entry tsi_new = new Entry(4f, param[4]);
    new_record.add(tsi_new);
    Entry tg_new = new Entry(5f, param[5]);
    new_record.add(tg_new);
```

Sl. 5.28. Pridruživanje unesenih parametara listi vrijednosti linijskog grafa

Cilj je pratiti napredak korisnika kroz nove unose, no kako će se to realizirati. Način na koji je ovdje to napravljeno je sljedeći: prvi unos se prikazuje sam na grafu, ali bez rezultata budući da se nema ni sa čime usporediti, što je i navedeno na mjestu gdje bi trebao biti prikazan rezultat. Drugi rezultat bi prepisao stari i ništa se ne bi dogodilo. Iz tog razloga dodana je još jedna lista za vrijednosti na grafu. Na pritisak gumba za spremanje rezultata ugrađena je funkcija spremanja postojećih rezultata u bazu podataka Room. Na sljedećem unosu u prvu listu bit će uvršteni ponovno uneseni rezultati, a u drugu listu će biti dohvaćeni prethodno spremljeni rezultati tako da će korisnik moći vidjeti svoj napredak ili pogoršanje. Prikaz spremanja i dohvaćanja vrijednosti prikazan je na slici 5.29.

```
getIntent().getFloatArrayExtra(EXTRA
1);
LabResults lab = new LabResults();
float tsh = param [0];
float ft4 = param [1];
float ft3 = param [2];
float tpo = param [3];
float tsi = param [4];
float tg = param [5];
lab.setTsh(tsh);
lab.setFt4(ft4)
lab.setFt3(ft3);
lab.setTpo(tpo);
lab.setTsi(tsi);
lab.setTg(tg);
appDatabase.myDao().addLabRes(lab);
```

```
List<LabResults> labR =
appDatabase.myDao().getLabRes();
for (LabResults labb: labR){
    tshR = labb.getTsh();
    ft4R = labb.getFt4();
    ft3R = labb.getFt3();
    tpoR = labb.getTpo();
    tsiR = labb.getTsi();
    tgR = labb.getTg();
    Entry tsh_pre = new Entry(0f,
tshR);
    previous_record.add(tsh_pre);
    Entry ft4_pre = new Entry(1f,
ft4R);
    previous_record.add(ft4_pre);
    Entry ft3_pre = new Entry(2f,
ft3R);
    previous_record.add(ft3_pre);
    Entry tpo_pre = new Entry(3f,
tpoR);
    previous_record.add(tpo_pre);
    Entry tsi_pre = new Entry(4f,
tpoR);
    previous_record.add(tsi_pre);
    Entry tg_pre = new Entry(5f,
tpoR);
    previous_record.add(tg_pre);
}
```

Sl. 5.29. Spremanje vrijednosti parametara i dohvaćanje

U drugom unosu doći se do generiranja rezultata tako što će se usporediti svaki pojedini novouneseni parametar i spremljeni parametri, pa ovisno o istom, boljem ili gorem rezultatu za pojedinu razinu hormona i antitijela ispisat će se odgovarajuća poruka (Slika 5.30). Konačni prikaz zaslona rezultata dan je slikom 5.31.

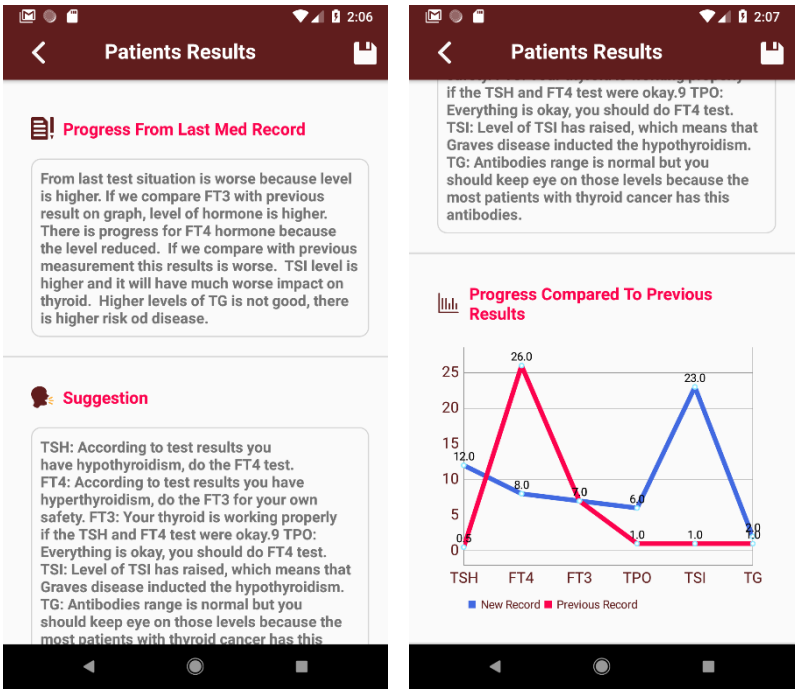
```
private void ft3Res (){
    List<LabResults> labR =
appDatabase.myDao().getLabRes();
    for (LabResults labb: labR) {

        tshR = labb.getTsh();
        ft4R = labb.getFt4();
        ft3R = labb.getFt3();
        tpoR = labb.getTpo();
    }
   param =
getIntent().getFloatArrayExtra(EXTRA1);
    float resCompare = compare(param[1],
ft3R);
    if (resCompare > 0) {
        res1 =
getString(R.string.FT3_worse);
    } else if (resCompare == 0) {
        res1 =
getString(R.string.FT3_stale);
    } else {
        res1 =
getString(R.string.FT3_progress);
    }
}
```

```
private void setResult() {
    if
(appDatabase.myDao().getLabRes().isEmpt
y()) {
        String res =
getString(R.string.no_data);
        tvProgressContent.setText(res);
    }
    else {
        tshRes();
        ft4Res();
        ft3Res();
        tpoRes();
        tsiRes();
        tgRes();
        String result = res0 + " "  +
res1 + " " +
                " " + res2 + " " + " "
+ res3 + " " + " "
                + res4 + " " + " " +
res5;

tvProgressContent.setText(result);
    }
}
```

Sl. 5.30. Generiranje rezultata



Sl. 5.31. Prikaz rezultata

5.5.4. Spremanje prikazanih rezultata u bazu podataka

Odabirom opcije spremanja na zaslonu rezultata, rezultati i preporuke se spremaju u bazu podataka Room, dok se graf sprema u galeriju mobitela pomoću ugrađene funkcije iz biblioteke, a u bazu se sprema put (engl. *path*) do te slike (Slika 5.32).

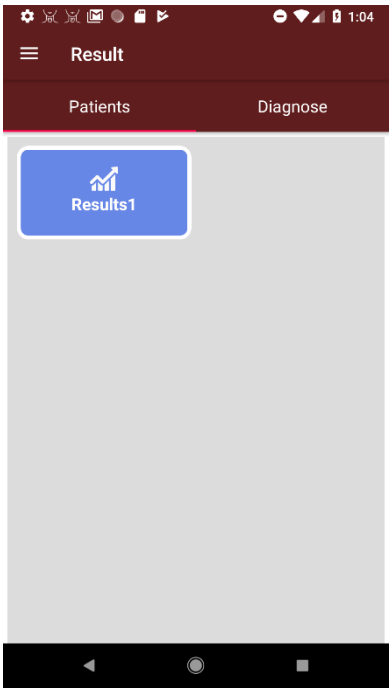
```
String image_title_p = "image_patients" + getUniqueId();
chResults.saveToGallery(image_title_p, 90);
File extBaseDir = Environment.getExternalStorageDirectory();
String pathP = extBaseDir.getAbsolutePath() + "/DCIM/" + image_title_p +
".jpg";

