Optimizacija Android mobilne aplikacije

Zamaklar, Fran

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

OPTIMIZACIJA ANDROID MOBILNE APLIKACIJE

Diplomski rad

Fran Zamaklar

Osijek, 2023.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 20.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Fran Zamaklar
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1259R, 07.10.2021.
OIB studenta:	73165060444
Mentor:	doc. dr. sc. Ivan Vidović
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Bruno Zorić
Član Povjerenstva 1:	doc. dr. sc. Ivan Vidović
Član Povjerenstva 2:	doc. dr. sc. Krešimir Romić
Naslov diplomskog rada:	Optimizacija Android mobilne aplikacije
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U sklopu ovog diplomskog rada potrebno je proučiti postojeću mobilnu aplikaciju napisanu u Java programskom jeziku te definirati nedostatke i moguća poboljšanja dizajna, arhitekture i implementacije. Nakon proučavanja aplikacije, potrebno je korišenjem proizvoljno odabranog pristupa razvoju programske podrške te Kotlin programskog jezika razviti novu verziju mobilne aplikacije. U praktičnom dijelu je potrebno implementirati određeni skup testova kako bi se osigurala kvaliteta i funkcionalnost mobilne aplikacije. Tema rezervirana za: Fran Zamaklar
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 od strane mentora:	20.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 01.10.2023.

Ime i prezime studenta:	Fran Zamaklar
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1259R, 07.10.2021.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Optimizacija Android mobilne aplikacije**

izrađen pod vodstvom mentora doc. dr. sc. Ivan Vidović

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. TEORIJSKA POZADINA OPTIMIZACIJE, PODRUČJE PROBLEMATIKE I STANJE U PODRUČJU 2

2.1. Optimizacija 2

2.1.1. Definicija 2

2.1.2. Značajke i obilježja 2

2.1.3. Važnost 2

2.1.4. Pojam autonomne optimizacije 3

2.2. Područje problematike 3

2.2.1. Problematika arhitekturnog modela 3

2.2.2. Problematika programske organiziranosti 3

2.2.3. Problematika sposobnosti ispitivanja i održavanja 4

2.3. Prikaz postojećih rješenja i stanje u području optimizacije mobilne Android aplikacije 4

2.3.1. Analiza stanja u području 4

2.3.2. Glavni izazovi 4

2.3.3. Računalna rješenja na postojeće izazove 5

2.3.4. Postojeća programska rješenja 5

2.3.4.1. *The Google Home Application* 6

2.3.4.2. *Discord* 6

2.3.4.3. *Gridle* 7

2.3.4.4. *Uber* 7

2.4. Idejno rješenje poboljšane verzije mobilne aplikacije za vođenje mortalitetne statistike 8

3. MODEL POBOLJŠANE VERZIJE MOBILNE APLIKACIJE ZA VOĐENJE MORTALITENE STATISTIKE 10

3.1. Nužni podaci za vođenje mortalitetne statistike 10

3.2. Postupak vođenja mortalitetne statistike prema računalnom rješenju 10

3.3. Funkcionalni zahtjevi mobilne aplikacije 10

3.3.1. Prijava i odjava 11

3.3.2. Registracija 11

3.3.3. Personalizacija osobnih podataka 11

3.3.4. Unos podataka 11

3.3.5. Pohrana podataka 12

3.3.6. Određivanje lokacije putem *Google Maps* 12

3.3.7. Prikaz i mogućnost ažuriranja pohranjenih podataka 13
3.3.8. Prikaz statističke analize podataka i izvoz podataka 13
3.4. Nefunkcionalni zahtjevi na mobilnu aplikaciju 13
3.4.1. Performanse 13
3.4.2. Sigurnosni zahtjevi 14
3.4.3. Raspoloživost 14
3.4.4. Ostali nefunkcionalni zahtjevi 14
3.5. Dizajn korisničkog sučelja i korisničko iskustvo 15
3.5.1. Elementi korisničkog sučelja 15
3.5.2. Očekivano korisničko iskustvo i pokazatelji korisničkog iskustva 15
3.6. Struktura baze podataka 16
3.7. Struktura mobilne aplikacije prema idejnom rješenju 17
4. PROGRAMSKA IZVEDBA MOBILNE APLIKACIJE 19
4.1. Programske tehnologije, okoline i jezici korišteni za izradu aplikacije 19
4.1.1. FIGMA 19
4.1.2. Operacijski sustav Android 20
4.1.3. *Android Studio* 20
4.1.4. XML 21
4.1.5. *Kotlin* 21
4.1.6. *Firebase* 21
4.1.7. *Google Maps* 22
4.1.8. *Koin* 22
4.1.9. *LiveData* 22
4.2. Programski predložak mobilne arhitekture 22
4.2.1. Predložak mobilne arhitekture MVVM 23
4.3. Programsko rješenje na strani korisnika 24
4.3.1. Unos podataka 24
4.3.2. Pohrana podataka u bazu podataka 26
4.3.3. Uređivanje osobnih podataka i podataka umrle osobe 27
4.4. Programsko rješenje na strani poslužitelja 28
4.4.1. Prijava i odjava 28
4.4.2. Registracija 29
4.4.3. Prikaz rezultata pohranjenih u bazu podataka 31
4.4.4. Prikaz statističke analize rezultata i njihov izvoz u obliku XSLX datoteke 32
4.4.5. Određivanje korisnikove lokacije 33
4.4.6. Povezivanje na bazu podataka 34

4.4.7. Provjera postojanja internetske povezanosti 34
5. KORIŠTENJE I ISPITIVANJE MOBILNE APLIKACIJE 36
5.1. Način korištenja mobilne aplikacije 36
5.2. Regresijsko ispitivanje mobilne aplikacije 37
5.3. Skupna analiza rada mobilne aplikacije 37
6. ZAKLJUČAK 39
LITERATURA 40
SAŽETAK 44
ABSTRACT 45
ŽIVOTOPIS 46
PRILOZI 47

1. UVOD

Svaka pojava u svome postojanju teži što boljoj vlastitoj verziji. Nekada je taj proces neminovan u smislu nužnih promjena i poboljšanja sustava, a nekada je to posljedica različitih vanjskih čimbenika poput trendova tržišta, potreba klijenata i nadmetanja s konkurencijom. Tako taj proces ne zaobilazi ni tehnologiju i tehnološke alate koji svakodnevno olakšavaju ljudski život. Tehnologija je svakodnevno podložna poboljšanju, a nadograđuje se sukladno novim životnim stilovima i ljudskim potrebama.

Cilj ovog diplomskog rada je optimizirati postojeće programsko rješenje – mobilnu *Android* aplikaciju za vođenje mortalitetne statistike. Optimizacija omogućava kvalitetniju strukturu mobilne aplikacije kroz razvoj u novoj programskoj tehnologiji sa suvremenim programskim predloškom za mobilnu aplikaciju. Također se optimizira mogućnost ispitivanja mobilne aplikacije kao nužnog nefunkcionalnog zahtjeva te se uvode novi funkcionalni zahtjevi poput prijave, odjave te registracije korisnika.

U drugom poglavlju definirani su područje djelovanja i svrha pojma optimizacije, ostvaren je prikaz postojećih ostvarenih rješenja prema principu optimizacije te se objašnjava problematika trenutnog rješenja. U trećem poglavlju opisan je model poboljšane verzije mobilne aplikacije za vođenje mortalitetne statistike te su detaljno prikazani svi bitni ulazni i izlazni parametri mobilne aplikacije, struktura baze podataka, funkcionalni i nefunkcionalni zahtjevi na aplikaciju te dizajn i struktura aplikacije preko dijagrama za modeliranje programske podrške. Četvrto poglavlje predstavlja programsku izvedbu poboljšane verzije mobilne aplikacije. Rad se zaključuje regresijskim ispitivanjem preko kojeg se evaluira rad poboljšane verzije mobilne aplikacije, prezentira način korištenja aplikacije te sumira cjelokupna izvedba programske podrške.

2. TEORIJSKA POZADINA OPTIMIZACIJE, PODRUČJE PROBLEMATIKE I STANJE U PODRUČJU

2.1. Optimizacija

U ovom poglavlju objašnjena je definicija optimizacije u području problematike, njezina obilježja, značajke i važnost, te su predstavljena prijašnja ostvarena rješenja u samom području.

2.1.1. Definicija

Optimizacija se može razmatrati kao temelj mnogih područja kao što su: primijenjena matematika, računarstvo, inženjerstvo i mnoge druge znanstvene discipline. Između ostalog, optimizacija ima ključnu ulogu u pronalasku mogućih životnih problema, od algoritmičkog programiranja do procesa istraživanja u područjima ekonomije, upravljanja znanjem, medicine, umjetne inteligencije i drugih disciplina. Njezino uporište ostvaruje se u pronalasku najboljeg rješenja kao uspješne istraživačke aktivnosti te je zaslužna za provođenje teorijskih i praktičnih razvoja novih tehnologija i metoda koje ostvaruju najučinkovitiji dizajn različitih sustava uz najmanje troškove procesa, a uz najveće prihode [1].

2.1.2. Značajke i obilježja

Tehnike optimizacije odvijaju se u tri smjera: kvantifikacija, oduzimanje podatkovnog sadržaja (dekontekstualizacija) i formulacija. Prvobitno se pretpostavlja da postoje statistički podaci, odnosno da bi trebali postojati statistički podaci s kojima se utvrđuje opseg rješenja problema – kvantifikacija. Zatim, procesom oduzimanja podatkovnog sadržaja podaci se pretvaraju u informaciju te se sumiraju stvarne prakse u opći sustav djelovanja. Najbolji primjer ovakvog pristupa su pokusi – metodologije s najboljom težnjom kako preusmjeriti problematičnu situaciju prema rješenju. Posljedično dolazi do formulacije, odnosno postavljanja novih, optimalnih kalkulacija i spoznaja o rješenju u novi kontekst. Međutim, čak i ako postoji optimalno stanje sustava, uvijek mora postojati veza između novih kalkulacija i postojanja, odnosno iterativne aktualizacije sustava prema vanjskom svijetu [2].

2.1.3. Važnost

Važnost optimizacije leži u njezinoj povezanosti s ljudskim razvojem. Ona se oslanja na dugoročne znanstvene spoznaje koje su usmjerene na društveni i tehnološki razvoj kroz inovacije i njihovo usavršavanje. Stoga takva veza nosi određene značajke – cilj koji održava vrijednosti razvoja te potrebu koja je prirodni produkt razvoja i ljudskog djelovanja [3].

2.1.4. Pojam autonomne optimizacije

Autonomni proces optimizacije (samooptimizacija) označava razvoj velikog opsega procesa s unaprijed definiranim parametrima, bez ljudske intervencije, a koji vlastitim iterativnim učenjem na temelju izlaznih parametara, poboljšavaju odzivne karakteristike sustava. Određivanje skupa značajnih i objektivnih ulaznih parametara procesa označava najkritičniji dio uspješne optimizacije. Uspješna optimizacija se također oslanja na identifikaciju prikladne autonomne opreme koja može učinkovito izvesti procesne pokuse te dati ispravnu analizu [4].

2.2. Područje problematike

2.2.1. Problematika arhitekturnog modela

Trenutno ostvareno rješenje mobilne aplikacije za vođenje mortalitetne statistike napravljeno je prema predlošku arhitekturnog modela MVC (engl. *Model-View-Controller*). Koncept koristi tri komponente: model (engl. *model*), pogled (engl. *view*) i upravljač (engl. *controller*) kako bi se upravljalo odgovornostima koje stvaraju korisničko sučelje. Model sadrži podatke, odnosno opisni sadržaj koji će se prikazivati na pogledu kojim upravlja kontroler te koji obavlja svu logiku potrebnu za rad s modelom i pogledom.

Rješenje mobilne aplikacije ostvareno je na dva fragmenta koji služe kao pogledi za unos podataka. Nakon što korisnik unese podatke na prvom pogledu, kontroler odrađuje logiku spremanja tih podataka u bazu podataka, pogled (u ovom slučaju kontekst) se postavlja na drugi fragment te se model priprema za unos preostalih podataka. Budući da je namijenjeno da korisnik odmah popunjava drugu stranicu (fragment) podataka, nakon što je popunjena prva stranica, osigurano je da se preostali nužni podaci ispravno spoje s prijašnjim podacima preko svojstvenog identifikatora u bazi podataka.

Ovakav pristup uzrokuje nekoliko nepotrebnih koraka koji se mogu unificirati, a to su: dohvat prvih dijelova podataka iz baze podataka, provjera preko svojstvenog identifikatora te konačno spremanje cjelovitog objekta u bazu. Također, posljedično se mogu pojaviti i potencijalne programske anomalije od kojih je najveća – gubitak veze između prvih i drugih dijelova podataka.

2.2.2. Problematika programske organiziranosti

Programska organiziranosti (engl. *source tree composition*) definira se kao organiziranost svih datoteka, direktorija i ponovno iskorištenih komponenti i znanja. Prednosti koje dolaze ovakvim pristupom označavaju da postoji jedna izvršiteljska pretvorba koda (engl. *build*) i konfiguracijskih procesa što se naposljetku može prevesti kao jedinstvena jedinica sustava. Takav pristup ogleda

se u funkcijskoj i sustavnoj dekompoziciji čime se stvaraju novi podsustavi i jedinice koji su povezani na semantičkoj razini, a imaju mogućnost ponovne upotrebe [5].

Trenutno rješenje programske podrške moguće je unaprijediti prema konceptima funkcijske i sustavne dekompozicije, u svrhu boljeg vizualnog, ali i semantičkog razumijevanja programske organiziranosti.

2.2.3. Problematika sposobnosti ispitivanja i održavanja

Sposobnost ispitivanja (engl. *testability*) i održavanja (engl. *maintability*) iznimno su bitan dio razvoja te se definiraju kao nefunkcionalni zahtjevi koji označavaju robusnost i kvalitetu programske podrške. Ova dva atributa usko su povezana s prijašnje dvije navedene problematike, a podložni su velikim oscilacijama u performansama ovisno o izvedbi arhitekturnog predloška i programske organiziranosti. Prema istraživanju [6], ishod usporedbe ispitivanja linija koda u klasama kod različitih arhitekturnih predložaka (MVC i MVVM (engl. *Model-View-ViewModel*)) označio je slabu mogućnost ispitivanja arhitekturnog predloška MVC. Također, na temelju različitih metrika poput: broja usko-povezanih klasa, količine izvršenog koda, dubine nasljeđivanja, indeksa sposobnosti održavanja te ciklomatske složenosti; bolju izvedbu imaju programske podrške koje su razvijene prema MVVM predlošku.

Shodno gore navedenim zaključcima, teži se ostvariti poboljšana programska podrška u smislu bolje mogućnosti ispitivanja i održavanja.

2.3. Prikaz postojećih rješenja i stanje u području optimizacije mobilne Android aplikacije

2.3.1. Analiza stanja u području

Prema gore navedenoj analizi, vidljivo je da je glavni temelj problematike arhitekturni predložak koji je neprilagođen izvedbama mobilne aplikacije. Premda je mobilna aplikacija funkcionalna i zadovoljava sve definirane funkcionalne zahtjeve, tu činjenicu nadvladava slabija sposobnost za daljnje proširivanje i nadogradnju u smislu funkcionalnih i nefunkcionalnih zahtjeva, a što je posljedica trome i usko povezane strukture mobilne aplikacije.

2.3.2. Glavni izazovi

Glavni izazovi optimizacije mogu se interpretirati kao: preuranjena konvergentnost (engl. *premature convergence*), neravnomjernost (engl. *ruggedness*), krivo navođenje (engl. *deceptiveness*), pristranost (engl. *neutrality*) te optimizacijska epistaza (engl. *epistasis*).

Pod preuranjenom konvergentnošću podrazumijeva se činjenica da je optimizacijski proces dosegnuo svoj lokalni optimum koji više ne može napredovati u prostoru u kojem se trenutno nalazi i promatra. Nasuprot toga, moguće je da postoji područje optimuma koje sadrži bolje i kvalitetnije rješenje. Mogući razlozi nastajanja ovog problema su: manjak raznovrsnosti rješenja te kontraproduktivne mjere koje vode optimizaciju u krivom smjeru.