String resuts = tvProgressContent.getText().toString().trim();
String suggest = tvSuggestionContent.getText().toString().trim();
Result result = new Result();
result.setResultInfo(resuts);
result.setSuggestion(suggest);
result.setGraphResult(pathP);

appDatabase.myDao().addResult(result);
Toasty.success(PatientResultActivity.this,
        "Results Data Is Successfully Saved").show();
```

Sl. 5.32. Spremanje svih parametara na zaslonu rezultata

Odaberemo li opciju rezultati u navigacijskoj ladici dolazimo u fragment (Slika 5.33) koji ima dva taba od kojih je jedan za već oboljele. Tamo se mogu pregledati svi do sada spremljeni rezultati, a pritiskom na jedan od rezultata bit će prikazani upravo ti odgovarajući podaci, a to je postignuto kodom sa slike 5.34 i može se primijeniti na sva dohvaćanja iz baze i prikazivanje rezultata, a dugim pritiskom na jedan od rezultata moguće je obrisati rezultat.



Sl. 5.33. Prikaz liste rezultata

```
List<Result> result = appDatabase.myDao().getResult();
    for (Result results: result){
        resultP = results.getResultInfo();
        suggestionP = results.getSuggestion();
        graphP = results.getGraphResult();
    }

    tvProgressContent.setText(resultP);
    tvSuggestionContent.setText(suggestionP);

    Glide.with(getActivity()).load(graphP)
            .into(ivPatientsGraph);
}
```

Sl. 5.34. Dohvaćanje podataka iz baze i slanje u drugi fragment

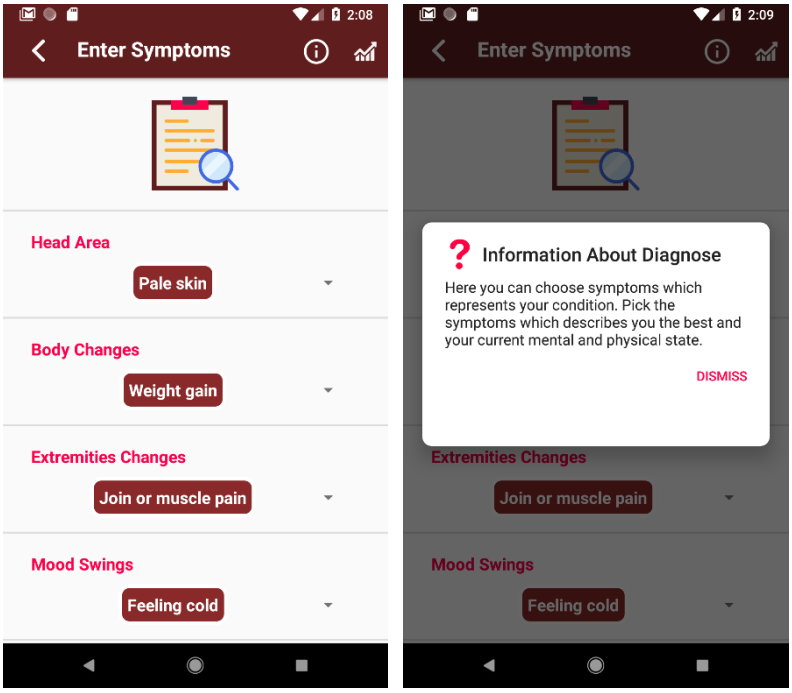
5.6. Programsko rješenje potpore potencijalno oboljelim korisnicima

U ovom poglavlju objasnit će se principi dijagnosticiranja potencijalnog oboljenja od bolesti štitnjače za korisnike koji sumnjaju na neku od bolesti ili jednostavno za osobe koje žele probati gdje će ih njihov odabir odvesti. Na sličan način kao i u prošlom poglavlju obradit će se od interakcije korisnika do generiranja rezultata, preporuka i u konačnici grafičkog prikaza rezultata na temelju odabira.

5.6.1. Odabir simptoma korisnika

Postupak kreće tako da korisnik odabere simptome koji su podijeljeni u pet kategorija koji se redom odnose na područje glave, promjena u tijelu, promjena u ponašanju ili oboljenje od neke bolesti koje mogu indirektno utjecati na bolesti štitnjače. Bolesti se nalaze unutar padajućeg izbornika koji sadrži određene vrijednosti, simptome podijeljene u kategorije kako je to

navedeno u potpoglavlju 4.4. Za ovaj postupak nije potrebna provjera budući da je za svaku kategoriju već predefiniran neki od simptoma. Izgled ovog ekrana zajedno s dijalogom koji objašnjavam postupak odabiranja simptoma vidljiv je na slici 4.35.



Sl. 5.35. Zaslon odabiranja simptoma i dijalog pojašnjenja

5.6.2. Implementacija algoritma na osnovu odabira korisnika

Odabrani simptomi odabirom opcije za prikaz rezultata se spremaju u polje niza znakova, no podaci se ne mogu samo tako poslati u sljedeći korak prikaza rezultata jer su za prikaz grafova potrebne brojčane vrijednosti. Sada je definirano polje niza znakova s odabirom korisnika (Slika 5.36). Preuzimanjem niza znakova pojedinih bolesti za simptomima moguće je usporediti odabire iz padajućih izbornika sa svakim pojedinim poljem simptoma bolesti.

```
param1 = sParam1.getSelectedItem().toString();
param2 = sParam2.getSelectedItem().toString();
param3 = sParam3.getSelectedItem().toString();
param4 = sParam4.getSelectedItem().toString();
param5 = sParam5.getSelectedItem().toString();
parameters = new String[]{param1, param2, param3, param4, param5};
```

Sl. 5.36. Prikaz uspoređivanja polja niza znakova i pretvaranje u brojevne vrijednosti

Za svaki odabir koji se poklapa s nekim od elemenata iz polja bolesti brojač će se povećati za jedan. Na takav način dobit će se priželjkivane brojčane vrijednosti. Kako je potreban postotak poklapanja taj broj iz brojača će se modificirati na način da će se podijeliti s ukupnim brojem svih simptoma i još pomnožiti sa 100 kako bi dobili postotak. Takva vrijednost se sprema u pojedinu varijablu za svaku bolest. U kreirano polje decimalnih brojeva ubacujemo varijable

redom kojim će se prikazivati i na grafu. Ovaj postupak pretvaranja uspoređivanja polja niza znakova u brojevne vrijednosti dan je slikom 5.37, a prikaz spremanja varijabli u polje Slikom 3.38.

```
private void check_d0 () {
    String[] a = getResources().getStringArray(R.array.Disease0);
    int matchCount0 = 0;
    for (int i = 0, j = 0; i < a.length && j < parameters.length; )
{
        int res = a[i].compareTo(parameters[j]);
        if (res == 0) {
            matchCount0++;

            i++;
            j++;
        } else if (res < 0) {
            i++;
        } else {
            j++;
        }
        d0 = (float) (matchCount0) / (a.length) * 100;
    }
}
```

Sl. 5.37. Prikaz uspoređivanja polja niza znakova i pretvaranje u brojevne vrijednosti

```
check_d0();
check_d1();
check_d2();
check_d3();
check_d4();
check_d5();
check_d6();

float [] percentage = new float [] {d0, d1, d2, d3, d4, d5, d6};
Intent intent = new Intent(DiagnoseInputActivity.this,
        DiagnoseResultActivity.class);
intent.putExtra(EXTRA, percentage);
```

Sl. 5.38. Prikaz uspoređivanja polja niza znakova i pretvaranje u brojevne vrijednosti

5.6.3. Prikaz rezultata grafički i generiranih preporuka pomoću algoritma

Poslane vrijednosti potrebno je dohvatiti i ugraditi unutar liste grafa. Kod ovog tipa korisnika za prikaz rezultata koristit će se stupčasti graf. Ovisno o postotku poklapanja bolesti on će narasti ili u slučaju nepoklapanja bit će nula odnosno neće postojati. Podaci se ugrađuju unutar liste kako je prikazano slikom 5.39.