Zatim, problem neravnomjernosti odnosi se na nepravilno doziranje u optimizacijskim procesima koji su lako promjenjivi pa je na taj način teško pronaći smjer za ispravnu optimizaciju. Uzrok ovakvom ponašanju moguća je slabija kauzalnost, odnosno manjak povezanosti doziranja parametra s promjenama u procesu optimizacije.

Krivo navođenje predstavlja problem neispravnog odabira optimuma na temelju sakupljenih informacija. Shodno tome, optimizacija može dodatno pogoršati stanje sustava što je kontraproduktivno.

Pristranost u optimizaciji temelji se na subjektivnom dojmu da je odabrani pristup optimizaciji najkvalitetniji od mogućih rješenja. Najčešće se ova problematika uspoređuje s mogućnošću daljnjeg razvoja kako bi se neutralizirala i objektivizirala.

Naposljetku, optimizacijska epistaza ima snažan utjecaj na mnoge prijašnje izazove budući da označava utjecaj ovisnosti određenih modula prema ostalim modulima na način da njihovim otkazivanjem ili promjenom, ovisni moduli mijenjaju svoje ponašanje. Stoga će kauzalnost oslabiti, a neravnomjernost porasti [7].

2.3.3. Računalna rješenja na postojeće izazove

Rješenja za navedene postojeće izazove svode se na kvalitetno poznavanje sustava koji se poboljšava i nadograđuje. Bitno je poznavati koje su mu mane i prednosti kako bi se svrsishodno moglo pristupiti rješavanju izazova, bilo da je riječ o problemu nepravilnog doziranja ili krivog navođenja. Iz predstavljenog se može zaključiti da računalna analiza prethodi kvalitetno ostvarenoj optimizaciji, neovisno radi li se o sustavima na poslužiteljskoj i klijentskoj strani ili se radi o korisničkom sučelju.

2.3.4. Postojeća programska rješenja

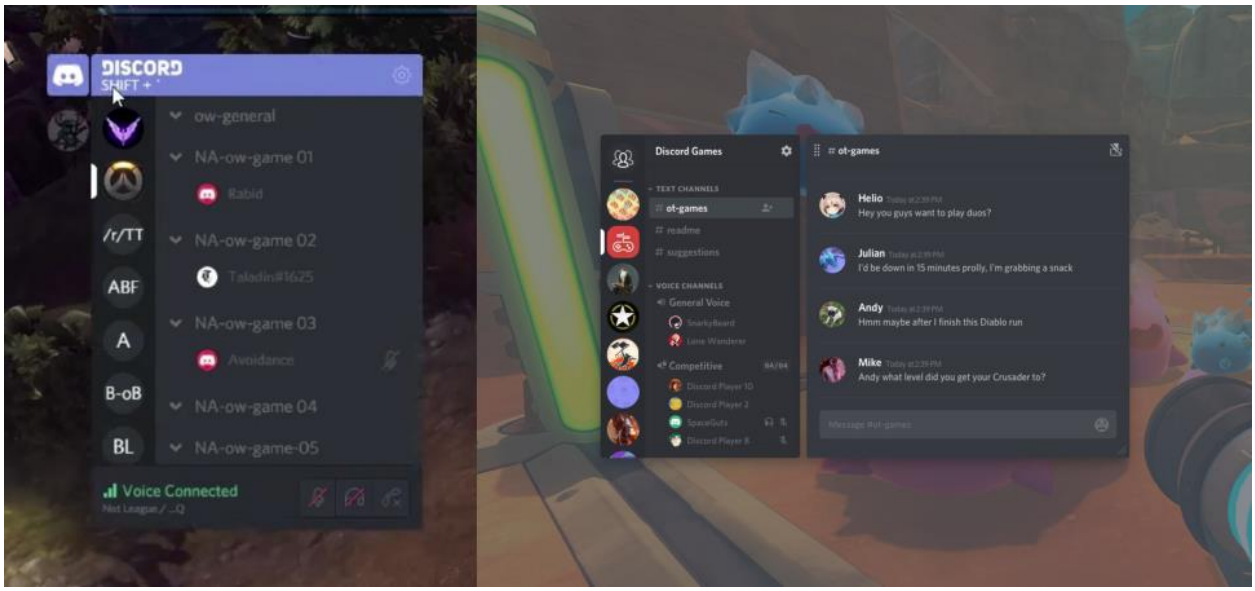
Današnja programska rješenja svakodnevno zahtijevaju određene načine optimizacije ovisno radi li se o poboljšanju performansi ili uvođenju novih funkcionalnosti koje su u koraku sa suvremenim tehnologijama i zahtjevima korisnika.

2.3.4.1. *The Google Home Application*

Google-ova mobilna aplikacija za upravljanje pametnim kućnim uređajima predstavljena je javnosti kao aplikacija koja primarno pruža svakodnevnu pomoć u upravljanju pametnim *Google* uređajima. Prema tome, aplikacija i njezino sučelje upravljanja mora biti, prije svega, izuzetno intuitivno. Međutim, povratnim informacijama korisnika o otežanom korištenju aplikacije, *Google* se odlučio za redizajn aplikacije i tako u svibnju 2022. godine predstavio novu verziju *The Google Home Application*. Uz redizajn korisničkog sučelja, optimizirane su funkcionalnosti upravljanja kamerom poput praćenja videozapisa uživo i pregleda povijesti snimljenog sadržaja. Ujedno su poboljšane određene automatizacijske mogućnosti kao što je mogućnosti pisanja vlastitog skriptnog automatizacijskog programskog rješenja [8].

2.3.4.2. *Discord*

Discord je multiplatformna aplikacija za ostvarivanje udaljene komunikacije (engl. *remote communication*). Aplikacija primarno radi kao pozadinski servis koji omogućuje korisniku komunikaciju pri radu s drugim aplikacijama. Potencijalni problem ovakvog servisa je činjenica da je moguće da prilikom rada u drugoj aplikaciji, koja zahtjeva takozvani *hard shutdown*, dođe *Discord* obavijest. Na slici 2.1., moguće je primijetiti kako prvobitni preklapajući prozor (engl. *overlay window*) korisniku omogućuje interakciju s *Discordom* unutar druge aplikacije, no nije omogućavao mogućnost direktnog odgovaranja na obavijesti. Korisnik bi morao izaći iz trenutne aplikacije na prisilni način (potencijalno izgubiti trenutno stanje aplikacije) te otvoriti dijaloški okvir kako bi odgovorio na poruku. Potaknuti ovakvim problemom, dizajnerski tim razvio je novi dizajn i optimizirao aplikaciju dodajući u preklapajući prozor dijaloški okvir kada korisniku dođe poruka [9].



Slika 2.1. Usporedba *Discord* dizajna preklapajućeg prozora za komunikaciju.

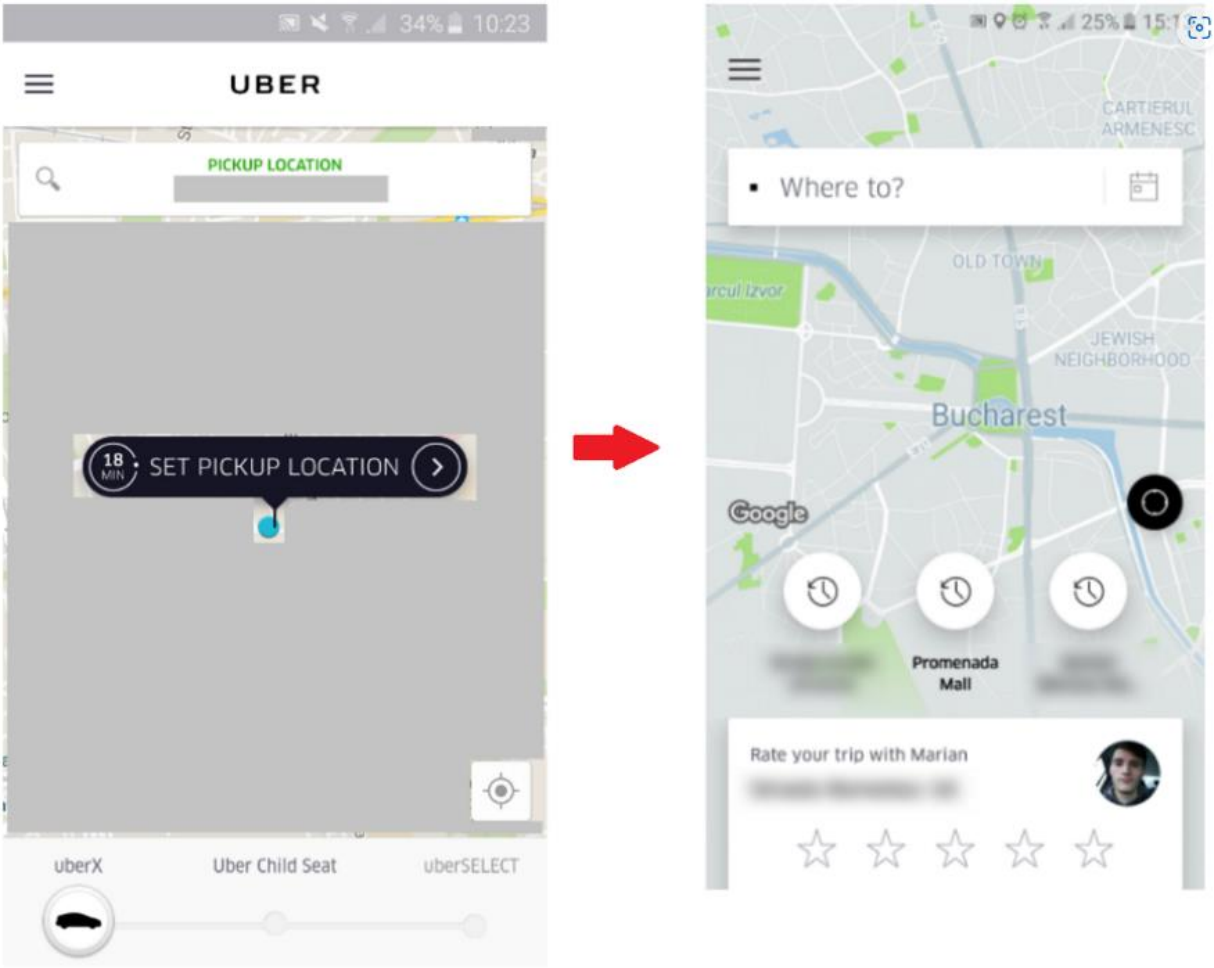
2.3.4.3. *Gridle*

Gridle je upravljačka platforma, stvorena 2013. godine koja poslodavcima omogućuje i olakšava investiranje i različite postupke u poslovnom svijetu poput: stvaranja ponude i potražnje, knjiženje računa te njihovo dokumentiranje. Platforma podržava tehnološki koncept upravljanja odnosa s korisnicima (engl. *customer relationship management*) kojim se upravljaju cjelokupni odnosi poduzeća s vanjskim sudionicima, pri tome se naglasak stavlja na potrošače i potencijalne potrošače. Glavni cilj platforme *Gridle* usredotočen je na pomoć malim poduzećima i obrtima pri njihovom rastu i razvoju. Budući da se tehnologije mijenjaju izuzetno brzo, projektni tim se 2020. godine odlučio na redizajn i optimizaciju ove SAAS platforme (engl. *Software As a Service*) kako bi korisnicima pružili što kvalitetnije korisničko iskustvo. Optimizacija i redizajn su se ostvarili na razinama: personalizacije, nenametljivih obavijesti i podsjetnika te intuitivnim uputama za korištenje proizvoda i pohrane podataka. Jedna od glavnih problematika bila je način unificiranja podataka iz različitih servisa (engl. *Cloud service*) u samu platformu te se stvaranjem vlastitog prostora platforme za pohranu podataka osiguralo da korisnici ne moraju pretraživati podatke po različitim servisima, već da im je sve na jednom mjestu [10].

2.3.4.4. *Uber*

Uber je multimedijska platforma koja pruža različite usluge poput: naručivanja usluge prijevoza, dostavljanja hrane, obavljanja službe morskog transporta te iznajmljivanja električnih bicikala i romobila [11]. Kroz godine postojanja platforma je postupno proširivala svoje usluge te su se shodno tim promjenama ostvarili novi zahtjevi za optimizaciju i redizajn programske podrške.

Tako je jedan od primjera problematike s kojom se platforma suočila bio problem prilagodbe korisničkog sučelja veličini zaslona. Prilikom odabira različitih opcija, koji su prikazani na zaslonu, korisnici bi znali odabrati pogrešni gumb usluge zbog zbijenosti pogleda na korisničkom sučelju. Rješenje se ostvarilo pojednostavljenjem kronologije koraka pristupa samoj usluzi. Prvobitno bi se postavio upit korisniku o njegovom odredištu te bi se zatim prikazale informacije o cijeni usluge, ruti i ostalim informacijama [12]. Na slici 2.2. moguće je vidjeti usporedbu prijašnjeg i novostvorenog rješenja.



Slika 2.2. Usporedba prijašnjeg i optimiziranog rješenja mobilne aplikacije *Uber* [13].

2.4. Idejno rješenje poboljšane verzije mobilne aplikacije za vođenje mortalitetne statistike

Optimizacija mobilne aplikacije za vođenje mortalitetne statistike primarno se ostvaruje radi poboljšanja rada mobilne aplikacije. Rad opisuje cjelokupni redizajn modela aplikacije uz postojeće i dodatne funkcionalnosti (prijava, registracija i personalizacija). Ujedno se ostvaruje regresijsko ispitivanje mobilne aplikacije kako bi se ispitnim slučajevima utvrdilo ispravni i

kvalitetni rad funkcionalnih i nefunkcionalnih zahtjeva na programsku podršku. Poglavlje 3. detaljnije objašnjava prijedlog poboljšane verzije mobilne aplikacije za vođenje mortalitetne statistike, kao i opseg njezinih mogućnosti.

3. MODEL POBOLJŠANE VERZIJE MOBILNE APLIKACIJE ZA VOĐENJE MORTALITENE STATISTIKE

3.1. Nužni podaci za vođenje mortalitetne statistike

Prema Potvrdi o smrti, koja je službeni obrazac evidentiranja preminule osobe, nužni podaci za evidentiranje sastoje se od dva dijela: uvodni dio i izvješće o uzroku smrti. Uvodni dio, odnosno opći podaci odnose se na identifikaciju osobe. Pod navedeno se ubrajaju: osobni identifikacijski broj, ime, prezime, spol i datum rođenja osobe. Zatim se u izvješću o uzroku smrti postavljaju podaci o smrti osobe, a to su: datum, vrijeme, uzrok i mjesto smrti. Preko predviđenih tekstualnih sučelja za unos teksta, korisnik upisuje ime, prezime i mjesto smrti osobe dok se datum, vrijeme i uzrok smrti, te datum rođenja unosi preko padajućih izbornika i kalendara.

Također, nužno je pohraniti podatke za prijavu, odnosno registraciju mrtvozornika. To su sljedeći podaci: ime, prezime, županija djelovanja, adresa e-pošte i lozinka. Podaci o mrtvozorniku pohranjuju se preko svojstvenog sučelja. Korisnik ima mogućnost uređivanja podataka o vlastitoj e-pošti i lozinki.

Izgled dokumenta Potvrde o smrti [14] može se vidjeti na obrascu u PDF obliku na stranici 16.

3.2. Postupak vođenja mortalitetne statistike prema računalnom rješenju

Postupak vođenja mortalitetne statistike prema poboljšanom računalnom rješenju sastoji se od evidentiranja korisnika koji obavlja službu mrtvozorništva, njegovog unosa i tromjesečnog praćenja podataka o mortalitetu područja u kojem obavlja službu te naposljetku podnošenja statističkog izvješća o mortalitetu. Prije podnošenja izvješća bitno je obaviti detaljan pregled unesenih podataka kako bi se osigurali mjerodavni podaci za analizu mortalitetne statistike. Uz mogućnost pregleda podataka u bazi podataka, korisniku se omogućuje izvoz podataka u XSLX obliku (*Excel*) te grafički prikaz podataka.

3.3. Funkcionalni zahtjevi mobilne aplikacije

Funkcionalni zahtjev je definicija zahtjeva, odnosno onoga što se od sustava traži da obavlja i onoga što ne obavlja. Često se izražava prema izrazu: „ako je traženi uvjet ispunjen, prema tome sustav treba odgovoriti“. Jesu li ispunjeni traženi funkcionalni zahtjeve provjerava funkcionalna verifikacija [15]. Sljedećim se poglavljima detaljno opisuje funkcionalni zahtjevi poboljšane verzije mobilne aplikacije za vođenje mortalitetne statistike.

3.3.1. Prijava i odjava

Mogućnosti prijave i odjave djelatnika službe mrtvozorništva ostvaruju se putem poslužiteljskog Google servisa – *Firebase Authentication*. Usluga omogućuje autentifikaciju korisnika putem korisničkog e-mail-a i lozinke, odjavljivanje korisnika te upravljanje korisničkim podacima u slučaju zaboravljene lozinke.

Preko jedinstvenog korisničkog sučelja za prijavu, korisnik unosi vlastitu lozinku i e-mail adresu. U slučaju zaboravljene lozinke, korisnik ima mogućnost obnove lozinke putem svoje elektroničke pošte dok je mogućnost odjavljivanja omogućena jedino prijavljenim korisnicima aplikacije.

3.3.2. Registracija

Funkcionalnost registracije mrtvozornika također se odvija putem instanci SDK (engl. *Software Development Kit*) usluge *Firebase Authentication*.

Sučelje omogućuje korisniku unos vlastitih podataka poput: e-pošte, lozinke, imena i prezimena te mjesta djelovanja službe (županija). Nakon unosa navedenih podataka, korisnik mora prihvatiti privolu o povjerljivosti obrade podataka kako bi pristupio aplikaciji.

3.3.3. Personalizacija osobnih podataka

Korisnik mobilne aplikacije ima mogućnost ažuriranja, odnosno uređivanja vlastitih osobnih podataka. Navedeno se primarno odnosi na njegove identifikacijske podatke za prijavu u aplikaciju (e-mail i lozinka), ali i ime i prezime.

3.3.4. Unos podataka

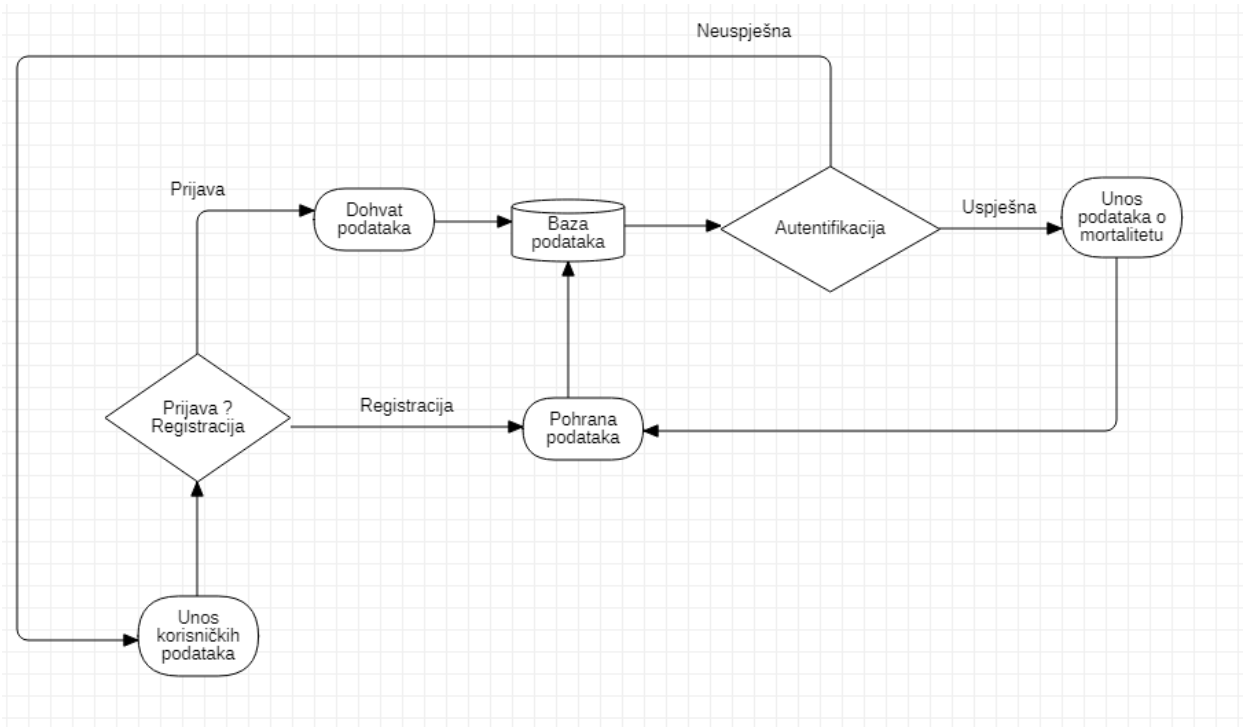
U sklopu mobilne aplikacije, razlikuju se dvije vrste unosa podataka za svrhu koju su namijenjeni. Prva vrsta su podaci za registraciju i prijavu korisnika, a drugi podaci su podaci vezani uz službu obavljanja djelatnosti mrtvozorništva.

Obje vrste podataka unose se preko svojstvenih pogleda čiji je izgled ubrizgan putem *Kotlin* programskog jezika, a oblikovao se označnim jezikom XML. Neki od pogleda su: *EditText, Spinner, TimePicker, RadioButton, ImageView* i ostali važni pogledi. Određeni podaci poput: lozinke i e-mail korisnika imaju svojstvena oblikovanja nad njihovim pogledima koji služe za njihov unos. Tako pogled za unos lozinke ima prikriveni tekstualni unos, a pogled za unos e-mail-a prilagođava korisnikovu tipkovnicu za unos e-mail-a.

3.3.5. Pohrana podataka

Uneseni se podaci pohranjuju u bazu podataka *Firestore* koja se nalazi na internetu. Baza podataka se ažurira u stvarnom vremenu, a omogućuje i izvanmrežni način rada ako aplikacija ostane bez pristupa internetu.

Proces pohranjivanja podataka odvija se za dva odvojena entiteta: podaci o mrtvozorniku i podaci vezani uz službu obavljanja mortalitetne statistike. Podaci o mrtvozorniku su nužni za pristup aplikaciji gdje onda korisnik može unositi podatke vezane za službu obavljanja mrtvozorništva. Također, mogućnost pohrane ažuriranih podataka nad korisnikovim podacima ili nad podacima vezanih za mortalitetnu statistiku uvjetovani su prvobitnim postojanjem tih istih podataka u bazi podataka. Slika 3.1. ilustrativno prikazuje cjelokupni opisani proces.



Slika 3.1. Dijagram tijeka pohrane podataka u bazu podataka.

3.3.6. Određivanje lokacije putem *Google Maps*

Funkcionalnost određivanja korisnikove lokacije ostvaruje se putem poziva *Google Maps* API (engl. *Application Programming Interface*). Uz implementaciju dodatnih dopuštenja koja omogućuju kvalitetniji prikaz mape, korisniku se omogućuje precizno lociranje njegovog položaja. Način i koraci stvaranja mape i lociranja su detaljnije objašnjeni u poglavlju 4.3.4.

3.3.7. Prikaz i mogućnost ažuriranja pohranjenih podataka

Korisničko iskustvo izrazito je bitan dio prilikom evaluacije kvalitete mobilne aplikacije. Iz tog razloga mogućnost pregleda i ažuriranja pohranjenih podataka nužne su funkcionalnosti koje pomažu korisniku u slučaju pogrešaka prilikom unosa podataka. Podaci se dohvaćaju iz baze podataka i prikazuju se u obliku liste na zasebnoj aktivnosti aplikacije. Ovo se ostvaruje unutar pogleda *RecycleView*.

3.3.8. Prikaz statističke analize podataka i izvoz podataka

Jedna od glavnih komponenti koje poboljšavaju korisničko iskustvo mobilne aplikacije za vođenje mortalitetne statistike je statistički prikaz shodno odabranim parametrima. Prema tome se implementira grafički statistički prikaz pohranjenih podataka. Metoda koja se koristi za obradu podataka je metoda uvećanja koja grupira podatke prema željenom parametru, odnosno prema uvjetu ispunjenja inkrementiranja. Na taj način se grafovi popunjavaju podacima koji se zatim prikazuju uz njihove pripadajuće legende i vrijednosti.

3.4. Nefunkcionalni zahtjevi na mobilnu aplikaciju

Nefunkcionalni zahtjevi na mobilnu aplikaciju opisuju kakve će performanse pružiti sustav programske podrške pri izvedbi funkcionalnih zadataka. Ono opisuje skup kvalitetnih značajki koje se očekuju pri korištenju programske podrške (sigurnosni zahtjevi (engl. *security*), raspoloživost (engl. *availability*), pouzdanost (engl. *reliability*), performanse (engl. *performance*), skalabilnost (engl. *scalability*), podudarnost (engl. *compatibility*), prenosivost (engl. *adaptibility*) i drugo) [16].

3.4.1. Performanse

Izvedba aplikacije, odnosno njezine karakteristike prema brzini, aktivnosti i zauzeću radne memorije mogu se odrediti prema vremenu odziva. Vrijeme odziva definira se kao interval između završetka slanja zahtjeva i početka posljedičnog odziva sustava. Zatim, bitno je kvalitetno rasporediti upotrebu resursa, odnosno vremenski interval kada su procesor, ulazno-izlazne jedinice i memorija zauzeti posluživanjem traženih usluga. Naposljetku, bitno je urediti frekvenciju prema kojoj su zahtjevi posluženi od strane sustava. To se često definira kao broj transakcija ili zahtjeva u jedinici vremena [17].

Uređivanjem i kontroliranjem navedenih karakteristika performansi, aplikacija će raditi optimalno.

3.4.2. Sigurnosni zahtjevi

Primarni sigurnosni principi odnose se na sigurnosne postavke i mehanizme, zaštitu od napada, neovlaštenih upada te oporavka rada sustava. Značajnije je shvatiti koji dio sustava bi trebao biti siguran i zašto, nego razmišljati kako zaštiti sustav. Ključne aktivnosti za sigurnosne zahtjeve su: uređivanje sigurnosnih resursa i njihova klasifikacija prema tome tko može njima upravljati [17].

Sigurnosni zahtjevi ostvaruju se u sklopu implementacije *Firebase Authentication* SDK koji imaju sigurnosne postavke i mehanizme za zaštitu od napada i neovlaštenih upada u sustav ili rukovanja podacima.

3.4.3. Raspoloživost

Raspoloživost je vjerojatnost da će sustav raditi i biti raspoloživ korisniku u određenom vremenskom trenutku. Određuju ju srednje vrijeme popravka, planirano vrijeme ispada, neplanirano vrijeme ispada, vrijeme oporavka i sigurnosna pohrana. Za raspoloživost su važne sljedeće aktivnosti: optimizacija i klasifikacija servisa sustava prema njihovoj značajnosti te definiranje hijerarhije servisa tako da se zna koji podređeni servisi moraju biti raspoloživi da bi radio nadređeni servis [17].

3.4.4. Ostali nefunkcionalni zahtjevi

Pored opisanih nefunkcionalnih zahtjeva, bitno je spomenuti i sljedeće nefunkcionalne zahtjeve: pouzdanost, mogućnost korištenja i ispitivanja sustava.

Pouzdanost programske podrške odnosi se na vjerojatnost da programska podrška neće uzrokovati kvar sustava u određenom periodu, pod određenim uvjetima rada [18]. Navedeno označava da sustav koji pruža uslugu mora biti osiguran da će pružati tu uslugu u određenom vremenskom razdoblju. Ovo se može odnositi na pouzdanost tehnologije koja se trenutno koristi i hoće li se ona nastaviti održavati u budućnosti. Što manja sposobnost održavanja, manja je i pouzdanost sustava.

Mogućnost korištenja mobilne aplikacije također je jedan od bitnih nefunkcionalnih zahtjeva. Prema ISO 9241-11, mogućnost korištenja se definira kao razina do koje se proizvod može upotrijebiti za određeno korištenje da bi se postigli ciljevi s učinkovitošću, djelotvornošću i zadovoljstvom [19]. Mogućnosti kvalitetnog korištenja doprinose mogućnosti brzog korisničkog učenja o sustavu i brzog oporavka od pogrešaka prilikom učenja. Tim se sustav ocjenjuje kao intuitivan.

Prema [20], ispitivanje programske podrške je skup aktivnosti s konkretnim ispitnim alatima i tehnikama koji se izvode nad mobilnim aplikacijama kako bi se potvrdila kvaliteta u funkcioniranju, izvedbi, usluge, prenosivosti, korištenju, sigurnosti, umreženosti i drugim bitnim stavkama. Iz tog razloga mogućnost ispitivanja kao nefunkcionalni zahtjev programske podrške iznimno je važna stavka koja povezuje i verificira različite ostale nefunkcionalne, ali i funkcionalne zahtjeve.

3.5. Dizajn korisničkog sučelja i korisničko iskustvo

Dizajn korisničkog sučelja opisuje iterativni skup koraka koji dovode do uspješne implementacije interaktivnog alata, dok dizajn korisničkog iskustva opisuje iterativni skup koraka koji vode k uspješnom rezultatu s interaktivnim, produktivnim i zadovoljavajućim procesom. Često se iz toga razloga stavlja naglasak na važnost korisničkog iskustva pri dizajniranju radi ostvarivanja općeg iskustva, a ne samog sučelja [21]. Primarno se treba osloniti na važnost prikaza informacija u sučelju koje upućuje na to da uputstva, koncepti i signalizacije formiraju rezultate za stvaranje vizualnog dizajna. Na taj način se može stvoriti intuitivan i suvremen dizajn koji pruža kvalitetno korisničko iskustvo upotrebom aplikacije.

3.5.1. Elementi korisničkog sučelja

Korisničko sučelje optimizirane verzije mobilne aplikacije za vođenje mortalitetne statistike sadrži dva konceptualno odvojena, ali semantički povezana dijela aplikacije. Prvi dio se sastoji od prikaza zaslona prijave ili registracije koji se, shodno korisnikovim zahtjevima, prilagođavaju i mijenjaju. Drugi dio ostvaruje se nakon prijave, odnosno registracije. Ono predstavlja središnji dio mobilne aplikacije gdje se odvija glavna logika cjelokupnog sustava. Ujedno se omogućuje pogled s pregledom unesenih podataka, mapa za lociranje korisnika te statistički prikaz podataka i profila korisnika.