```
public void addbarEntries() {
    barEntries = new ArrayList<>();
    percentage = getIntent().getFloatArrayExtra(EXTRA);

    barEntries.add(new BarEntry(0f, percentage[0]));
    barEntries.add(new BarEntry(1f, percentage[1]));
    barEntries.add(new BarEntry(2f, percentage[2]));
    barEntries.add(new BarEntry(3f, percentage[3]));
    barEntries.add(new BarEntry(4f, percentage[4]));
    barEntries.add(new BarEntry(5f, percentage[5]));
    barEntries.add(new BarEntry(6f, percentage[6]));

    set = new BarDataSet(barEntries, "Percentage");
    set.setColor(Color.rgb(255, 173, 51));
    barData = new BarData(set);
```

Sl. 5.39. Umetanje vrijednosti u stupčasti graf

U rezultatima je potrebno prikazati onu bolesti i postotak koji imaju najveći postotak poklapanja s odabranim simptomima. Kako bi se to ostvarilo potrebno je pronaći najveći postotak u polju koje je poslano. Napravljen je algoritam sortiranja i traženja najvećeg iznosa u polju s postotcima podudaranja. Potrebno je pronaći i ime bolesti koje odgovara tom postotku. Iskoristit će se polje niza znakova koje je koristi kako bi se prikazala imena bolesti na osi x. Na taj način će se varijabli pridružit određeni element polja. Broj položaja najvećeg postotka u polju iskoristit će se i kod polja s nizom znakova bolesti budući da je jednak broj elemenata u polju i kod jednog i drugog polja – sedam. To je ostvareno pomoću koda na slici 5.40.

```
public void setResult() {
    percentage = getIntent().getFloatArrayExtra(EXTRA);

    for (int i = 0; i < percentage.length; i++) {
        if (percentage[i] > max) {
            max = percentage[i];
            diseaseName = xValues[i];
        }
    }

    DecimalFormat two = new DecimalFormat("0.00");
    String bestMatch = two.format(max);
    String results = "Najveći postotak poklapanja ima" + " " +
            diseaseName + " " + " " + "u iznosu od" + " " + bestMatch +
" " + "%";
    tvMatchContent.setText(results);
}
```

Sl. 5.40. Generiranje najvećeg postotka i naziva bolesti

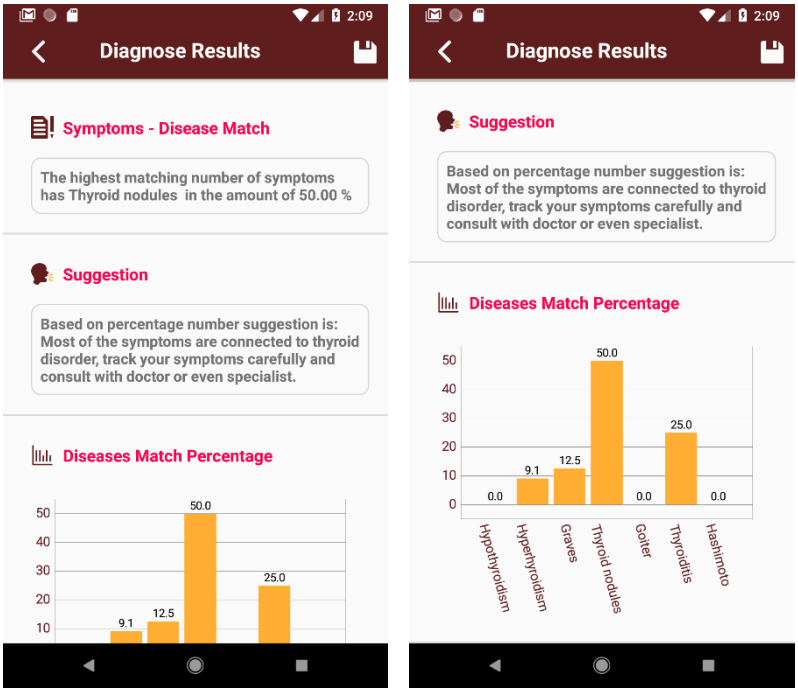
Određivanje preporuka napravljeno je tako da se uzela najveća vrijednost i iz polja svih postotaka ponovno i to uspoređivalo za rasponima vrijednostima ranije definiranim. Ovisno kojem rasponu pripada dodijeljen je određen niz znakova. To izgleda kao na slici 5.41. Sada kada su svi elementi dobiveni ne ostaje drugo nego pogledati sliku 5.42 i vidjeti kako to sve skupa izgleda.

```
private void setSuggestion() {
    check35();
    check36_70();
    check71_100();

    String suggestion = "Na temelju postotaka dobivenih rezultata
preporuka je sljedeća:" + " " + suggest;
    tvSuggestionContent.setText(suggestion);
}

private void check35() {
    if (max < 35) {
        suggest = getString(R.string.percentage_35);
    } else {
        return;
    }
}
```

Sl. 5.41. Generiranje najvećeg postotka i naziva bolesti

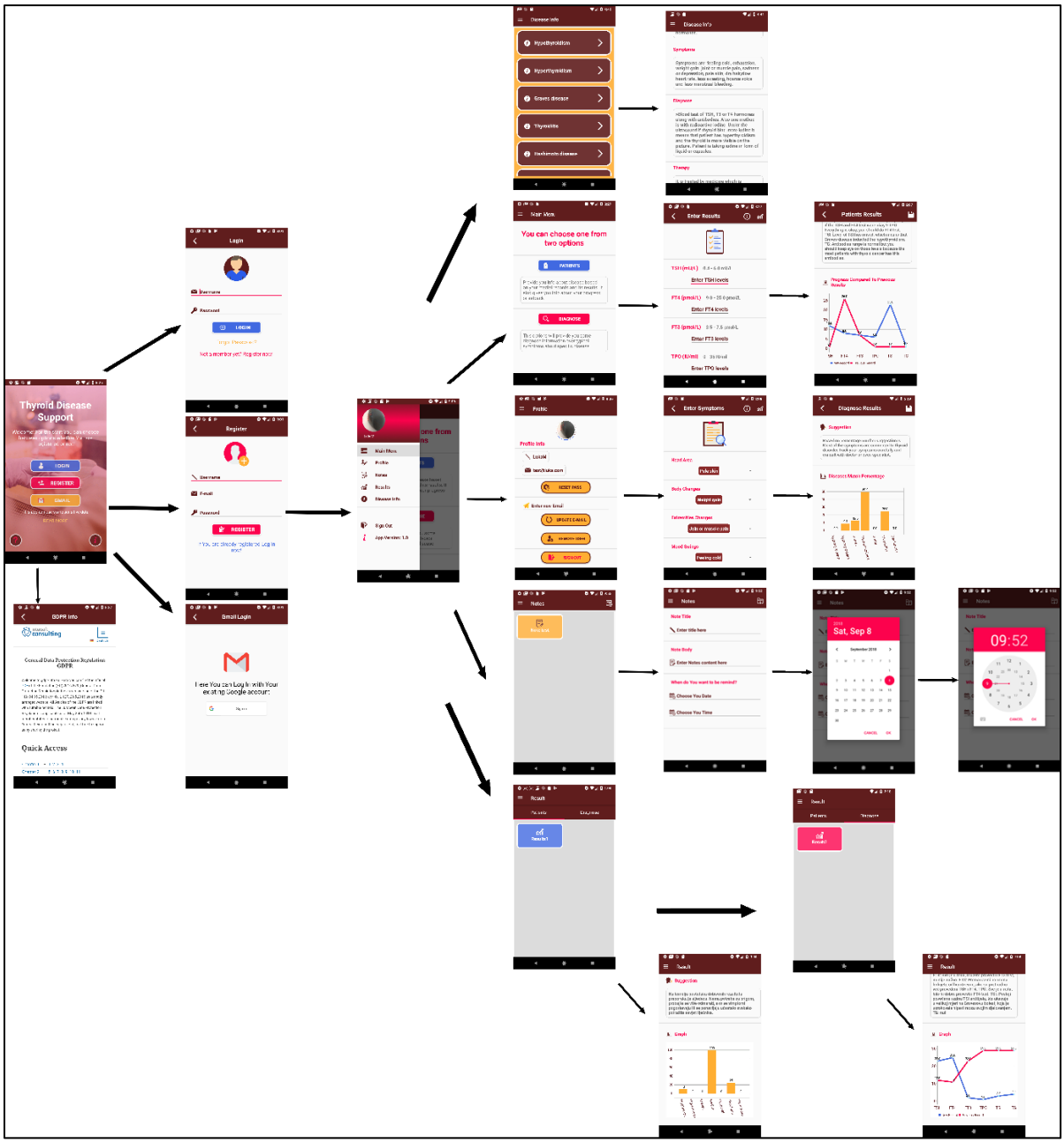


Sl. 5.42. Konačni prikaz zaslona potencijalnog oboljelih korisnika

Spremanje rezultata i dohvaćanje istih u fragmentu se u tabu dijagnoza radi po istom principu opisanom u potpoglavlju 5.5 te neće biti govora o spremanju ovih podataka na zaslonu rezultata.

5.7. Grafički prikaz svih zaslona aplikacije

Nakon napravljenog programskog rješenja na slici 5.43 dan je potpuni prikaz navigacije i svih zaslona kroz aplikaciju. Ovaj prikaz će pomoći svakom korisniku pri korištenju aplikacije služeći mu kao neka vrsta upute. Navigacija je koncipirana tako da se pristupa preko navigacijske ladice koje zapravo ključ navigacije i iz koje se može otići u bilo koji dio, ali isto tako i odjaviti korisnika. Više detalja dano je u potpoglavlju 5.2.



Sl. 5.43. Prikaz svih zaslona aplikacija i njihove međusobne navigacije u aplikaciji

6. TESTIRANJE MOBILNE APLIKACIJE

Tema u nastavku je definiranje testnih slučajeva, provođenje testova, unaprjeđenje koda na osnovi testova, kao i krajnje generiranje izvještaja o testiranju koja će zaokružiti cijeli postupak testiranje.

6.1. Implementacija testnog alata

Prije početka pisanja ikakvih testova u Android Studiju, potrebno je u Gradle-u implementirati ovisnosti, kojima u programskom okruženju omogućujemo korištenje svih funkcija i alata koji su na potrebni za testiranje korištenjem alata Espresso. Princip je implementacije izvršen kao i kod Firebase-a i Room-a te sinkronizacijom s postojećom implementacijom. Testovi se izvršavaju unutar programskog okvira JUnit4 kako je naznačeno na slici 6.1. Namijenjen je pisanju automatiziranih testova na temelju korisničkog sučelja. Iz njega proizlaze sve anotacije pri pisanju testova kako je spomenuto u potpoglavlju 6.3.2 u nastavku.