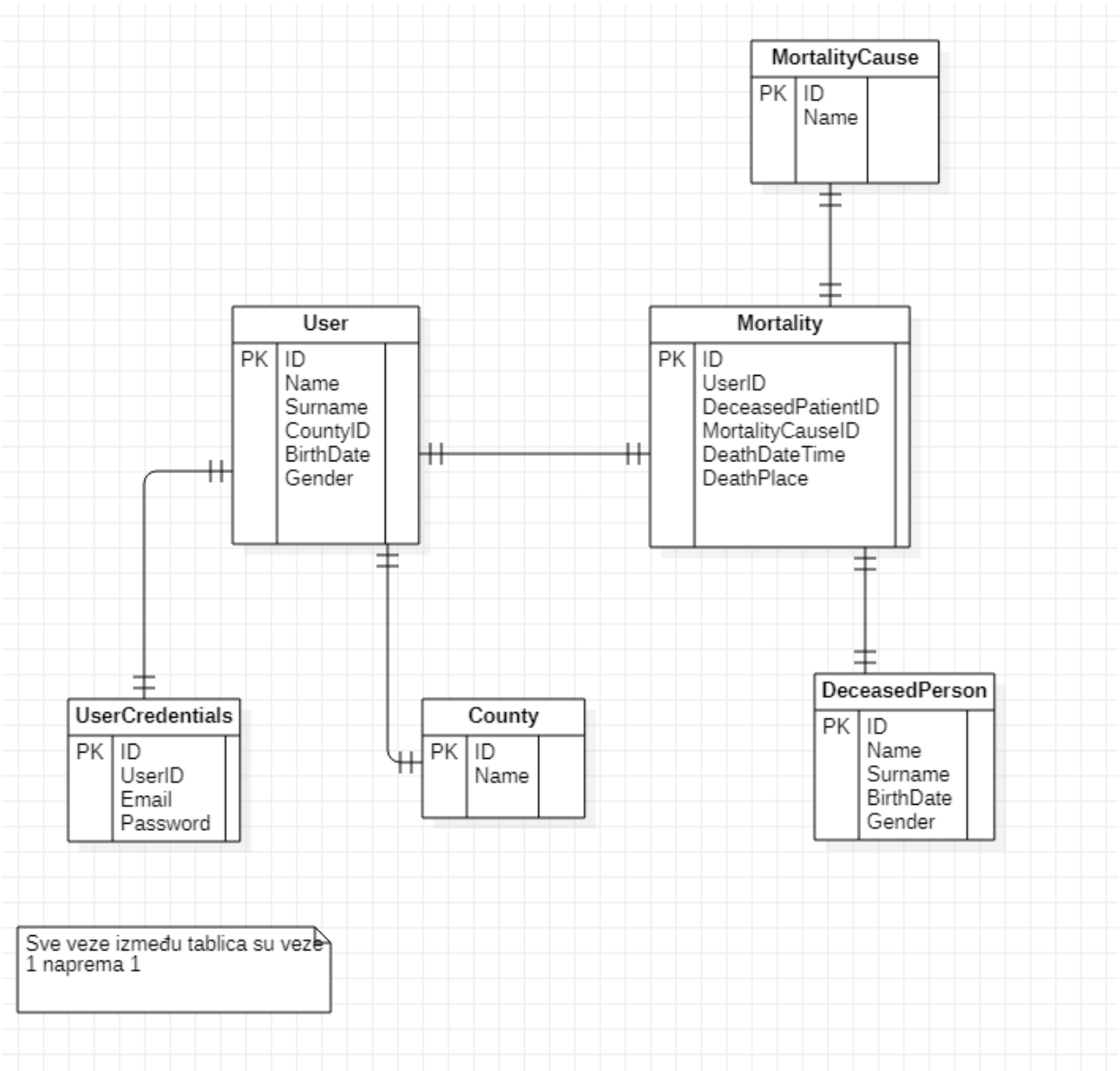
3.5.2. Očekivano korisničko iskustvo i pokazatelji korisničkog iskustva

Korisničko iskustvo sastoji se od korisničkih vjerovanja, preferenci, ideja, ponašanja i emocija dok ostvaruje interakciju sa sustavom. Prema tome, ono je neovisno u domeni ljudskog dojma, no izrazito ovisi o postavkama i načinima uporabe koji utječu na taj dojam, te je povezano s mogućnostima koje dolaze korištenjem sustava. Mjeri se metrikama kao što su kvaliteta i učinkovitost, prema korisnikovom dojmu o pouzdanosti i inovativnosti te ostalim strukturama povezanim s ljudskim emocionalnim povratnim informacijama pri uporabi sustava [22].

Prvi korak ka kvalitetnom korisničkom iskustvu jest ispunjavanje zahtjeva korisnika mobilne aplikacije, a ukoliko se one ostvare na intuitivan i estetski prilagođen način, očekuje se da će korisničko iskustvo biti zadovoljavajuće.

3.6. Struktura baze podataka

Slika 3.2. prikazuje *ER* (engl. *entity relationship*) dijagram međusobno povezanih entiteta u bazi podataka. Baza podataka ostvaruje se u udaljenoj bazi podataka *Firestore.*



Slika 3.2. *ER* model baze podataka.

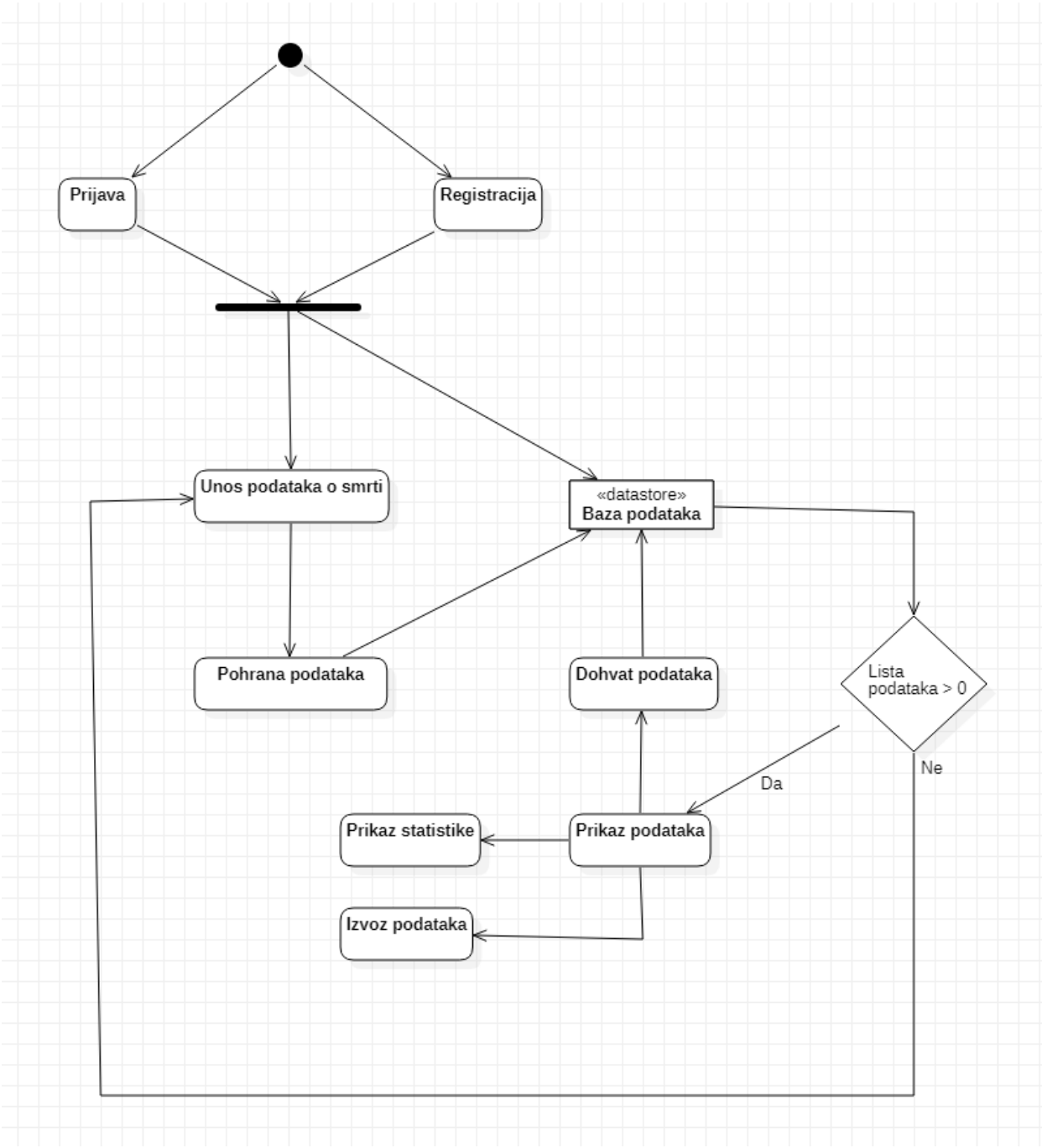
Primarni povezani entiteti su korisnik mobilne aplikacije – mrtvozornik (engl. *user*) i mortalitet (engl. *mortality*). Mortalitet sadrži strani ključ *UserID* preko kojeg su svi mortaliteti, koje je unio određeni korisnik, povezani. Također postoji još jedna značajnija veza stranim ključem, a to je

veza mortaliteta i preminule osobe (engl. *deceased person*). Na taj način se povezuju osobni podaci umrle osobe s uzrokom, mjestom i vremenom smrti.

Osim glavnih kolekcija baze podataka, postoje i određeni šifrarnici koji predstavljaju nepromjenjive podatke koji određuju stanja, svojstva i ponašanja glavnih tablica. To su: županija (engl. *county*), uzrok smrti (engl. *mortality cause*) te korisnički podaci (engl. *credentials*).

3.7. Struktura mobilne aplikacije prema idejnom rješenju

Predložena struktura optimizirane mobilne aplikacije za vođenje mortalitetne statistike predstavljena je dijagramom aktivnosti na slici 3.3.



Slika 3.3. Dijagram aktivnosti mobilne aplikacije prema idejnom rješenju.

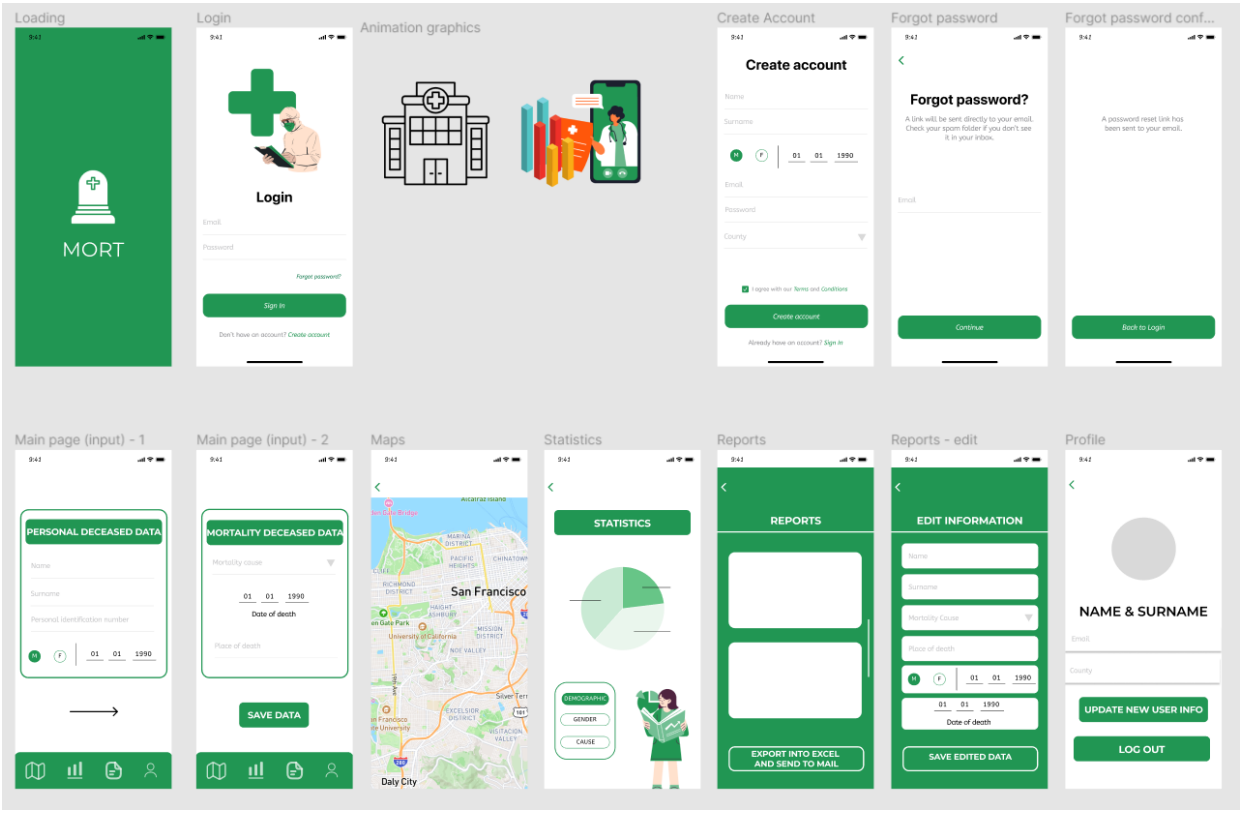
4. PROGRAMSKA IZVEDBA MOBILNE APLIKACIJE

4.1. Programske tehnologije, okoline i jezici korišteni za izradu aplikacije

Optimizirana verzija mobilne Android aplikacije za vođenje mortalitetne statistike razvijena je u programskom razvojnom okruženju (engl. *integrated development environment*) *Android Studio*. Vizualni koncept mobilne aplikacije ostvario se putem FIGME, programskog alata za razvoj dizajna korisničkog sučelja, a navedeno se implementiralo XML označnim jezikom. Klijentska i poslužiteljska strana povezane su zajedno s korisničkim sučeljem putem objektno-orijentiranog programskog jezika *Kotlin* principima ubrizgavanja (engl. *injection*) i povezivanja (engl. *binding*). Uz povezivanje i prikaz korisničkog sučelja na zaslonu, *Kotlin* udružuje različite mogućnosti i funkcionalnosti aplikacije kao što su: pohrana i dohvaćanje podataka s baze podataka *Firestore*, autentifikaciju korisnika, određivanja lokacije korisnika putem servisa *Google Maps* te ostale važne funkcionalnosti aplikacije. U sljedećim će se poglavljima detaljno razraditi načini izvedbe navedenih programskih komponenti i funkcionalnosti mobilne aplikacije.

4.1.1. FIGMA

FIGMA je programski alat koji omogućuje korisnicima jednostavniji pristup dizajnu. Svojim intuitivnim i snažnim okruženjem pretvara korisnikove ideje u responzivna i interaktivna sučelja [23]. Na slici 4.1. prikazani su ostvareni dizajni zaslona optimizirane verzije mobilne aplikacije za vođenje mortalitetne statistike. U prilogu 1 nalazi poveznica na FIGMA repozitorij.



Slika 4.1. Idejni dizajn sučelja mobilne aplikacije za vođenje mortalitetne statistike u programskom alatu FIGMA.

4.1.2. Operacijski sustav Android

Operacijski sustav Android mobilni je operacijski sustav kojeg je razvila razvojna tvrtka *Android, Inc.* Zamišljen je u obliku otvorenog izvornog koda (engl. *open source code*) kako bi se omogućili vanjski razvoj i utjecaj korisnika na nadogradnju operacijskog sustava [24]. Jedna od glavnih postavki, koje omogućuju razvoj aplikacija prema operacijskom sustavu *Android*, su *Android* SDK. To je skup biblioteka i razvojnih alata koje olakšavaju proces razvoja i pronalaska pogrešaka (engl. *debugging*) [25]. Arhitektura *Android* operacijskog sustava sadrži pet različitih slojeva, a to su: aplikacije, aplikacijski okvir, biblioteke, *Android Runtime* i *Linux* jezgra. Neke od glavnih mogućnosti operacijskog sustava *Android* su sljedeće: automatizacija, optimizirano grafičko sučelje, višejezična podrška, mrežna i izvan mrežna povezanost (2G-5G tehnologije, *bluetooth*, WI-FI, NFC, GPS) i ostalo [26].

4.1.3. *Android Studio*

Android Studio službena je razvojna okolina za razvoj *Android* aplikacija. Temelji se na snažnom prevoditelju koda i razvojnim alatima koje pruža IntelliJ IDEA. Također, ono pruža dodatne mogućnosti koje poboljšavaju produktivnost prilikom izgradnje *Android* aplikacija kao što su:

- prilagodljivi gradivni sustav temeljen na *Gradle*-u,
- virtualne uređaje (engl. *emulators*) koji su brzi i osnaženi raznim mogućnostima,
- jedinstvenu okolinu za razvoj aplikacija nad svim *Android* uređajima,
- veliki opseg ispitnih alata i platformi koji olakšavaju ispitivanje aplikacija.

Svaki projekt u *Android Studiju* sastoji se od jednog ili više modula koji sadrže datoteke izvornog koda i resurse. Svaki modul aplikacije sadrži sljedeće datoteke: *manifest* (sadrži datoteku AndroidManifest.xml), *java* (sadrži *Kotlin* i *Java* programske datoteke) i *res* (sadrži sve neprogramske resurse poput UI datoteka i multimedijskog sadržaja) [27].

4.1.4. XML

XML (engl. *Extensible Markup Language*) označni je jezik sličan HTML-u (engl. *HyperText Markup Language*), ali bez predefiniranih oznaka za korištenje. Umjesto toga, moguće je definirati vlastite oznake. Budući da je XML standardiziran takozvanom standardnom XML sintaksom, prijenos podataka u obliku XML-a olakšan je jer primatelj može pročitati (engl. *parsing*) podatke neovisno o platformi i mrežnoj infrastrukturi [28]. U *Android Studio*, XML služi za implementaciju i oblikovanje podataka vezanih za korisničko sučelje – pogledi (engl. *views*) [29].