```
import org.junit.runner.RunWith;

@RunWith(AndroidJUnit4.class)
```

Sl. 6.1. Prikaz implementacije JUnit programskog okvira unutar Espresso testa

6.2. Pisanje testnih slučajeva

Osmišljavanje testnih slučajeva bazira se na temeljnim funkcionalnostima aplikacije. Testovi opterećenja ili pak neki drugi koji testiraju sve aspekte aplikacije neće se provoditi. U tablici 6.1. prikazani su testni slučajevi koji su osmišljeni na temelju glavnih funkcionalnosti programski ostvarene aplikacije vezanih uz prijavu korisnika, unosa parametara te komunikacije s bazom podataka.

Tab. 6.1. Prikaz testnih slučajeva

Funkcionalnost	Koraci	Opis
Prijava i odjava korisnika	1. Ispunjavanje korisničkih podataka 2. Klik na gumb za prijavu 3. Otvaranje navigacijske ladice 4. Klik na opciju izbornika za odjavu 5. Klik na potvrdu odjave korisnika	Provjera autentifikacije korisnika
Unos parametara već oboljelih korisnika i spremanje rezultata u bazu Room	1. Otvaranje activityja za unos parametara testova 2. Unos parametara 3. Klik na opciju za kalkulacijom na akcijskoj traci 4. Odobravanje dopuštenja za pristup resursima 5. Klik na opciju za spremanje u bazu podataka Room 6. Otvaranje navigacijske ladice i odabir rezultata 7. Pregled detalja rezultata i brisanje rezultata	Provjera funkcionalnosti prve glavne značajke aplikacije
Odabir simptoma potencijalno oboljelih korisnika i spremanje u bazu Room	1. Otvaranje activityja za odabir simptoma korisnika 2. Odabir simptoma 3. Klik na opciju za kalkulacijom na akcijskoj traci 4. Odobravanje dopuštenja za pristup resursima 5. Klik na opciju za spremanje u bazu podataka Room 6. Otvaranje navigacijske ladice i odabir rezultata 7. Pregled detalja rezultata i brisanje rezultata	Provjera druge glavne značajke aplikacije

Promjena slike profila preko kamere	1. Prijava korisnika 2. Otvaranje navigacijske ladice 3. Odabir opcije profila iz izbornika 4. Dugi klik na sliku i izbor opcije kamera 5. Prihvaćanje dopuštenja za kameru i lokaciju	Provjera manipulacije korisnicima

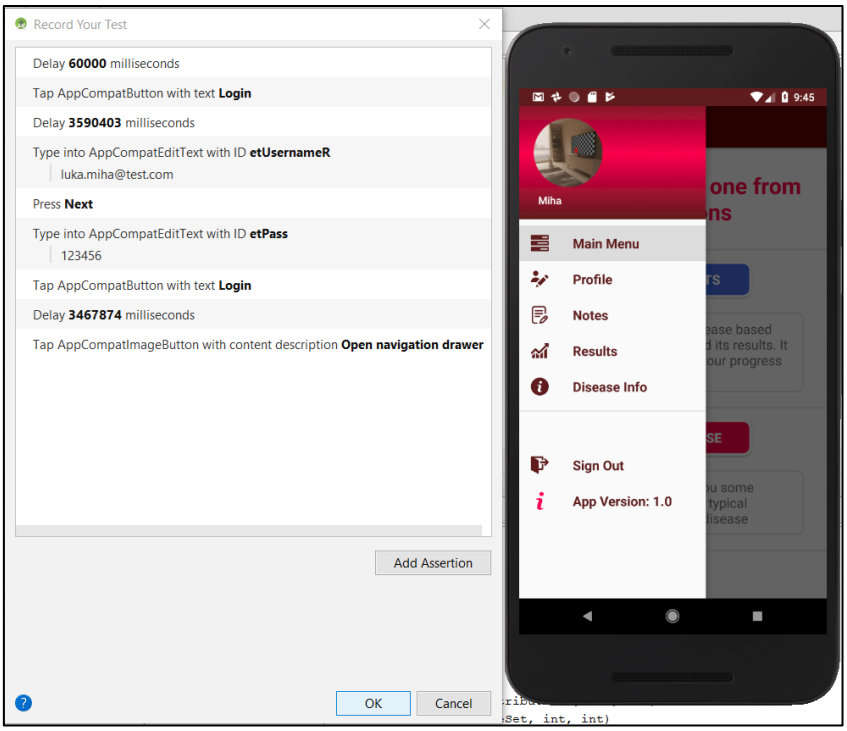
6.3. Provođenje testova

U pootpoglavlju bit će opisani principi provođenja testova od najjednostavnijih do najtežih te prepreke i problemu pri njihovom provođenju.

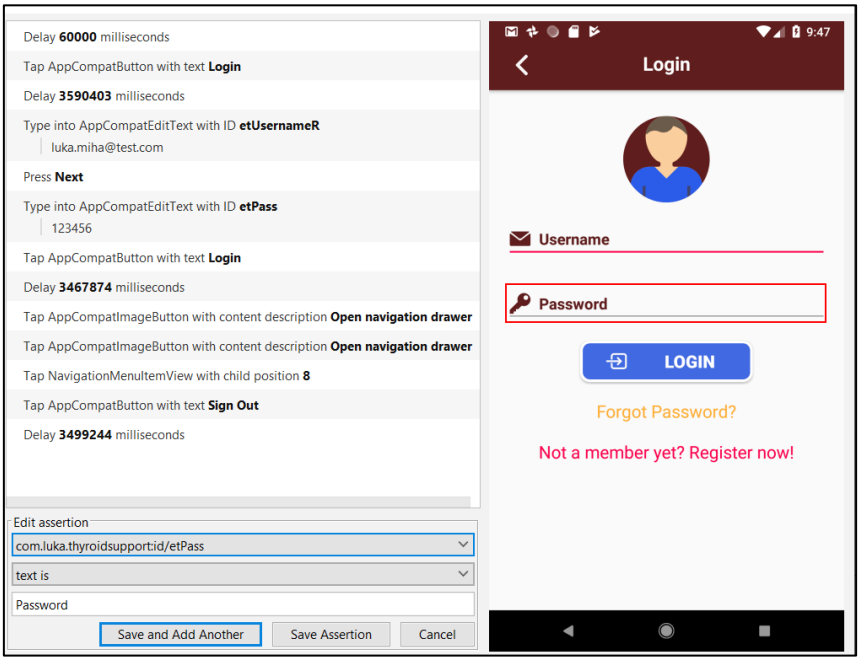
6.3.1. Snimač testova

Snimač testova služi generiranju testova korisničkog sučelja na način da korisnik odabire elemente na zaslonu aplikacije te sve snima redom kako se odabiru elementi. Ovaj alat radi do određene razine i složenosti. Pri završetku korištenja aplikacije testni kod se generira i dostupan je korisniku što se može vidjeti i na slici 6.2.

Druga važna opcija odnosi se na otkrivanje elemenata odabirom elementa na ekranu aplikacije. Za korištenje ove opcije uopće nije potrebno koristiti snimač. Odabirom opcije umetanja (engl. assertions) trenutni zaslon aplikacije sa svim elementima se spremi te je dostupan korisniku na korištenje. Odabir elementa daje detalje o tom elementu, a za testiranje u ovoj okolini najvažniji je zasigurno identifikator (ID) elementa. Takav način otkrivanja identifikatora može (Slika 6.3.) poslužiti kao pomagalo otkrivanja identifikatora prije pisanja testova, ako se unaprijed prema testnim slučajevima zna koji će elementi biti potrebni za izvođenje testa. Na taj način se ne bi tražio identifikator za svaki element pojedinačno i pritom bi se uštedilo vrijeme.



Sl. 6.2. Prikaz snimača testa zajedno s koracima koji su doveli do prikazanog zaslona na emulatoru



Sl. 6.3. Prikaz opcije snimača testa pri identifikaciji elemenata

6.3.2. Pisanje Espresso testova

Nakon snimača testova koji se neće koristiti pri testiranju ove aplikacije, više o razlogu u sljedećem poglavlju, objasnit će se koraci izvršavanja pisanja Espresso testa na složenijem primjeru aplikacije.

Najprije je potrebno saznati nešto više o konstrukciji testova. Ključne riječi za svaki Espresso test su anotacije koje se pišu prije svake metode odnosno testa, a redom su: @Rule, @Test, @Before i @After. Pravila predstavljaju ono što se treba izvršiti u aplikaciji prije pokretanja samog testa. Kada dođe do toga trenutka u aplikaciji kada zahtjeva određenu radnju, informaciju ili značajku, test će se neometano nastaviti izvoditi. Pravila se mogu definirati ovisno o elementima i resursima koje aplikacija treba pri radu. Primjer pravila dan je slikom 6.4. Test označava tijelo samog testa u kojem se definiraju radnje na svaki od elemenata korisničkog sučelja preko jedinstvenog identiteta. Anotacije prije i poslije mogu se koristit i ne moraju ovisno o složenosti testiranja. Mogu se koristiti zadane funkcije postavljanja za prije i prekida testa za poslije koje u slučaju pogreške izbacuju iznimku i test se prekida.

```
@Rule
public ActivityTestRule<LoginActivity> mActivityRule = new
ActivityTestRule<>(
        LoginActivity.class);
```

Sl. 6.4. Prikaz pravila koje otvara početni zaslon pri pokretanju testa

Test koji će se opisati odnosi se na unos parametara, spremanja istih te na kraju pregled spremljenih rezultata i njegovo brisanje. Potrebno je definirati pravila za ovaj test, a ima ih dosta kao što je vidljivo na slici 6.5. Pravilo sadrži definiranje početnog zaslona, sa svim parametrima laboratorijskih testova, sve do prihvaćanja dopuštenja pristupa galeriji uređaja zbog spremanja slike grafa.

```
@Rule
public ActivityTestRule<PatientInputActivity> mActivityRule = new
ActivityTestRule<>(
        PatientInputActivity.class);
        private double TSH = 3.0;
        private double FT4 = 3.0;
        private double FT3 = 3.2;
        private double TPO = 3.0;
        private double TSI = 3.5;
        private double FG = 3.0;

@Rule
public GrantPermissionRule grantPermissionRule = GrantPermissionRule
        .grant(Manifest.permission.WRITE_EXTERNAL_STORAGE);

@Rule
public ActivityTestRule<MainMenuActivity> mActivityRule1 = new
ActivityTestRule<>(
        MainMenuActivity.class);
```

Sl. 6.5. Prikaz pravila koja se definiraju za test unosa parametara i pregledavanje rezultata

Kao što bi učinio i korisnik, potrebno je popuniti polja s brojevima odnosno iznosima testova definiranih u pravilu. Definirane varijable samo pozovemo na za to predviđeno mjesto (Slika

6.6). Nakon unosa parametara potrebno je prijeći u sljedeći korak ne bi li se prikazali rezultati. Elemente je dovoljno naći po identifikatoru ili u nekim drugim slučajevima prema opisu ili nazivu niza znakova.

```
@Test
public void enter_tests(){

    onView((withId(R.id.etParam1)))
            .perform((typeText(String.valueOf(TSH))), closeSoftKeyboard());
    onView((withId(R.id.etParam2)))
            .perform((typeText(String.valueOf(FT4))), closeSoftKeyboard());

    onView((withId(R.id.etParam3)))
            .perform((typeText(String.valueOf(FT3))), closeSoftKeyboard());

    onView((withId(R.id.etParam4)))
            .perform((typeText(String.valueOf(TPO))), closeSoftKeyboard());

    onView((withId(R.id.etParam5)))
            .perform((typeText(String.valueOf(TSI))), closeSoftKeyboard());

    onView((withId(R.id.etParam6)))
            .perform((typeText(String.valueOf(FG))), closeSoftKeyboard());
```

Sl. 6.5. Prikaz popunjavanja polja parametara korisnika

Na zaslonu rezultata potrebno je prihvatiti dopuštenje koje se izvršava automatski zbog definiranog pravila. Odabirom elementa spremanja rezultata, spremaju se podaci kako je prikazano slikom 6.6. Ponekad je potrebno provjeriti postoji li element na zaslonu naredbom provjeri koja provjerava je li element sa baš tim specifičnim identifikatorom prikazan.

```
onView(withId(R.id.action_calculate_patients)).perform(click());
onView(withId(R.id.action_save_results_patients)).check(matches(isDisplayed
()));
onView(withId(R.id.action_save_results_patients)).perform(click());
```

Sl. 6.6. Pristup zaslonu rezultata i spremanje podataka pomoću naredbe menija u akcijskoj traci

Korištenjem *contrib* paketa opisanog u potpoglavlju 2.6.1 lagano se otvara navigacijska ladica pomoću metoda prikazanih na slici 6.7. U svrhu dodatnog osiguranja se provjerava je li ladica zatvorena na lijevoj strani zaslona. Potrebno je izabrati opciju izbornika rezultati kako bi se pristupilo rezultatu koji je netom spremljen.