4.1.5. *Kotlin*

Kotlin je moderan programski jezik dizajniran da olakša rad razvojnim inženjerima. Jezgrovit, siguran i kompatibilan s programskim jezikom *Java*, *Kotlin* pruža mnoštvo načina kako ponovno upotrijebiti programski kod između više različitih platformi za produktivno programiranje. Mogućnosti koje *Kotlin* nudi su sljedeće:

- manje programskog koda, ali se povećava njegova čitljivost,
- maleni broj uobičajenih problema koji su temeljeni na unutarnjim *Google* podacima,
- *Kotlin* podržava: biblioteke *Jetpack* koje su snažni alati za izgradnju korisničkih sučelja, korutine (engl. *coroutines*), metode proširenja (engl. *extension functions*), *lambda* funkcija (engl. *lambdas*) i drugo,
- podržava višeplatformski razvoj,
- kompatibilnost i kooperativnost s programskim jezikom *Java* [30].

4.1.6. *Firebase*

Firebase je *Google*-ova platforma za razvoj i unaprjeđivanje mobilnih aplikacija. Pruža mnoštvo servisa koji su izrazito moćni i pristupačni. Servisima se pristupa preko krajnjih točaka API (engl.

Application Program Interface), a zatim se preko dobivene reference pristupa funkcionalnostima platforme *Firebase*. Mogućnosti koje uključuje *Firebase* su sljedeće: analitika, autentifikacija, pohrana i spremište podataka, konfiguracijske postavke, mogućnost stvaranja obavijesti i drugo. Servisi uključuju:

- *Authentication* – servis za autentifikaciju putem prijave i identifikacije,
- *Realtime Database i Cloud Firestore* – servis za pohranu podataka u stvarnom vremenu na udaljenu nerelacijsku bazu podataka,
- *Cloud Storage* – servis za pohranu podataka s velikim memorijskim zauzećem [31].

4.1.7. *Google Maps*

Google Maps je alat za lokacijsko pretraživanje kojeg je razvio *Google*. Omogućuje određivanje željene geografske lokacije, rute dolaska do lokacije, detaljni prikaz atrakcija i informacija o lokaciji te virtualni uvid u lokaciju putem usluge *Google Street View*. *Google Maps* koristi satelitsku podršku, GPS uređaja, Wi-Fi i *Google-ovu* mrežnu infrastrukturu kako bi locirao korisnika. i stvorio kartu lokacije [32].

4.1.8. *Koin*

Koin je biblioteka ubrizgavanja ovisnosti koja uz pomoć jednostavnih alata i API poziva omogućuje korisniku izgradnju *Kotlin* aplikacija. Omogućuje korištenje instanci i ovisnosti u bilo kojem dijelu programskog rješenja. Također omogućuje i višeplatformsko ubrizgavanje ovisnosti čime se ostvaruje izgradnja višeplatformskih mobilnih aplikacija koje dijele zajednički kod [33].

4.1.9. *LiveData*

LiveData je klasa koja omogućuje korisniku promatranje podataka za koje je vezana pritom je svjesna životnog ciklusa komponenti koje su pokrenute u aplikaciji. Ovakav pristup omogućuje klasi *LiveData* da ažurira samo one promatrače aplikacijske komponente koja je aktivna po vlastitom životnom ciklusu. Neke od prednosti *LiveData* uključuju: usklađenost korisničkog sučelja i stanja podataka, *memory leaks*, padove aplikacije zbog zaustavljenih aktivnosti, svjesnost o rukovanju životnim ciklusom i drugo [34].

4.2. Programski predložak mobilne arhitekture

Svaka programska podrška koja ima interakciju s korisnikom zahtjeva korisničko sučelje. Međutim, isto tako je i vječni inženjerski problem način same integracije korisničkog sučelja s aplikacijskom domenom. Predlošci mobilne arhitekture (engl. *architecture patterns*) predstavljaju

uobičajena shematska rješenja za ponavljajuće probleme dizajna, a pružaju inženjerima pristup velikom rasponu sustavnog znanja kroz povezane materijale [35].

Neki od primjera predložaka mobilnih arhitektura su sljedeći:

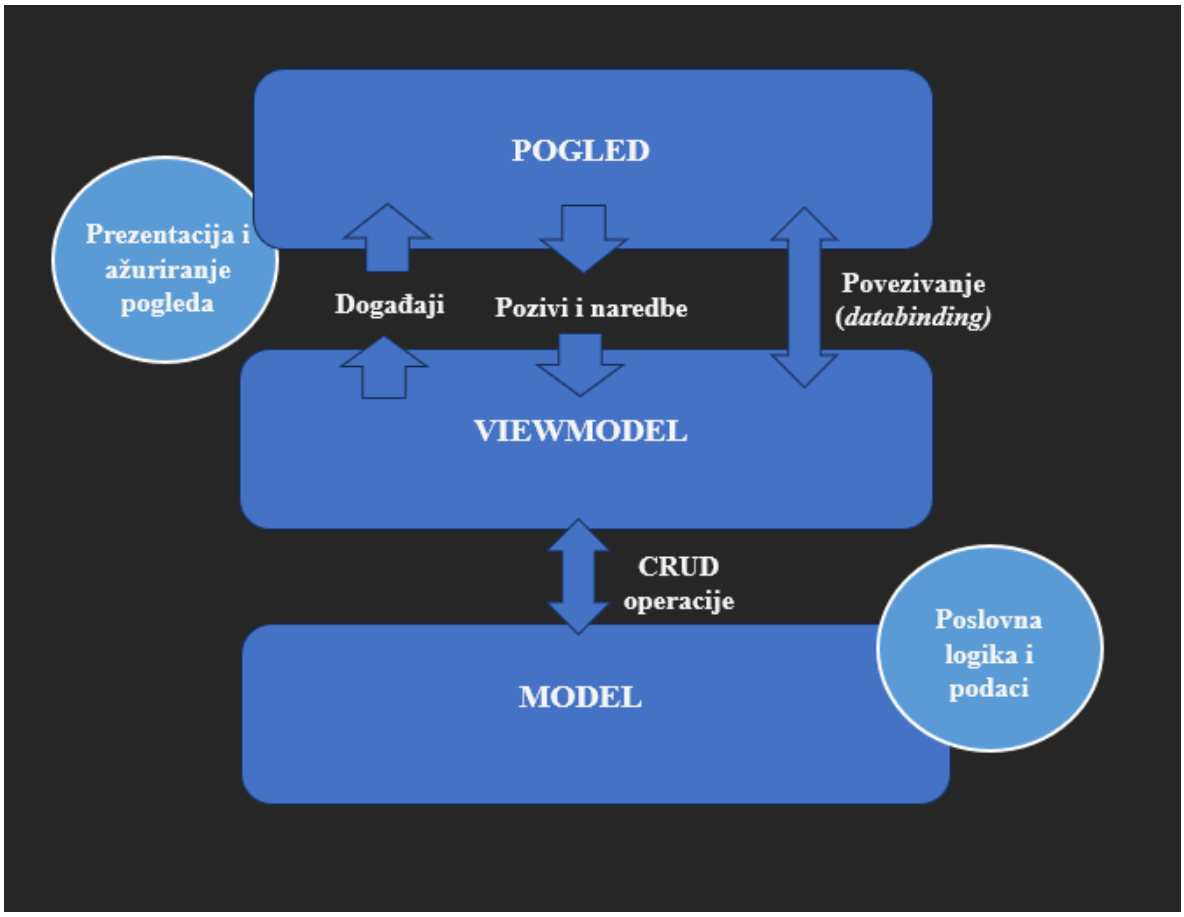
- *Model-View-Controller* (MVC),
- *Model-View-Presenter* (MVP),
- *Model-View-ViewModel* (MVVM),
- *Model-View-Intent* (MVI).

U optimiziranoj verziji mobilne aplikacije za vođenje mortalitetne statistike koristi se predložak mobilne arhitekture MVVM.

4.2.1. Predložak mobilne arhitekture MVVM

Model-View-ViewModel varijacija je predloška *Model-View-Controller*, a dizajnirana je prema modernim razvojnim platformama korisničkog sučelja gdje pogled postaje odgovornost dizajnera, a ne razvojnog inženjera. Jedna od važnih značajki na koju se oslanja ovaj suvremeni predložak je mehanizam povezivanja podataka između dizajnerske i programske strane (engl. *data binding*) [36]. Zbog navedenog, pogled ima mogućnost vlastitog povezivanja (engl. *binding*) s podacima te nakon što se oni učitaju, pogled se ažurira i mijenja automatski.

Predložak se sastoji od tri komponente: modela, pogleda i *viewmodel*-a. Model predstavlja podatke i poslovnu logiku sustava dok *viewmodel* služi za upravljanje stanjem pogleda, prijenosa podataka i poslovne logike na pogled. Odnos između navedenih komponenti prikazani su prema slici 4.2. [37].



Slika 4.2. Odnosi komponenti unutar predloška MVVM.

4.3. Programsko rješenje na strani korisnika

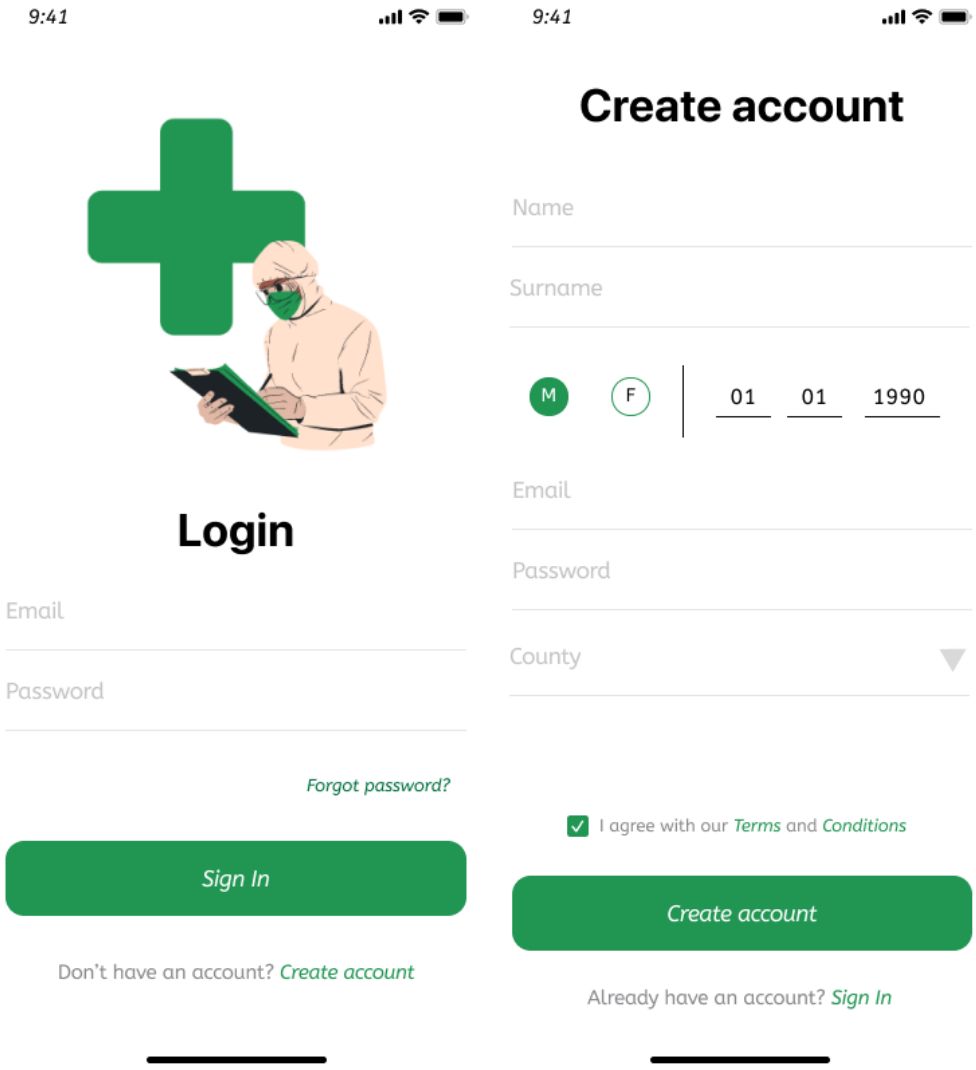
Poglavlje 4.3. predstavlja programsko rješenje na strani korisnika gdje se objašnjavaju i predstavljaju programska rješenja načina unošenja podataka, pohrane te personalizacije osobnih podataka i podataka o mortalitetu, te određivanje korisnikove lokacije.

4.3.1. Unos podataka

Unos podataka sastoji se od dva semantički odvojena, ali povezana dijela: unosa podataka za prijavu, odnosno registraciju te unosa mortalitetnih podataka.

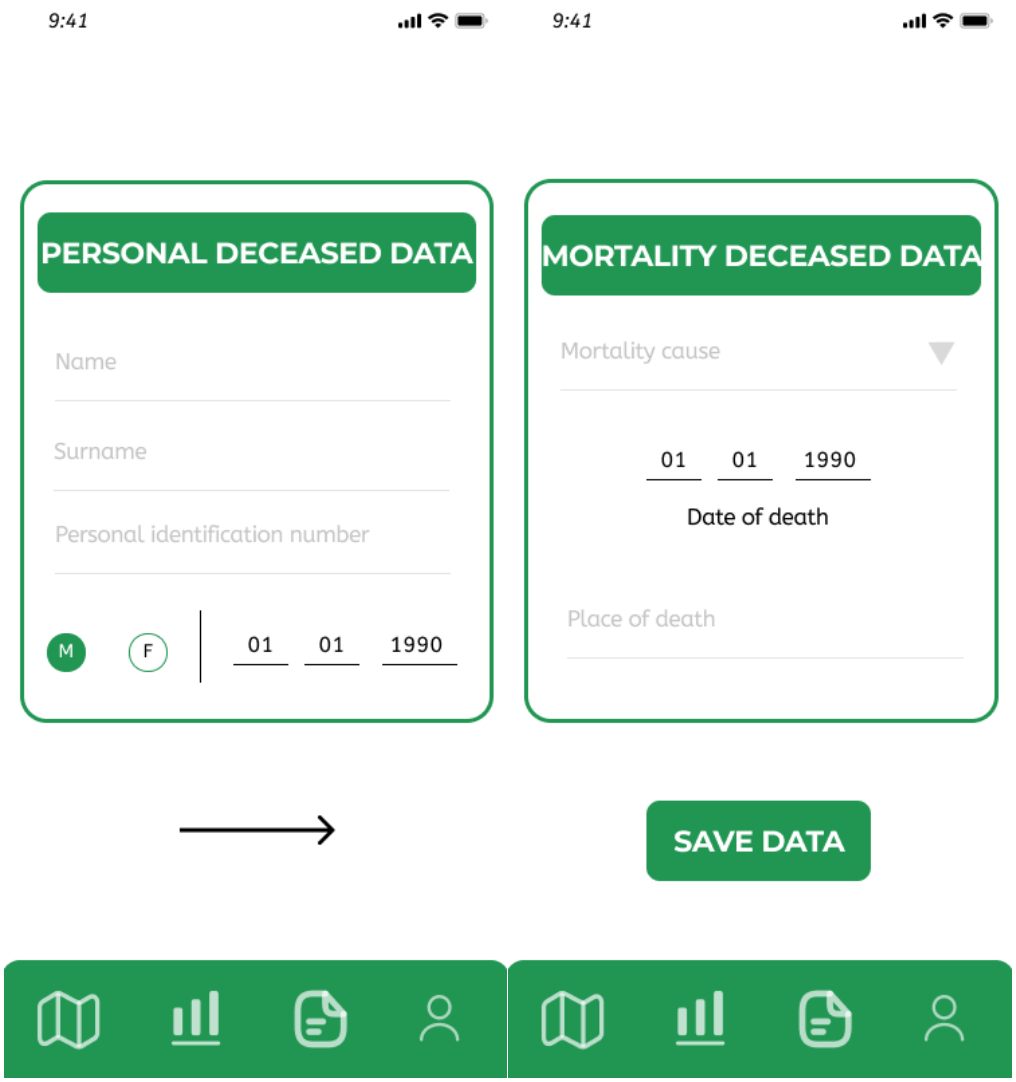
Unos podataka za prijavu, odnosno registraciju ostvaren je način da korisnik unese nužne podatke u poglede *EditText* ovisno radi li se o fragmentu prijave ili registracije. Fragment prijave sastoji se od unosa korisnikove e-mail adrese i lozinke dok fragment registracije sadrži unos korisnikovog: imena, prezimena, spola, datuma rođenja, e-mail adrese, lozinke i županije djelovanja. Budući da MVVM ima posebnu karakteristiku povezivanja korisničkog sučelja i podataka s njegovim prikazom, resursi za unos podataka ostvarili su se u XML datotekama koji služe za dizajn

korisničkog sučelja te su se putem svojstva povezivanja ubrizgali u *Kotlin* klase. Slika 4.3. predstavlja korisnička sučelja za prijavu i registraciju ostvarena postupcima povezivanja i ubrizgavanja.