```
pauseTestFor(3000);
onView(withId(R.id.drawer_layout))
        .check(matches(isClosed(Gravity.LEFT)))
        .perform(DrawerActions.open());
onView(withId(R.id.drawer_layout)).check(matches(isDisplayed()));
onView(withId(R.id.nav_view))
        .perform(NavigationViewActions.navigateTo(R.id.nav_Results));
```

Sl. 6.7. Prikaz pravila koje otvara početni zaslon pri pokretanju testa

Metoda koja pristupa adapteru za prikaz rezultata u obliku rešetke (engl. *grid view*) odabire element iz prikaza koji se pronađe po jedinstvenom broju pozicije i prikazuju se detalji toga rezultata. Provjerava se jednakost prikazanog rezultata s rezultatima dobivenim netom nakon unosa parametara. Slikom 6.8. opisan je povratak na prikaz i brisanje jednog rezultata. Ostali testovi su napravljeni na vrlo sličan način uz male varijacije.

```
onData(anything()).inAdapterView(withId(R.id.gvResP)).atPosition(0).perform(click
());
pressBack();
onData(anything()).inAdapterView(withId(R.id.gvResP)).atPosition(1).perform(longC
lick());
onView(withText("Yes")) .inRoot(isDialog())
        .check(matches(isDisplayed()));
onView(withId(android.R.id.button1)).perform(click());
pressBack();
```

Sl. 6.8. Prikaz pravila koje otvara početni zaslon pri pokretanju testa

6.3.3. Problemi i prepreke pri izvođenju testova

Snimač testova pokazao se kao vrlo komplicirano rješenje provođenja testova. Dobra stvar je kako se test može snimiti i pokrenuti, ali u većini slučajeva oni neće uvijek raditi jer se ubacuje puno dodatnih stvari koje nisu potrebne kao što je pauziranje niti na određeno vrijeme. Sintaksa testa je napisana u obliku hvatanja iznimki i kao takva je vrlo nepregledna. Pisanjem vlastitih metoda prema korisničkom sučelju i pokretanje takvih testova je puno preglednije i učinkovitije rješenje, što je i vidljivo na primjeru prijave korisnika (slika 6.9).

```
         onView(withId(R.id.etUsernameR))
                 .perform(typeText(loginEmail));
         onView(withId(R.id.etPass))
                 .perform(typeText(loginPass));
         onView(allOf(withId(R.id.bConfirmLogin),
withText("Login")))
                 .perform(click());
}
```



```
ViewInteraction appCompatButton =
onView(
        allOf(withId(R.id.bLogin),
withText("Login"),
                childAtPosition(

childAtPosition(
withId(android.R.id.content),
                                        0),
                                2),
                isDisplayed()));
appCompatButton.perform(click());

ViewInteraction appCompatEditText3 =
onView(

allOf(withId(R.id.etUsernameR),
                        childAtPosition(
childAtPosition(
withId(android.R.id.content),

0),
                                1),
                        isDisplayed()));
appCompatEditText3.perform(replaceTe
xt("luka.miha@test.com"),
closeSoftKeyboard());
```

```
    ViewInteraction
appCompatEditText4 = onView(
allOf(withId(R.id.etPass),
childAtPosition(
childAtPosition(
withId(android.R.id.content),

0),
                                2),
isDisplayed()));
appCompatEditText4.perform(replace
Text("123456"),
closeSoftKeyboard());

    ViewInteraction
appCompatButton2 = onView(

allOf(withId(R.id.bConfirmLogin),
withText("Login"),
childAtPosition(
childAtPosition(
withId(android.R.id.content),
0),
                                3),
isDisplayed()));
appCompatButton2.perform(click());
}
```

Sl. 6.9. Prikaz testa pomoću Espressa (gore) i snimača testova (dolje)

Prikazivanje izvještaja o pogrešci pri izvođenju testa ponekad ne daje precizno što ne valja u testu i gdje se nalazi problem. Pogreška pokazuje na liniju ili dvije iznad u kojima se uopće ne nalazi nova dodana linija koda, a linija na koju se upućuje je implementirana prije i radi dobro.

Pri unosu parametara laboratorijskih testova prvotno se koristila metoda zamjeni tekst (engl. *replace text*) koja je radila nepouzdano. Metoda zamjeni tekst zamijenjena je metodom unesi tekst (engl. *type text*) koja je radila nakon toga svaki puta. Prilikom pokretanja testova vrlo je važno onemogućiti sve tri vrste animacije mobilnog uređaja koje su redom: skaliranje prozora aplikacije, animacije tranzicije te skaliranje trajanja animacija.

6.3.4. Unaprjeđenje programskog koda

Pogreške su se događale najviše pri identifikacije elemenata po jedinstvenom identifikatoru. Ovakve vrste testova najviše se baziraju na tome da se pronađu svi ti elementi kojima se daju naredbe. Na takav način aplikacija radi upravo one korake koji joj se zadaju. Najveća pogreška se dogodila kod popunjavanja parametara testova. U testu su zadani parametri redom koji se popunjavaju kako su navedeni u sučelju, prvo hormoni, pa antitijela. Propust se dogodio kada se na petom elementu test srušio i dao poruku kako element ne postoji iz razloga što se već pojavio u testu. Pregledavanjem *.xml* datoteke otkrivena je pogreška u jedinstvenim

identifikatorima koji su se negdje ponavljali, duplicirali. Pogreška je vjerojatno napravljena jer su polja za antitijela naknadno dodana u aplikaciju. Nadalje, za prikaz rešetke postojao je resurs s istim jedinstvenim identifikatorom što je onemogućilo otvaranje detalja pojedinog rezultata. Izmijenjeno je da sada svaki korisnik ima svoj vlastiti resurs s jedinstvenim identifikatorom. Izvješće na temelju provedenih testova dano je tablicom 6.2.

Provedeni UI testovi uvelike su pridonijeli stabilnosti aplikacije. Kako je tekla implementacija testova tako se nailazilo i na neke od grešaka. Nakon otklonjene pogreške koja tijekom razvoja nije bila uočljiva moglo je doći do ozbiljnog propusta koji bi utjecao na ključnu značajku ove aplikacije. Aplikacija bi svaki puta prikazivala pogrešan rezultat i zato je provođenje testova važno jer su uvijek dogodi poneka pogreška u kodu koja je u ovom slučaju nije bila ozbiljna u pogledu razvoja, nego njezine same primjene.

Tab. 6.2. Izvješće testiranja

Verzija programske podrške:		1.0	Vrijeme provedeno u kreiranju testa:		06.09. – 10.09.2018.	
R. Br.	**Testni slučaj**		**Specifikacija testnog slučaja**			
1	Prijava i odjava korisnika		Popunjavanje polja s korisničkim podacima, otvaranje navigacijske ladice i odjava korisnika			
2	Unos parametara već oboljelih korisnika i spremanje rezultata u bazu		Unos parametara laboratorijskih mjerenja, spremanje slike, rezultata i preporuka te otvaranje navigacijske ladice i prikaz spremljenih rezultata			
3	Odabir simptoma i spremanje rezultata u bazu		Odabir simptoma korisnika, spremanje slike grafa, rezultata i preporuka te otvaranje navigacijske ladice i prikaz spremljenih rezultata			
4	Otvaranje kamere za promjenu slike profila		Prijava korisnika, otvaranje navigacijske ladice prikaz profila i otvaranje kamere			
Greške dobivene testiranjem			**Početak testiranja:**	19:03		
R.Br.	Testni slučaj		Datum	Greška – opis	Ispravak opis	Verzija
1	Unos parametara testova		06.09.	Greška pri funkciji unosa brojeva	Zamjena funkcije zamjena teksta sa unesi tekst	1.0
1	Unos parametara testova		06.09.	Greška pri unosu parametara, jednaki i ponavljajući identifikatori	Provjera *.xml* resursa i ispravljanje identifikatora	1.0
1	Spremanje slika u galeriju		07.09.	Aplikacija ne prihvaća dopuštenja	Definiranje pravila za pristup resursima	1.0
2	Odabir rezultata u prikazu rešetke		08.09.	Postoji resurs s jednakim identifikatorom za oba tipa korisnika	Stvaranje novog resursa koji se dodjeljuje drugom tipu korisnika	1.0
3	Otvaranje kamere uređaja		09.09.	Aplikacija ne prihvaća dopuštenja	Definiranje pravila za pristup resursima	1.0
Kraj testiranja: 21:24				**Ukupan broj pogrešaka:**	4 (25% prolaznost)	

7. ZAKLJUČAK

U ovom diplomskog radu osmišljena je, modelirana i programski ostvarena mobilna aplikacija koja korisnicima pomaže kod problema bolesti štitnjače.

Postavljeni ciljevi ostvareni su pomoću prikladne programske okoline. Predefinirane vrijednosti pojedinih parametara i rezultati testova omogućili su konačno ostvarenje aplikacije koristeći se analizama, postupcima i vrijednostima koji su također vrlo detaljno razjašnjeni prije programskog rješenja. Aplikacija cilja prema dvije skupine korisnika, a to su: oboljeli korisnici i oni koji sumnjaju na bolest. Implementirane su radnje koje korisniku omogućuju registraciju, prijavu, promjenu slike profila, kao i ažuriranje podataka vezanih uz njegov korisnički račun. Generiranje rezultata i preporuka pri odabiru parametara vrši se na temelju rada s poljem vrijednosti te uvjetnog grananja za obje vrste korisnika. Algoritam uspoređivanja dvaju polja niza znakova daje postotke podudaranja odabranih simptoma s pojedinom bolesti. Uz to, rješenje je potpomognuto brojilom i operacijom dijeljenja ukupnog broja simptoma bolesti s brojem odabranih simptoma. Stablo odlučivanja se koristilo pri generiranju svih preporuka na temelju konstantnih vrijednosti hormona, antitijela, ali i već gotovih preporuka ovisno o ishodu odlučivanja. Rezultate je moguće spremiti u bazu podataka i kasnije ih pregledavati. Obavijesti su kreirane uz pomoć menadžera alarma ugrađenog u sustav i notifikacijskog kanala. U novijim inačicama Androida svaka aplikacija ima svoj kanal za obavijesti što se i ovdje moralo napraviti s prijemnikom emitiranja kojem bi bila poslana notifikacija svaki puta kada bi došlo do određenog postavljenog trenutka u budućnosti, a samim time bio bi obaviješten i korisnik.

Provođenjem automatiziranih testova grafičkog sučelja su utvrđene pogreške koje nisu previše izražene, jer rezultati testova pokazuju razinu funkcionalnih pogrešaka vezanih uz unos parametara i odobravanje dozvole za pristup resursima, što je rezultiralo prolaznošću od samo 25% posto glavnih funkcionalnosti aplikacije pri prvom pokretanju testova.

LITERATURA

[1] Study Tonight, Android Architecture, About Android

https://www.studytonight.com/android/android-architecture (posjećeno: 01.06.2018.)

https://www.studytonight.com/android/introduction-to-android (posjećeno:01.06.2018.)

[2] Android Developer, Android Architecture

https://developer.android.com/guide/platform/ (posjećeno: 01.06.2018.)

[3] Android Developer, Android Studio

https://developer.android.com/studio/features/ (posjećeno: 01.06.2018.)

https://developer.android.com/studio/intro/ (posjećeno: 01.06.2018.)

[4] Oracle, About Java

http://www.oracle.com/technetwork/java/intro-141325.html (posjećeno: 04.06.2018.)

[5] Java T Point, Features of Java

https://www.javatpoint.com/features-of-java (posjećeno: 04.06.2018.)

[6] Android Developers, Room, dostupno na:

https://developer.android.com/training/data-storage/room/ (posjećeno: 02.08.2018.)

https://developer.android.com/topic/libraries/architecture/room (posjećeno:02.08.2018.)

[7] P. Bhuarya, S. Nupur, A. Chatterje, R. S. Thakur, Mobile Application Testing: Tools & Challenges, International Journal of Engineering and Computer Science, Vol. 5, Issue 10, Oct. 2016, pp. 18679-18681 http://ijecs.in/index.php/ijecs (posjećeno: 05.06.2018.)

[8] Software Testing Class, Black box, White Box Testing,

https://www.softwaretestingclass.com/difference-between-black-box-testing-and-white-box-testing/ (posjećeno: 05.06.2018.)

[9] R. Zafar, Code Project, Different Types of Testing, 20 Mar 2012

https://www.codeproject.com/Tips/351122/What-is-software-testing-What-are-the-different-ty (posjećeno: 05.06.2018.)

[10] L. Sharma, Tools QA, Manual Testing, April 9 2016

http://toolsqa.com/software-testing/manual-testing/ (posjećeno: 07.06.2018.)

[11] V. Garousi, M. V. Mäntylä, When and what to automate in software testing? A multi-vocal literature review - Manuskript, Information and Software Technology, Ankara, Turkey, October 2015, pp. 1-35

[12] S. Pittet, Attlassian: Testing types https://www.atlassian.com/continuous-delivery/different-types-of-software-testing (posjećeno: 07.06.2018.)

[13] A. Santos, I. Correira, Mobile Testing in Software Industry using Agile: Challenges and Opportunities, 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), Graz, Austria, 13-17 April 2015, pp. 1-2

[14] Software Testing Help, 5 Mobile Testing Challenges and Solutions, updated June 7 2018 https://www.softwaretestinghelp.com/5-mobile-testing-challenges-and-solutions/ (posjećeno: 07.06.2018.)