Slika 4.3. Korisnička sučelja za prijavu i registraciju.

Središnji dio mobilne aplikacije sastoji se od unosa podataka umrle osobe i detaljima smrti. Podaci o umrloj osobi uključuju unos imena, prezimena, datuma rođenja i osobnog identifikacijskog broja osobe. Pritiskom na gumb u obliku strjelice, korisniku se prikazuje sučelje za unos podataka o detaljima smrti umrle osobe. To su: uzrok, datum i vrijeme smrti. Pogledi za unos podataka o detaljima smrti umrle osobe produkt su ostvarenog dizajna sučelja u označnom jeziku XML i ubrizgavanja tog sučelja u pripadajuće *Kotlin* klase, a koje je olakšano postupkom povezivanja. Navedena sučelja se prikazuju na slici 4.4.



Slika 4.4. Korisnička sučelja za unos podataka o umrloj osobi.

4.3.2. Pohrana podataka u bazu podataka

Pohrana podataka na udaljenu bazu podataka *Firestore* odvija se uz pomoć pristupa internetu. Zato je nužno dodati dozvolu pristupa internetu kako bi prilikom rada mobilna aplikacija mogla koristiti mrežu mobilnog uređaja. Dozvole (engl. *permissions*) za pristup različitim uslugama i interakciji s korisnikovim uređajem definiraju se unutar datoteke *manifest* projektnog rješenja.

Baza podataka na udaljenoj bazi *Firestore* sadrži tri temeljne kolekcije: *users*, *causes* i *counties*. *Firestore* NoSQL baza podataka, odnosno nerelacijska baza koja pohranjuje podatke u obliku kolekcija i dokumenata. Tako kolekcije *causes* i *counties* sadrže statički pohranjene podatke o uzrocima smrti i županijama djelovanja dok dinamička kolekcija *users* sadrži atribute o karakteristikama korisnika aplikacije i kolekcije niže razine: *deceased_persons* i *mortalities* čiji dokumenti sadrže referencu na *userID* korisnika aplikacije. Tako se dohvatom podataka o

korisniku dohvaćaju i njegove kolekcija umrlih osoba s informacijama o njihovim smrtima koje je pohranio u bazu podataka.

Nakon unosa podataka s pripadajućih sučelja koji su opisani u prethodnom poglavlju, analogno slijedi njihova pohrana. Pohranjivanje se izvodi pritiskom na odgovarajuće gumbe ovisno u kojem dijelu aplikacije se korisnik nalazi (na slici 4.3. – *„Create account“*, na slici 4.4. – pritiskom na strjelicu i gumb „*Save data*“). Prije pohranjivanja podataka o korisniku aplikacije ili unesenim informacijama o umrloj osobi odvija se validacija unesenih podataka na nekoliko razina. Prva razina je provjera jesu li svi podaci uneseni, a ukoliko nisu javlja se odgovarajuća poruka i indikator koji podaci nedostaju. Zatim slijedi validacija na razini usporedbe s podacima u bazi podataka. Na slici 4.3 prilikom registracije korisniku se onemogućuje registracija s već postojećom adresom elektroničke pošte, dok se na slici 4.4 korisniku onemogućuje pohrana umrle osobe ako već postoji određena osoba u bazi podataka koja dijeli jednak osobni identifikacijski broj. Naposljetku se odvija provjera na razini sintaksnog oblika unesenih podataka. Tako na slici 4.4 atribut *„Personal identification number*“ mora sadržavati numeričke znakove koji moraju odgovarati duljini od 11 znamenki.

Nakon što se utvrdi ispravnost podataka, podaci se predaju poslužiteljskom dijelu aplikacije koji je zaslužan za komunikaciju s bazom podataka te se preko nužnih instanci podaci spremaju u obliku *HashSet* vrijednosti gdje je ključ (engl. *key*) tekstualni opis atributa (u relacijskoj bazi podataka to bi bio stupac), a vrijednost (engl. *value*) je vrijednost tog atributa, odnosno ono što je korisnik unio kao podatak. Ovisno o uspješnosti pohrane podataka korisniku se prikazuje poruka o ishodu pohrane.

4.3.3. Uređivanje osobnih podataka i podataka umrle osobe

Kako bi se korisniku pružilo što kvalitetnije korisničko iskustvo preko funkcionalnih mogućnosti, uz unos i stvaranje podataka mobilna aplikacija pruža i ostale *CRUD* operacije (engl. *create-read-update-delete*) nad podacima.

Dohvat, način obrade i prikaz podataka detaljno su objašnjeni u poglavlju *„Programsko rješenje na strani poslužitelja*“. Nakon dohvata podataka u obliku liste, korisnik ima mogućnost pritiska na jednu od umrlih osoba. Ako je taj pritisak bio kratkotrajan, izvodi se metoda koja je zadana unutar *lambda* funkcije *SetOnClickListener*, a koja korisnika preusmjerava na novo sučelje. Novo sučelje pruža mu mogućnost ažuriranja osobnih podataka umrle osobe ili informacija o smrti. Ovisno o izboru (želi li korisnik ažurirati osobne podatke o umrloj osobi ili informacije o okolnostima smrti umrle osobe), korisniku se prikazuju sučelja za unos ažuriranih podataka o

umrloj osobi koji se zatim spremaju u bazu podataka. Međutim, ako je pritisak korisnika na jednu od umrle osobe bio dugotrajan, izvodi se metoda definirana unutar *lambda* funkcije *SetOnLongClickListener* s kojom se ostvaruje brisanje podataka o umrloj osobi.

Isto tako, korisniku je pružena mogućnost uređivanja vlastitih podataka kao što su ime, prezime i adresa elektroničke pošte, ali je za njihovo uređivanje nužno ponovno unijeti lozinku. Korisnik ima mogućnost uređivanja i vlastite lozinke, ali samo na način da na slici 4.3 odabere da je zaboravio lozinku (gumb „*Forgot password?"*) te si na elektroničku poštu pošalje poveznicu s kojom će ponovno postaviti lozinku korisničkog računa.

4.4. Programsko rješenje na strani poslužitelja

U ovom poglavlju se detaljno obrađuju funkcionalnosti ostvarene na strani poslužitelja. To su: prijava, odjava i registracija korisnika putem usluga servisa *Firebase*, prikaz pohranjenih podataka o mortalitetu iz baze podataka za pojedinačnog korisnika, prikaz statističkih podataka na temelju skupne analize po određenim parametrima te kako je ostvareno povezivanje na bazu podataka i provjera postojanosti internetske povezanosti.

4.4.1. Prijava i odjava

Firebase Authentication omogućuje servise na poslužiteljskoj strani koji se ostvaruju u mobilnoj aplikaciji putem alata SDK. Također je osnažen kvalitetnim korisničkim sučeljem kojim je korisniku omogućeno praćenje stanja korisnika koji se služe njegovom uslugom autentifikacije unutar mobilne aplikacije. Podržava autentifikaciju putem lozinke, brojeva mobitela, povezivanja s *Google* ili *Facebook* računom te drugim načinima. Isto tako je bitno naznačiti kako servis maksimalno iskorištava mogućnosti suvremenih industrijskih standarda poput *OAuth 2.0* i *OpenID Connect*-a pa se tako može jednostavno integrirati s vlastitom poslužiteljskom stranom [38].

Usluga prijave i odjave korisnika optimizirane verzije mobilne aplikacije odvija se putem servisa *Firebase Authentication*. Slika 4.4. predstavlja metodu *login* koja se poziva prilikom prijave korisnika. Zatim se logika prepušta *viewmodel*-u koji prenosi poslužiteljskoj strani informacije o korisnikovoj e-mail adresi i lozinci. Na slici 4.5. može se vidjeti ostvarena metoda *isRegistered* unutar repozitorija metoda koje služe za rad s modelom *User*. Navedena metoda vraća vrijednost podatka *boolean* te ako je vrijednost istina (engl. *true*) znači da korisnik postoji u bazi, servis ga autentificira i dopušta ulaz u nastavak mobilne aplikacije. Međutim ako *User* ne postoji u bazi podataka, tada mu se pojavljuje obavijest o neuspješnoj prijavi. Ostvarena logika za odjavu korisnika iz mobilne aplikacije prikazana je na slici 4.6.

```
private fun login() {
    try {
        val email = binding.emailInput.text.toString().trim()
        val password = binding.passwordInput.text.toString().trim()
        val errors = Validator.validateUserForLogin(email, password)
        if (errors.isNotEmpty()) {
            Toast.makeText(context, errors.first().Description, Toast.LENGTH_LONG).show()
        } else {
            userViewModel.isUserRegistered(email, password)
            Handler(Looper.getMainLooper()).postDelayed({
                if (firebaseAuth.currentUser != null) {
                    Toast.makeText(context, text: "Logged in successfully!", Toast.LENGTH_SHORT)
                        .show()
                    startActivity(
                        Intent(
                            context,
                            MainActivity::class.java
                        )
                    )
                } else
                    Toast.makeText(context, text: "Error! Login unsuccessful!", Toast.LENGTH_LONG)
                        .show()
            }, delayMillis: 2000)
        }
    }catch (exception: Exception){
        Toast.makeText(context, text: "Unexpected error occurred, please right again in a minute!", Toast.LENGTH_LONG).show()
        exception.message?.let { Log.e(ContentValues.TAG, it) }
    }
}
```

Slika 4.4. Metoda *login* za prijavu korisnika.

```
override fun isRegistered(email: String, password: String) {
    firebaseAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener { task ->
            task.isSuccessful

        }
}
```

Slika 4.5. Metoda za autentifikaciju korisnika prilikom prijave.

```
private fun logout() {
    firebaseAuth.signOut()
    Utils.switchActivity(
        requireActivity(),
        AuthenticationActivity::class.java,
        killOld: true
    )
}
```

Slika 4.6. Metoda *logout* za odjavu korisnika.

4.4.2. Registracija

Registracija korisnika u sustav se također odvija putem usluga servisa *Firebase Authentication*. Nakon što korisnik unese sve potrebne podatke za registraciju u sustav i pošalje zahtjev za stvaranjem računa, metoda *register* pruža poslužiteljskoj strani podatke o korisniku koji se zatim

putem potrebnih referenci pohranjuju u udaljenu bazu podataka *Firestore*. Podaci se pohranjuju u obliku dokumenta, a svaki dokument dobiva jedinstveni identifikator koji je zapravo i identifikator korisnika u bazi podataka. Ako je registracija neuspješna ili se dogodila pogreška, sustav će baciti iznimku i upozoriti korisnika. Na slikama 4.7. i 4.8. prikazane su metode *register* i *save* koje su zaslužne za pohranjivanje novog korisnika u sustav.

```
private fun register() {
    try {
        val name = binding.nameInput.text.toString()
        val surname = binding.surnameInput.text.toString()
        val email = binding.emailInput.text.toString()
        val password = binding.passwordInput.text.toString()

        val gender =
            if (binding.genderHolder.checkedRadioButtonId == binding.genderMale.id)
                binding.genderMale.text.toString()
            else
                binding.genderFemale.text.toString()

        val birthDate =
            String.format("${binding.datePicker.dayOfMonth}/${binding.datePicker.month}/${binding.datePicker.year}")

        val newUser = User( id: "", name, surname, email, birthDate, gender, selectedCounty)
        val isConsentChecked = binding.checkForConsent.isChecked
        val errors: MutableList<Error> = Validator.validateUserForRegistration(
            newUser, password, isConsentChecked, notAllowedEmails)
```

```
    if (errors.isNotEmpty()) {
        var collectedErrors = String()
        errors.forEach { error -> collectedErrors += (error.Description + " ") }
        Toast.makeText(context, collectedErrors, Toast.LENGTH_LONG).show()

    } else {
        userViewModel.saveUser(newUser, password)
        Handler(Looper.getMainLooper()).postDelayed({
                if (firebaseAuth.currentUser != null) {
                    Toast.makeText(context, text: "Sign in successful", Toast.LENGTH_LONG).show()
                    startActivity(Intent(context, MainActivity::class.java))
                } else {
                    Toast.makeText(context, text: "Unsuccessful sign in", Toast.LENGTH_LONG).show()
                } }, delayMillis: 2000)
    }
} catch (exception: Exception) {
    Toast.makeText(context, text: "Unexpected error occurred, please wait and try again.", Toast.LENGTH_LONG).show()
    Log.e(ContentValues.TAG, msg: "Error: ${exception.message}")
}
```

Slika 4.7. Metoda *register* za registraciju korisnika.

```
override fun save(user: User, password: String) {
    firebaseAuth.createUserWithEmailAndPassword(user.email, password)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                user.id = firebaseAuth.currentUser!!.uid
                val documentReference: DocumentReference =
                    userCollectionReference.document(user.id)
                val newUser: MutableMap<String, Any> = HashMap()
                newUser["name"] = user.name
                newUser["surname"] = user.surname
                newUser["email"] = user.email
                newUser["birthDate"] = user.birthDate
                newUser["gender"] = user.gender
                newUser["county"] = user.county

                documentReference.set(user).addOnSuccessListener {  it: Void!
                    Log.d(
                        ContentValues.TAG,
                         msg: "OnSuccess: user profile created for ${user.email}"
                    )
                    return@addOnSuccessListener
                }
                    .addOnFailureListener {  it: Exception
                        Log.d(ContentValues.TAG,  msg: "OnFailure: ${it.message}")
                        return@addOnFailureListener
                    }
            }
        }
}
```

Slika 4.8. Metoda *save* za pohranu korisnika u bazu podataka.

4.4.3. Prikaz rezultata pohranjenih u bazu podataka

Jedna od najbitnijih funkcionalnosti je dohvat podataka iz baze podataka. Postojanje podataka nužno je za izvedbu svih ostalih funkcionalnosti, odnosno kako bi se upotrijebile potpune mogućnosti i potencijali mobilne aplikacije. Tako je i prikaz rezultata jedna od takvih funkcionalnosti.

Unutar mobilne aplikacije, podaci se prikazuju na nekoliko različitih načina. Glavni način je u obliku liste koju prikazuje pogled *RecyclerView*. Svaki pojedinačni element liste oblikovan je kao dio pogleda (engl. *item view*) kojim upravlja *RecycleAdapter*. *RecycleAdapter* je zaslužan za povezivanje dijela pogleda s podacima koji dobije adapter s baze podataka te logikom rukovanja sa samim djelićem. Budući da se pogled *RecycleView* stvara prilikom metode *OnCreateView*, bitno je imati spremne podatke koji će se predati adapteru za prikazivanje na *RecycleView*. Međutim, budući da se radi dohvat podataka s udaljene baze podataka, radi osvježivanja stanja adaptera, podaci se spremaju u tip podataka oblika *MutableLiveData*. *MutableLiveData* u suradnji s *viewmodel*-om omogućuje promatranje (engl. *observing*) stanja varijable. Na taj se način osigurava

postojanje podataka. Kada se dohvate podaci, promatrač (engl. *observer*) putem životnog ciklusa *viewmodel*-a javlja adapteru da se stanje varijable promijenilo što omogućuje da se adapter popuni podacima i ostvari prikaz podataka. Prikaz se također može filtrirati s obzirom na unos vrijednosti u filter koji filtrira podatke prema osobnom identifikacijskom broju umrle osobe. Metoda *setUpRecycler* koja odrađuje opisanu logiku prikazana je na slici 4.9.

```
private fun setUpRecycler() {
    binding.queryResultsRecylcer.layoutManager = LinearLayoutManager(
        context,
        RecyclerView.VERTICAL,
        reverseLayout: false
    )
    val itemDecorator = DividerItemDecoration(context, DividerItemDecoration.VERTICAL)
    itemDecorator.setDrawable(ContextCompat.getDrawable(context!!, R.drawable.divider)!!)
    binding.queryResultsRecylcer.addItemDecoration(itemDecorator)
    adapter = SearchReportAdapter()
    viewModelPatient.getPatientsSync().observe(viewLifecycleOwner) {  it: MutableList<Patient>?
        if (it != null) {
            if (it.isNotEmpty()) {
                adapter.setPatients(it)
                patientsForExport = it
            }
        }
    }
    adapter.onReportSelectedListener = this
    adapter.onReportSelectedLongListener = this
    binding.queryResultsRecylcer.adapter = adapter
}
```

Slika 4.9. Metoda *setUpRecycler* za prikaz podataka u obliku vertikalne liste.