[15] B. Kirubakaran, V. Karthikeyani, Challenges and solution approach through automation, 2013 International Conference https://ieeexplore.ieee.org/document/6496451/ (posjećeno: 08.06.2018.)

[16] V. Zakharov, StackOverflow, Statement from Google Employee https://stackoverflow.com/a/20487527 (posjećeno: 08.06.2018.)

[17] Android Developers, Espresso https://developer.android.com/training/testing/espresso/ (posjećeno: 07.09.2018.) https://developer.android.com/training/testing/espresso/basics (posjećeno: 07.09.2018.)

[18] D. M. Milotić, Što je štitnjača i koje su najčešće bolesti štitnjače, Pliva zdravlje https://www.plivazdravlje.hr/aktualno/clanak/26709/Sto-je-stitnjaca-i-koje-su-najcesce-bolesti-stitnjace.html (posjećeno: 01.06.2018.)

[19] B. Brady, Thyroid Gland: Overview, University of Texas Dell Medical School https://www.endocrineweb.com/conditions/thyroid-nodules/thyroid-gland-controls-bodys-metabolism-how-it-works-symptoms-hyperthyroi (posjećeno: 30.05.2018.)

[20] S. L. Lee, M. C. Stoppler, J. R. Balentine, Thyroid Problems, eMedicineHealth, Reviewed on 11/20/2017

https://www.emedicinehealth.com/thyroid_problems/article_em.htm#thyroid_problems_overview (posjećeno: 08.06.2018.)

[21] D. Haarburger, Thyroid disease: thyroid function test and interpretation, Continuing Medical Education, [S.l.], Vol. 30, n. 7, pp. 241-243, jun. 2012 http://www.cmej.org.za/index.php/cmej/article/view/2515/2432 (posjećeno: 02.06.2018.)

[22] A. Maniakas, L. Davies, M. Zafereo, Thyroid Disease Around the World, Otolaryngologic Clicnics of North America, Vol. 51, Isssue 3, June 2018, pp. 631-642

[23] D. Đokić, Kratki pojmovnik bolesti štitnjače, Poliklinika-Aviva https://poliklinika-aviva.hr/zdravisavjeti/kratki-pojmovnik-bolesti-stitnjace/ (posjećeno: 06.06.2018.)

[24] Medline Plus, U.S. National Library of Medicine, Topic last reviewed: 18 April 2016 https://medlineplus.gov/thyroiddiseases.html (posjećeno: 31.05.2018.)

[25] M. P. J. Vanderpump, The epidemiology of thyroid disease, British Medical Bulletin, Oxford Academic, Vol. 99, Issue 1, 1 September 2011, pp. 39–51 https://academic.oup.com/bmb/article/99/1/39/298307 (posjećeno: 12.06.2018.)

[26] M. C. Skarulis, B. C. Stack, Womens Health, March 16, 2018 https://www.womenshealth.gov/a-z-topics/thyroid-disease (posjećeno: 01.07.2018.)

[27] L. J. DeGroot, The Non-Thyroidal Illness Syndrome, The National Center for Biotechnology Information, February 2015 https://www.ncbi.nlm.nih.gov/books/NBK285570/ (posjećeno: 04.06.2018.)

[28] R. Mathur, W. C. Shiel, Hypothyroidism Symptoms, Diet, Natural and Medical Treatments and Tests, Medicine Net, Medically Reviewed on 4/2/2018 https://www.medicinenet.com/hypothyroidism/article.htm#hypothyroidism_underactive_thyroid_definition_and_facts (posjećeno: 16.06.2018.)

[29] J. Ahmed, M. A. R. Soomrani, TDTD: Thyroid Disease Type Diagnostics, 2016 International Conference on Intelligent Systems Engineering (ICISE), Islamabad, Pakistan, 15-17 Jan. 2016, pp. 44 - 50

[30] H. B. Burch, Thyroid Tests, National Institute of Diabetes and Digestive and Kidney Diseases https://www.niddk.nih.gov/health-information/diagnostic-tests/thyroid (posjećeno: 12.08.2018.)

[31] H. Kodaz, I. Babaoglu, H. Iscan, Thyroid Disease Diagnosis Using Artificial Immune Recognition System (AIRS), TURKEY, ICIS '09 Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ICIS 2009, Seoul, Korea, November 24-26, 2009 pp.756-761

[32] S. A. Tuncer, A. Alkan, Segmentation of Thyroid Nodules with K-Means Algorithm on Mobile Devices, 2015 16th IEEE International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 2015 November 19-21, pp. 345 – 348

[33] S. Sapna, Fusion of Big Data and Neural Networks for Predicting Thyroid, 2016 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT), Mysore, Karnataka, India 9-10 Dec. 2016, pp. 243 – 247

[34] N. B. Merchant, F. S. Mirza, Interpretation of Thyroid Function Tests in Hospitalized Patients, Hospital Medicine Clinics, Vol. 4, Issue 2, April 2015, pp. 243-257

[35] M. A. Qureshi, K. Eksioglu, Expert Advice Ensemble for Thyroid Disease Diagnosis, 2017 25th Signal Processing and Communications Applications Conference (SIU), Antalya, Turkey. 15-18 May 2017, pp. 1-4

[36] R. V. Garcia-Mayor, Limitations of current thyroid function tests, Endocrinología, Diabetes y Nutrición (English ed.), Vol. 64, Issue 7, August–September 2017, pp. 404-405

[37] P. Jay, MP Android Chart, Github https://github.com/PhilJay/MPAndroidChart (posjećeno: 19.08.2018.)

[38] Georges Elhomsy, Medscape, Antithyroid Antibody, Dec 04, 2014 https://emedicine.medscape.com/article/2086819-overview (posjećeno: 28.08.2018.)

[39] The British Association of Endocrine and Thyroid Surgeons, British Thyroid Foundation, 2015, http://www.btf-thyroid.org/information/leaflets/34-thyroid-function-tests-guide (posjećeno: 16.08.2018.)

[40] Shomon M, R. N. Fogoros, VeryWellHealth, Understading Your Thyroid Blood Tests and Results, February 08, 2018 https://www.verywellhealth.com/interpret-your-thyroid-test-results-3231840 (posjećeno: 26.08.2018.)

[41] Bumptech, Glide Image Library, Github https://github.com/bumptech/glide (posjećeno: 19.07.2018.)

[42] Hdodenhof, Circle Image View, Github https://github.com/hdodenhof/CircleImageView (posjećeno: 03.08.2018.)

[43] GrenderG, Toasty, Github https://github.com/GrenderG/Toasty (posjećeno: 21.08.2018.)

[44] Flaticon, Freepik Company S. L., Icons https://www.flaticon.com/search?word=info

[45] Stackoverflow, Developers Community https://stackoverflow.com/ (posjećeno: 12.09.2018.)

[46] Nurik R., Android Assets Studio, Github https://romannurik.github.io/AndroidAssetStudio/ (posjećeno: 16.07.2018.)

SAŽETAK

Mobilna aplikacija ostvarena u ovom diplomskom radu služi kao potpora korisnicima koji mogu biti oboljeli ili oni koji sumnjaju na bolesti štitnjače. Mobilna aplikacija prati napredak, generira preporuke i prikazuje rezultate na temelju korisnikova unosa ili odabira podataka. Radnje koje su korisniku dostupne u sklopu korisničkog računa su prijava i registracija te uređivanje profila. Prikaz rezultata odvija se na temelju unosa parametara, tako da se uspoređuju zadnja dva rezultata testova. Odabir grupiranih simptoma uspoređuje se s definiranim poljima niza znakova bolesti što za rezultat daje postotak podudaranja. Postupci se temelje na sortiranju, uspoređivanju polja i uvjetnog grananja, odnosno stabla odlučivanja. Moguće je pristupiti informacijama o bolestima kao i mogućnost kreiranja podsjetnika. Rezultati se mogu pregledati naknadno. Potvrda stabilnosti aplikacije napravljena je pisanjem automatiziranih testova koji upućuju na sitne propuste pri pisanju koda koji bi uvelike utjecali na konačni rezultat.

Ključne riječi: baza podataka, bolesti štitnjače, laboratorijski podaci, mobilna aplikacija, testiranje programske podrške.

ABSTRACT

Mobile application realized in this graduate thesis serves as a support to users who may be affected or those who is suffering from thyroid disease. A mobile application keeps track of progress, generates recommendations and it displays results based on user input or data selection. Actions that are available to users in context of user account are login, register and profile editing. Showing results is based on comparison between two last laboratory tests. Furthermore, that array of picked symptoms is compared with another array of symptoms for every disease. Described comparison as a results gives matching percentage. The procedures are based on sorting, comparing the arrays and the conditional branching, i.e. the decision tree. User can access infomation about diseases and create reminder. Results can be reviewed later. Finally, stability of application is confirmed by writing and running automated tests which point to minor bugs in writing code that would greatly affect the final result.

Key words: database, thyroid diseases, laboratory tests, mobile application, software testing.

ŽIVOTOPIS

Luka Mihačić rođen je 10. travnja1994. godine u Osijeku, Hrvatska. Stanuje u Belišću, na adresi Matije Gupca 2. Godine 2001. započinje osnovnoškolsko obrazovanje u OŠ Ivana Kukuljevića u Belišću te je izrazito aktivan u Košarkaškom Klubu Belišće. Opću gimnaziju u Valpovu završava odličnim uspjehom i na temelju njega 2013. izravnim upisom upisuje preddiplomski sveučilišni studij računarstva na Elektrotehničkom fakultetu u Osijeku. Završava preddiplomski sveučilišni studij računarstva 2016. godine na Fakultetu Elektrotehnike, Računarstva i Informacijskih tehnologija Osijek i stječe titulu prvostupnika inženjera računarstva. U sklopu obvezne prakse počinje raditi u tvrtki UHP Digital 2017. godine u kojoj trenutno radi kao test inženjer. Od ostalih znanja i vještina posjeduje znanje engleskog jezika, principe programiranja, posebice Javu i Android. Posjeduje vozačku dozvolu B kategorije.

PRILOZI

Prilog 1. Dokument rada

Prilog 2. Pdf rada

Prilog 3. Programski kod mobilne aplikacije