Na sličan način ostvaruje se logika prikazivanja podataka za ažuriranje, samo što se u ovom slučaju podaci popunjavanju u obliku tekstualne naznake (engl. *hint*) u poglede *EditText* koji služe za unos podataka. Naznake odražavaju i prikazuju stanje podataka u bazi podataka, te omogućuju korisniku njihovo uređivanje putem pogleda *EditText.*

4.4.4. Prikaz statističke analize rezultata i njihov izvoz u obliku XSLX datoteke

Kako bi se olakšao statistički aspekt obavljanja službe mrtvozorništva, ostvarene su funkcionalnosti statističkog prikaza pohranjenih podataka te njihov izvoz u obliku XSLX datoteke i slanje na elektroničku adresu podrške mobilne aplikacije.

Statistički prikaz ostvaren je u obliku „pita“ dijagrama (engl. *pie chart*) uz pomoć implementacije ovisnosti (engl. *dependency*) paketa *PieChart* [39]. Omogućen je prikaz postotnog udjela podataka s obzirom na demografiju umrlih osoba i uzroka smrti. Korištena metoda za obradu i pripremu podataka za prikaz je metoda uvećavanja koja obrađuje broj pohranjenih podataka s obzirom na uvjet i prikazuje ih kao odvojene entitete.

Funkcionalnost pretvorbe podataka u XSLX datoteku omogućuje korisniku detaljan ispis svih umrlih osoba i njihovih podataka koji su spremni za slanje u obliku izvještaja. Datoteka se također automatski šalje na elektronsku poštu podrške mobilne aplikacije te se po završetku postupka pretvorbe briše iz memorije. Navedene funkcionalnosti pretvorbe u XSLX datoteku su ostvarene uz pomoć paketa *Apache POI* i *Apache POI-ooxml* [40], dok se slanje elektronske pošte s privitkom odvija uz podršku API paketa *JavaMail* [41] putem protokola SMTP (engl. *Simple Mail Transfer Protocol*). Slika 4.10. prikazuje logiku odvijanja asinkronog slanja elektroničke pošte s privitkom.

```
    val message: Message = MimeMessage(session)
    message.subject = "Email from MORT Application"

    val addressTo: Address = InternetAddress( address: "mort.testings@gmail.com")
    message.setRecipient(Message.RecipientType.TO, addressTo)

    val multipart = MimeMultipart()
    val attachment = MimeBodyPart()
    attachment.attachFile(File(tempFile.toUri()))

    val messageBody = MimeBodyPart()
    messageBody.setContent( o: "<h1>Mortality statistic from database</h1>",  type: "text/html")
    multipart.addBodyPart(messageBody)
    multipart.addBodyPart(attachment)

    message.setContent(multipart)

    CoroutineScope(Dispatchers.IO).launch {  this: CoroutineScope
        Transport.send(message)
    }
    Toast.makeText(context,  text: "Email sent successfully!", Toast.LENGTH_LONG).show()
} catch (exception: Exception) {
    Toast.makeText(context,  text: "Error while sending e-mail!", Toast.LENGTH_SHORT).show()
}
```

Slika 4.10. Logika sastavljanja i asinkronog slanja elektronske pošte s privitkom.

4.4.5. Određivanje korisnikove lokacije

Određivanje korisnikove lokacije dodatna je funkcionalnost koja korisniku olakšava unos podatka o mjestu smrti umrle osobe budući da se cjelokupni podaci o umrloj osobi upisuju na mjestu smrti. Aplikacija automatski locira korisnika na temelju *Google Maps* API-ja ako je korisnik omogućio dozvole za pristup internetu te pristup finom i grubom lociranju. Uz definirane dozvole unutar datoteke *manifest*, također je potrebno stvoriti API ključeve koji omogućuju pozive prema uslugama *Google Maps* API-ja. Uz pomoć *callback* metode *getMapAsync* i varijable *fusedLocationProviderClient* ostvaruje se učitavanje pogleda mape, kao i automatsko lociranje

korisnika s obzirom na zadnju registriranu lokaciju. Slika 4.11. prikazuje opisanu implementiranu logiku određivanja lokacije korisnika mobilne aplikacije.

```
override fun onMapReady(googleMap: GoogleMap) {
    mMap = googleMap
    if (ActivityCompat.checkSelfPermission(
            context: this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            activity: this,
            arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
            LOCATION_PERMISSION_REQUEST_CODE
        )
        return
    }
    mMap.isMyLocationEnabled = true
    fusedLocationProviderClient.lastLocation.addOnSuccessListener { it: Location!
        val userLocation = LatLng(it.latitude, it.longitude)
        mMap.addMarker(MarkerOptions().position(userLocation).title( title: "Your current location!"))
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(userLocation, DEFAULT_ZOOM))
    }
}
```

Slika 4.11. Metoda *onMapReady* koja određuje korisnikovu lokaciju.

4.4.6. Povezivanje na bazu podataka

Spajanje na bazu podataka *Firestore* odvija se prema sljedećim pozadinskim koracima:

1. povezivanje i ugrađivanje SDK paketa koji služe za povezivanje *Google*-ovih servisa s mobilnom aplikacijom,
2. postavljanje konfiguracijskih pravila unutar baze podataka, odnosno prava pristupa i postupanja s bazom podataka,
3. globalna varijabla koja sadrži instancu *Firebase*-ove usluge koja se želi koristiti (*Authentication, Firestore, FirebaseStorage* i drugo),
4. globalna varijabla nad kolekcijom dokumenata kojoj se želi pristupiti.

Ujedno je potrebno omogućiti pristup internetu unutar *manifest* datoteke mobilne aplikacije kako bi se moglo pristupiti udaljenoj bazi podataka. Nakon što se svi navedeni koraci ostvare, smatra se da je aplikacija povezana s bazom podataka *Firestore*.

4.4.7. Provjera postojanja internetske povezanosti

Postojanost interneta nužna je za određene funkcionalnosti poput: prikaza mape, određivanja korisnikove lokacije i slanja XSLX datoteke putem elektroničke pošte. Zato je potrebno osigurati

provjeru postojanosti interneta prije korištenja navedenih funkcionalnosti kako ne bi došlo do neželjenih stanja mobilne aplikacije. Navedeno se ostvaruje klasom *ConnectivityManager* koja pruža API pozive za rukovanje zahtjevima kojima se povezuje uređaj na internetsku mrežu. Zahtjevi se provjeravaju na temelju sposobnosti internetske postojanosti (engl. *Internet capabilities*), odnosno statusa postojanja interneta te opcija prijenosa podataka (engl. *data transport options*). API pozivi određuju je li uređaj trenutno povezan na mrežu i ako jest, provjeravaju se sposobnosti mreže s obzirom na njezinu povezanost s internetom [42].

U suradnji s naslijeđenom klasom *LiveData* promatra se postojanost internetske povezanosti. Na taj se način ostvaruje konstantno promatranje objekta klase *NetworkConnectivityChecker*. Ako postoji povezanost s mrežom, provjeravaju se njezine sposobnosti povezivanja na internet na način da se stvara *socket* koji izvodi pokušaj spajanja na internet. Ako je pokušaj povezivanja na internet uspješan, vrijednost istina (engl. *true*) se pridružuje klasi *LiveData* uz metodu *postValue*, a promatrač će omogućiti poziv funkcionalnosti koja je omogućena samo uz postojanje internetske povezanosti [43]. Slika 4.12. prikazuje provjeru ima li mreža mobilnog uređaja na kojem se nalazi mobilna aplikacija povezanost s internetom.

```
object DoesNetworkHaveInternet {
    fun execute(socketFactory: SocketFactory): Boolean {
        return try {
            Log.d(TAG, msg: "PINGING Google...")
            val socket = socketFactory.createSocket() ?: throw IOException("Socket is null.")
            socket.connect(InetSocketAddress( hostname: "8.8.8.8", port: 53), timeout: 1500)
            socket.close()
            Log.d(TAG, msg: "PING success.")
            true
        } catch (e: IOException) {
            Log.e(TAG, msg: "No Internet Connection. $e")
            false
        }
    }
}
```

Slika 4.12. Provjera mrežne povezanosti s internetom.

5. KORIŠTENJE I ISPITIVANJE MOBILNE APLIKACIJE

5.1. Način korištenja mobilne aplikacije

Osnovni korak prije korištenja mobilne aplikacije je postavljanje (engl. *install*) aplikacije na mobilni uređaj korisnika. Nakon postavljanja, aplikacija je spremna za korištenje. Pokretanjem aplikacije korisniku se pojavljuje zaslon učitavanja. Zaslon učitavanja omogućuje aplikaciji pozadinsko učitavanje podataka i provjeru aktivnosti sesije, odnosno je li korisnik već prijavljen u mobilnoj aplikaciji.

Ako korisnik nije prijavljen, korisniku se prikazuje zaslon prijave u središnji sustav mobilne aplikacije. Ovaj dio mobilne aplikacije naziva se autentifikacijski dio i služi za provjeru unesenih korisničkih podataka prilikom prijave. Ukoliko korisnik nema korisnički račun, ima mogućnost stvoriti račun pritiskom na gumb „*Create account"*. Zatim se korisniku prikazuje fragment s tekstualnim okvirima za unos nužnih podataka koje je potrebno ispuniti kako bi korisnik stvorio korisnički račun. Pritiskom na gumb „*Create account"* odrađuje se pozadinska logika spremanja korisničkog računa koja korisnika prosljeđuje na središnji sustav mobilne aplikacije.

Zaslon središnjeg dijela sustava mobilne aplikacije prikazuje se korisniku ako se nije prethodno odjavio, korisnik se uspješno prijavio s ispravnim korisničkim podacima te korisnik je uspješno stvorio korisnički račun putem registracijske forme. Središnji dio mobilne aplikacije odrađuje glavnu logiku izvedbe mobilne aplikacije. Korisniku se prvobitno prikazuje fragment s mogućnošću unosa osobnih podataka umrle osobe. Fragment je dio zaslona uz kojeg još dodatno postoji navigacijska traka koja omogućuje korisniku pristup ostalim funkcionalnostima mobilne aplikacije. Nakon unosa osobnih podataka o umrloj osobi, stiskom na gumb strjelice, korisnika se prosljeđuje na drugi fragment koji je zaslužan za unos podataka o smrti umrle osobe, dok se u pozadini spremaju osobni podaci umrle osobe na bazu podataka. Po završetku unosa podataka o smrti umrle osobe, korisnik pritiskom na gumb „*Save data*" izvodi njihovu pohranu te ga aplikacija ponovno prosljeđuje na prvi fragment zaslona kako bi mogao unijeti podatke za novu umrlu osobu.

Kao što je već spomenuto, navigacija omogućuje pristup ostalim funkcionalnostima unutar mobilne aplikacije. Spomenute funkcionalnosti uključuju određivanje korisnikove lokacije putem usluge *Google Maps*, vizualizirani prikaz statističke analize podataka prema demografiji i uzroku smrti, pregled unesenih podataka i njihovo ažuriranje ili brisanje, te pregled vlastitog profila i ažuriranje vlastitih korisničkih podataka. Prilikom pregleda unesenih podataka, korisnik se prosljeđuje na novi fragment ako pritisne na podatak. Novotvoreni fragment omogućuje opcije

ažuriranja osobnih podataka umrle osobe ili podataka o smrti umrle osobe. Međutim, ako se korisnik odluči za duži pritisak nad podatkom, tada će se taj podatak izbrisati. Također, korisnik ima opciju izvoza podataka u obliku XSLX datoteke i slanja iste na elektroničku poštu podrške mobilne aplikacije ako stisne na gumb *„Export and send to email“*. Nakon toga će se korisniku proslijediti datoteka na njegov prijavljeni korisnički mail.

Zaslon vizualnog prikaza statističke analize sastoji se od „pita“ dijagrama i legende koji se vizualiziraju nakon što se odabere jedan od gumbova ponuđenih u izborniku (*„Demographic*“ ili *„Causes*“). Ono prikazuje postotni udio podataka s obzirom na dani filter. Dijagram je interaktivan alat koji se može rotirati i naglašavati pojedine udjele pritiskom na njih.

Pritiskom na ikonicu mape na zaslonu središnjeg sustava, pokreće se zaslon *Google Maps*-a koji automatski određuje i postavlja marker na lokaciju korisnika. Prije samog prikaza mape, korisnik treba odobriti pristup lociranju mobilnog uređaja od strane mobilne aplikacije.

Naposljetku, korisnik ima mogućnost pregleda i ažuriranja vlastitih korisničkih podataka na zaslonu njegova profila gdje se također nalazi i gumb za odjavu iz mobilne aplikacije.

5.2. Regresijsko ispitivanje mobilne aplikacije

Regresijsko ispitivanje se definira kao proces ponovnog ispitivanja promijenjenih dijelova programske podrške kako bi se osiguralo da nisu nastale nove pogreške pri toj promjeni [44]. Tijekom ispitivanja, raspisani su ispitni slučajevi koji pokrivaju područja funkcionalnosti mobilne aplikacije, a vidljivi su u *Excel* tablici u prilogu 2.

Provedenim regresijskim ispitivanjem utvrdilo se ispravno djelovanje funkcionalnosti optimizirane verzije mobilne aplikacije, te su svi ispitni slučajevi bili uspješni.

5.3. Skupna analiza rada mobilne aplikacije

Odraz rada optimizirane verzije mobilne aplikacije za vođenje mortalitetne statistike ogleda se u analizi komponenti sustava s obzirom na izvedbu funkcionalnosti prema funkcionalnim i nefunkcionalnim zahtjevima, te kako one međusobno surađuju u sustavnoj integraciji.

Optimizacija mobilne aplikacije za vođenje mortalitetne statistike zaključuje se kao uspješna s obzirom na nekoliko ostvarenih postavki. Prva i temeljna postavka je unaprjeđeni arhitekturni predložak mobilne aplikacije – MVVM. Novim predloškom se omogućilo upravljanje i prikaz podataka s obzirom na životni ciklus *viewmodel*-a koji služi kao spona između pogleda i modela gdje se nalaze podaci. Također, predložak omogućuje jednostavan rad s klasama koje omogućuju

promatranje (engl. *observing*) kao što su *MutableLiveData* što omogućuje pozadinsko učitavanje podataka neovisno o stvaranju pogleda. Isto tako, posrednici koji dodatno pospješuju predložak i doprinose kvalitetnoj izvedbi funkcioniranja mobilne aplikacije s obzirom na nefunkcionalne, a onda u konačnici i funkcionalne zahtjeve uključuju: biblioteka za ubrizgavanje ovisnosti *Koin*, programska rješenja za asinkrono izvođenje – *coroutines* te oblikovni obrazac *repository pattern*. Posljedica implementacije ovakvih elemenata je i kvalitetnija i preglednija programska organiziranost koda koja se podijelila na temeljne i reprezentativne entitete koji predstavljaju određena semantička područja mobilne aplikacije unutar programskog koda. Regresijskim ispitivanjem se utvrdilo da arhitekturni model MVVM pospješuje mogućnost ispitivanja mobilne aplikacije te da je mobilna aplikacija lako proširivi i popravljiv produkt. Na taj su se način osigurale mogućnosti za daljnje proširivanje s obzirom na nove zahtjeve, te ažuriranje postojećeg sustava i njegovih funkcionalnosti. Proširivanjem funkcionalnosti s mogućnošću prijave i registracije, osigurala se sigurnost i personalizacija podataka s obzirom na korisnika koji može raditi s njima. Također, sigurnost i pouzdanost mobilne aplikacije upotpunile su se i validacijskim provjerama nad podacima (kao što je pozadinska provjera elektroničke adrese korisnika prilikom stvaranja korisničkog računa ili provjera prilikom unosa osobnog identifikacijskog broja umrle osobe) te upitima za pristup korištenju postavki mobilnog uređaja poput: geolociranja i pristupa internetu. Ujedno se i osigurava izvanmrežno korištenje aplikacije s ograničenim funkcionalnostima koje ne trebaju internetsku mrežu za vlastiti rad. Radi poboljšanja korisničkog iskustva, ostvarile su se i funkcionalnosti koje olakšavaju statistički dio obavljanja službe mrtvozorništva. To su: vizualni statistički prikaz analize podataka s obzirom na demografiju i uzrok smrti, te izvoz i slanje podataka na elektroničku poštu u obliku XSLX datoteke (*Excel* tablice). Osim podrške u statističkom dijelu mobilne aplikacije, razvijena je i podrška na razini ažuriranja podataka umrle osobe i korisnikovih podataka gdje korisnik ima mogućnost obnavljanja lozinke ako ju je zaboravio ili ažuriranja korisničke elektroničke adrese. Nadalje, ažurirani su implementacijski paketi i ovisnosti na suvremenu i stabilnu verziju kako bi mobilna aplikacija bila u skladu s najnovijim mogućnostima koje oni pružaju. Naposljetku, ostvario se redizajn mobilne aplikacije s obzirom na estetski izgled kako bi se mobilna aplikacija tematski kvalitetnije prilagodila svrsi obavljanja djelatnosti mrtvozorništva, a i olakšao rad korisniku.

6. ZAKLJUČAK

Današnji čovjek živi u svijetu gdje postoji mnoštvo mogućnosti, prilika i opcija koje se nalaze u raznim granama njegova postojanja i djelovanja. Kada postoji mnoštvo mogućnosti, stvara se težnja za optimumom, a prema zakonu prirode, ljudske aktivnosti su uvijek usmjerene prema najboljem mogućem izboru. Taj put prema tom izboru i optimumu omogućuje optimizacija.

Ovim diplomskim radom ostvarena je optimizirana *Android* mobilna aplikacija za vođenje mortalitetne statistike. Primarni cilj postupka optimizacije mobilne aplikacije ogleda se u poboljšanju načina korištenja, mogućnosti ispitivanja i daljnjeg proširivanja, te ažuriranja i nadogradnje postojećeg sustava novim funkcionalnostima. Ovi izazovi uspješno su riješeni implementacijom novog arhitekturnog predloška MVVM. U kombinaciji s navedenim predloškom, upotrijebljeni su potporni alati poput: *Koin* i *LiveData* koji pospješuju postojanje predloška. Uređena je programska organiziranost koda prema oblikovnom obrascu *repository pattern* kako bi se omogućila čitkost i razumljiv daljnji razvoj nad aplikacijom. Aplikacija je razvijena u programskom jeziku *Kotlin* koji je olakšao razvoj svojim suvremenim predefiniranim metodama i *lambda* funkcijama. Isto tako je izgled korisničkog sučelja razvijen u programskom alatu FIGMA koja pruža interaktivna i kreativna sučelja za rad na dizajnu aplikacije. Reciklirane su postojeće funkcionalnosti prijašnje verzije mobilne aplikacije te su nadograđene novima poput: mogućnosti za prijavu, registraciju i odjavu korisnika, ažuriranje i brisanje različitih vrsta podataka, izvoz podataka i njihovo slanje na elektroničku poštu podrške.

Rezultati optimizacije mobilne aplikacije za vođenje mortalitetne statistike ogledaju se u kvalitetnoj izvedbi i zajedničkom djelovanju funkcionalnih i nefunkcionalnih zahtjeva na programsku podršku. Osigurana je postojanost određenih funkcionalnosti aplikacije ako aplikacija radi u izvanmrežnom stanju čime se povećala pouzdanost mobilne aplikacije. Upotrebljivost mobilne aplikacije se također povećala uvođenjem novih funkcionalnosti od kojih je najbitnija mogućnost rukovanja podacima unutar aplikacije. Uvođenjem mogućnosti prijave i registracije sustav je postao sigurniji, a podaci zaštićeni i personalizirani za točno određenog korisnika.

Naposljetku, odrađenim regresijskim ispitivanjem postojećih i novih funkcionalnosti se utvrdio ispravan rad optimizirane verzije mobilne aplikacije, ali i mogućnosti za proširivanje i poboljšanje u skladu s zahtjevima korisnika i tehnološkim napretkom svakodnevnice, odnosno vječna optimizacija.

LITERATURA

[1] C., A., Floudas, P., M., Pardalos, „Encyclopedia of Optimization“, sv. 2, New York: Springer, 2009.

[2] F., McKelvey, J., Neves, „Introduction: optimization and its discontents, Review of Communication“, sv. 2, str. 95-112, lip. 2021., dostupno na: https://www.tandfonline.com/doi/full/10.1080/15358593.2021.1936143 [6.6.2023.]

[3] D., Ferreira da Silva, „Globality Critical Ethnic Studies1“, sv. 1-35 original emphases, 2015.

[4] M., Christensen,L.,P.,E., Yunker, F., Adedeji, et al., „Data-science driven autonomous process optimization. Commun Chem“, sv. 4, str. 112 , 2021., dostupno na: https://www.nature.com/articles/s42004-021-00550-x [6.6.2023.]

[5] M., de Jonge, „Source Tree Composition. In: Gacek, C. (eds) Software Reuse: Methods, Techniques, and Tools“, „ICSR Lecture Notes in Computer Science“, sv. 2319, Springer, Berlin, Heidelberg, 2002., dostupno na: https://link.springer.com/chapter/10.1007/3-540-46020-9_2 [9.6.2023.]

[6] A., Wilson, F., Wedyan, S., Omari, „An Empirical Evaluation and Comparison of the Impact of MVVM and MVC GUI Driven Application Architectures on Maintainability and Testability," u *2022 International Conference on Intelligent Data Science Technologies and Applications* (IDSTA), str. 101-108, San Antonio, 2022.

[7] T., Weise, M., Zapf, R., Chiong, A., Nebro, „Why Is Optimization Difficult?“, 2009.

[8] Google, „Google Home“, dostupno na: https://home.google.com/the-latest/#google-home-app [Pristupljeno 18.6.2023.]

[9] Discord, „Discord Blog“, dostupno na: https://discord.com/blog/redesigning-the-discord-overlay [Pristupljeno 18.6.2023.]

[10] Eleken, „UI/UX redesign for a client experience platform“, dostupno na: https://www.eleken.co/cases/gridle [6.8.2023.]

[11] Uber, „Uber“ dostupno na: https://www.uber.com/hr/hr/ [6.8.2023.]

[12] UXMagazine, „How to Redesign an App: When to Do It and What to Start With“, dostupno na: https://uxmag.com/articles/how-to-redesign-an-app-when-to-do-it-and-what-to-start-with [6.8.2023.]

[13] Medium, „Sonia Francu – blog“, dostupno na: https://sfrancu.medium.com/ [6.8.2023.]

[14] S., Mihel, V., Stamenić, M., Popović, V.,Petrovečki, D., Mayer, „Priručnik o popunjavanju potvrde o smrti“, Ministarstvo zdravstva i socijalne skrbi, Hrvatski zavod za javno zdravstvo, Hrvatska, pro. 2011.

[15] A., Ray, C., Ackermann, R., Cleaveland, C., Shelton, C., Martin, Chapter 6 „Functional and Nonfunctional Design Verification for Embedded Software Systems“, sv. 6. u M. V. Zelkowitz, „Advances in Computers“, Elsevier, sv. 83, str. 277-321, 2011., dostupno na: https://www.sciencedirect.com/science/article/abs/pii/B9780123855107000060 [19.6.2023.]

[16] M., A., Haque, M., A., Rahman, M., S., Siddik, „Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study," u *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, sv. 1-5, Dhaka, Bangladeš, 2019.

[17] H.-T., Jung, G.-H., Lee., „A systematic software development process for non-functional requirements," u *International Conference on Information and Communication Technology Convergence (ICTC)*, sv. 431-436, Jeju, Južna Korea , 2010., dostupno na: https://ieeexplore.ieee.org/document/5674806 [19.6.2023.]

[18] S., Meskini, A., Nassif, L., Capretz, „Reliability Models Applied to Mobile Applications. Proceedings“ u *7th International Conference on Software Security and Reliability Companion, SERE-C*, 2013., dostupno na: https://www.researchgate.net/publication/259232374_Reliability_Models_Applied_to_Mobile_Applications [20.6.2023.]

[19] S., Swaid, „Usability of Mobile Apps: An Integrated Approach.“, 2017., dostupno na: https://www.researchgate.net/publication/312121456_Usability_of_Mobile_Apps_An_Integrated_Approach [20.6.2023.]

[20] J., Gao, X., Bai, W.-T., Tsai, T., Uehara, „Mobile Application Testing: A Tutorial.Computer“, sv. 2, str. 46-55, California, Sjeverna Amerika, 2014.

[21] R., Roth, „User Interface and User Experience (UI/UX) Design. Geographic Information Science & Technology Body of Knowledge.“, 2017., dostupno na: https://www.researchgate.net/publication/317660257_User_Interface_and_User_Experience_UIUX_Design [21.6.2023.]

[22] N., Samrgandi, „User Interface Design & Evaluation of Mobile Applications.“, str. 55-63, 2021.

[23] FIGMA, „Figma Learn“, dostupno na: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma- [6.8.2023.]

[24] Investopedia, „Android Operating System (OS): Definition and How It Works“, dostupno na: https://www.investopedia.com/terms/a/android-operating-system.asp [8.8.2023.]

[25] GeeksForGeeks, „Android SDK and it's Components“, dostupno na: https://www.geeksforgeeks.org/android-sdk-and-its-components/ [8.8.2023.]

[26] ELPROCUS, „What is an Android Operating System & Its Features“, dostupno na: https://www.elprocus.com/what-is-android-introduction-features-applications/ [8.8.2023.]

[27] Android Developers, „Meet Android Studio“, dostupno na: https://developer.android.com/studio/intro/ [8.8.2023.]

[28] Developer Mozilla, „XML introduction“, dostupno na: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction [8.8.2023.]

[29] GeeksForGeeks, „A Complete Guide to Learn XML For Android App Development“, dostupno na: https://www.geeksforgeeks.org/a-complete-guide-to-learn-xml-for-android-app-development/ [8.8.2023.]

[30] Kotlin, „Kotlin for Android“, dostupno na: https://kotlinlang.org/docs/android-overview.html [8.8.2023.]

[31] Medium, „What is Firebase? The complete story, abridged.“, dostupno na: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 [8.8.2023.]

[32] Aritmetrics, „Google Maps“, dostupno na: https://www.arimetrics.com/en/digital-glossary/google-maps [8.8.2023.]

[33] InsertKoin, „Koin“, dostupno na: https://insert-koin.io/ [12.9.2023.]

[34] Developers, „LiveData“, dostupno na: https://developer.android.com/topic/libraries/architecture/livedata [12.9.2023.]

[35] A. Syromiatnikov, D. Weyns, „A Journey through the Land of Model-View-Design Patterns“ u *2014 IEEE/IFIP Conference on Software Architecture,* sv. 21-30, Sydney, Australija, dostupno na: https://ieeexplore.ieee.org/document/6827095 [13.8.2023.]

[36] Microsoft, „Introduction to Model/View/ViewModel pattern for building WPF apps“, dostupno na: https://learn.microsoft.com/en-us/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps [10.8.2023.]

[37] C., Anderson, „The Model-View-ViewModel (MVVM) Design Pattern“, u *Pro Business Applications with Silverlight 5,* sv. 461-499, Berkeley, Apress, 2012.

[38] Firebase, „Firebase Authentication“, dostupno na: https://firebase.google.com/docs/auth/#How%20Does%20It%20Work? [13.8.2023.]

[39] M., Phill, „PieChart, Class PieData“, dostupno na: http://localhost:63342/sze2vaa1tn9isdfmukthdzmnpcptdcj76lnre/MORT/MPAndroidChart-v3.1.0-javadoc.jar/com/github/mikephil/charting/data/PieData.html [10.9.2023.]

[40] Apache, „XSSFWorkBook“, dostupno na: http://localhost:63342/sze2vaa1tn9isdfmukthdzmnpcptdcj76lnre/MORT/poi-ooxml-5.2.3-javadoc.jar/org/apache/poi/xssf/usermodel/XSSFWorkbook.html#%3Cinit%3E() [10.9.2023]

[41] JavaMail, „JavaMail Reference Implementation“, dostupno na: https://javaee.github.io/javamail/ [10.9.2023.]

[42] Android Developers, „Monitoring connectivity status and connection metering“, dostupno na: https://developer.android.com/training/monitoring-device-state/connectivity-status-type#:~:text=The%20ConnectivityManager%20provides%20an%20API%20that%20enables%20you,that%20include%20device%20capabilities%20and%20data%20transport%20options. [17.9.2023.]

[43] CodingWithMitch, „CodingWithMitch.com“, dostupno na: https://codingwithmitch.com/ i https://github.com/mitchtabian/food2fork-compose/tree/master [17.9.2023.]

[44] K., K., Aggarwal, Y. Singh, „Software Engineering Programs Documentation, Operating Procedures“, u *New Age Internationl Publishers, Revised Second Edition,* 2005.

SAŽETAK

Optimizacija je vječni alat kojim se svijet svakodnevno unaprjeđuje i napreduje na različitim razinama. Ona se ostvaruje i vrijedi za svaku pojavu pa tako i za tehnološku. Cilj ovog diplomskog rada je predstaviti raspon mogućnosti i ishoda optimizacije *Android* mobilne aplikacije. Na temelju analize i problematike prijašnjeg rješenja mobilne aplikacije, predstavljeno je novo idejno rješenje mobilne aplikacije koje se temelji na arhitekturnom predlošku MVVM (*Model-View-ViewModel)*. Kroz diplomski rad, objašnjene su postojeće i reciklirane funkcionalnosti prijašnje verzije mobilne aplikacije, ali i proširene i nove mogućnosti korištenja mobilne aplikacije kao što su prijava, odjava i registracija korisnika; izvoz i slanje podataka u obliku XSLX datoteke te ažuriranje pohranjenih podataka unutar aplikacije. Novi arhitekturni predložak optimizirane verzije mobilne aplikacije osnažen je i suvremenim alatima poput: biblioteke za ubrizgavanje ovisnosti *Koin,* klasa za praćenje životnog ciklusa podataka *LiveData* i programskih rješenja za asinkrono izvođenje *coroutines*. Regresijskim ispitivanjem se ocijenio rad optimizirane verzije mobilne aplikacije.

Ključne riječi: mobilna *Android* aplikacija, mortalitetna statistika, optimizacija, predložak programske arhitekture MVVM

ABSTRACT

Title: **Android mobile application optimization**

Optimization is an eternal tool that daily enhances and pushes the world forward in variety areas. It affects every instance which also includes technology. The prime goal of this master thesis is to bring forward the range of possiblities and outcomes that comes with optimization of mobile *Android* application. Based on the evaluation and analysis of the problem which is contained in previous solution, new ideal solution is put forward which is fortified with sovereign architecture pattern MVVM (*Model-View-ViewModel*). Throughout the thesis, all of the previous and reused functionalities are complemented with new features such as user sign in, registration and sign out, data export in XSLX file which is being sent to the central application email, and update of the data in application. Fresh architecture pattern MVVM for optimized version of mobile application is additionally reinforced by modern tools that includes: dependency injection library *Koin*, specialized class for tracking data lifecycle *LiveData* and asyncronous programme solution *coroutines*. The summary of completed solution is evaluated by doing regression testing.

Keywords: architecture pattern MVVM, mobile *Android* application, mortality rate, optimization

ŽIVOTOPIS

Fran Zamaklar rođen je 15. siječnja 1999. godine u Zagrebu. Pohađao je Osnovnu školu Ivana Kukuljevića u Belišću. Upisao je Opću gimnaziju u Srednjoj školi Valpovo te ju završava 2018. godine. Iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, prijediplomski sveučilišni studij Računarstvo koji završava 2021. godine. Po završetku prijediplomskog studija, upisuje diplomski sveučilišni studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija, izborni blok Programsko inženjerstvo.

_____

Fran Zamaklar

PRILOZI

[1] Poveznica na FIGMA repozitorij: https://www.figma.com/file/0Qvk20iEfVd4zLmYwnH3hy/MORT-mobile-application?type=design&node-id=0%3A1&mode=design&t=BK5uc3r4iOWTfp8v-1

[2] XSLX datoteka – „MORT App – Regression testing“

[3] Poveznica na Github repozitorij: https://github.com/franzamaklar/MORT