Mobilna Android aplikacija za vođenje mortalitetne statistike

Zamaklar, Fran

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

Sveučilišni preddiplomski studij Računarstva

MOBILNA ANDROID APLIKACIJA ZA VOĐENJE MORTALITETNE STATISTIKE

Završni rad

Fran Zamaklar

Osijek, 2021.

SADRŽAJ

1. UVOD

Mortalitet je svakodnevna pojava. To je neminovni ishod ljudskog života. Budući da se javlja u kontinuitetu i bezvremenski je prisutan, javlja se potreba za statističkim vođenjem njegove promjene stanja i oblika. Analitički gledano, ono može utjecati na različite društvene, humanističke, psihološko-emocionalne i financijsko-političke aspekte ljudskog života. Može biti pojava izumiranja civilizacije, pada valute i drugih resursa, a negdje buđenje apatije i distopije. Premda je taj proces nezaustavljiv, statistika smrtnosti ljudi daje uvid u ono kako poboljšati kvalitetu ljudskog života i tako spriječiti nepotrebne gubitke.

Cilj ove mobilne Android aplikacije je omogućiti korisniku lakši unos, prikaz i praćenje podataka o mortalitetu. Na temelju unesenih podataka o preminulim osobama želi se prikazati statistička projekcija umrlih po određenim parametrima. Konačni podatci iz baze podataka prikazat će se u tablici.

U drugom poglavlju objasnit će se problematika mortalitetne statistike, njezine značajke i obilježja, postojeća programska rješenja te rješenje predložene mobilne aplikacije. U trećem poglavlju opisat će se model vlastite mobilne aplikacije za vođenje mortalitetne statistike. Prikazat će se svi parametri unosa, struktura baze podataka, nefunkcionalni i funkcionalni zahtjevi, dizajn te struktura aplikacije. U četvrtom poglavlju predstavit će se programska izvedba mobilne aplikacije. Naposljetku slijedi prezentacija ispitivanja i korištenja mobilne aplikacije, kao i ispitni slučajevi i analiza rada aplikacije.

1.1. Zadatak završnog rada

U završnom radu treba proučiti i opisati probleme i zahtjeve aktivnosti vezanih za vođenje mortalitetne statistike (mrtvozorništvo). Nadalje, na temelju događaja, uvjeta i parametara na terenu, treba definirati model provedbe aktivnosti, unos, pohranu, statističku sintezu i analizu podataka na određenom području i u određenom vremenskom razdoblju, te stvaranje izvješća i ispis rezultata postupka vođenja mortalitetne statistike. Također, potrebno je definirati arhitekturu aplikacije, opisati programske tehnologije, razvojnu okolinu i statističke postupke potrebne za njeno programsko ostvarenje. U praktičnom dijelu rada, treba dizajnirati i programski ostvariti mobilnu aplikaciju s bazom podataka, ugraditi navedeni model provedbe aktivnosti. Mobilna aplikacija treba omogućiti praćenje mortalitetne statistike na temelju podataka koji su strogo propisani odgovarajućim pravilnikom u sustavu javnog zdravstva. Mobilnu aplikaciju potrebno je ispitati i analizirati za primjerene načine korištenja, događaje i ulazne podatke.

2. PROBLEMATIKA VOĐENJA MORTALITETNE STATISTIKE I STANJE U PODRUČJU

2.1. Mortalitetna statistika

2.1.1. Definicija

Referirajući se na [1], mortalitetna statistika zrcalna je slika zdravstvene prevalencije populacije. S obzirom na kvalitetu zdravlja i zdravstvenog sustava, smrt je daleko najteža posljedica koja postoji. Shodno tome, mortalitetna statistika prikazuje koji su svi uzroci koji dovode do smrti što je zapravo i najveći problem svakog zdravstvenog sustava.

2.1.2. Značajke i obilježja

Državni zavod za statistiku (DZS) u Republici Hrvatskoj nositelj je najvećeg dijela znanstvenih istraživanja o mortalitetu, a Hrvatski zavod za javno zdravstvo operativno je tijelo koje provodi javnozdravstvena mortaliteta istraživanja. Evidencija upisa umrlih osoba isključivo je propisana Zakonom o državnim maticama (NN 96/93) i umrle osobe se upisuju u propisane registre odnosno očevidnike umrlih – matice umrlih. Postupak upisa umrlih osoba započinje izdanom Potvrdom o smrti koja je propisana Pravilnikom o obrascu potvrde o smrti (NN 46/11) koju izdaje mrtvozornik nakon obavljenog pregleda umrle osobe. Što se tiče opisa ostalih obrazaca, Zakon o zdravstvenoj zaštiti (NN 150/08, 155/09, 71/10, 139/10, 22/11, 84/11) i Pravilnik o načinu pregleda umrlih i utvrđivanju vremena, mjesta i uzroka smrti (NN 46/11) propisuju precizan izgled i fond potrebnih podataka obrazaca kao što su iskaznica mrtvozornika te očevidnik o obavljenim pregledima. Putem Statističkog izvješća o smrti prikupljaju se statistički podatci o umrlim osobama koji se koriste u razne znanstvene svrhe, za strategiju demografskog razvoja kao i za praćenje broja umrlih osoba kroz određeno vremensko razdoblje. Potvrda o smrti važan je dio Statističkog izvješća jer se u nju unose utvrđeni uzroci smrti za koje onda postoje posebno propisane šifre čiji zbroj na kraju daje podatke važne za izvješće. U sastavu Službe za epidemiologiju u Republici Hrvatskoj nalazi se Odjel za mortalitetnu statistiku koji sastavlja spomenuta izvješća [2].

2.1.3. Važnost

Prema navođenju [3], za Svjetsku zdravstvenu organizacija (SZO) mortalitetna statistika predstavlja iznimno značajan izvor zdravstvenih podataka. Stopa smrtnosti jedan je od ključnih pokazatelja zdravstvenog stanja populacije i važna je u kreiranju zdravstvene zaštite ljudi na određenom području. Pokazatelj stope smrtnosti ujedno ukazuje na nizak standard života ljudi koji

se reflektira na socijalnu neujednačenost, nisku stopu obrazovanja ljudi koji nemaju odgovarajuću zdravstvenu zaštitu i više umiru.

2.1.4. Službeni obrazac evidentiranja preminule osobe

Potvrda o smrti je službeni je obrazac evidentiranja preminule osobe. Potvrda se sastoji od dva dijela. Uvodni dio vezan je uz identifikaciju osobe i njezine osobne podatke, dok je drugi dio izvješće o uzroku smrti. Kako izgleda Potvrda o smrti [4], može se vidjeti na obrascu u PDF obliku na stranici 16.

2.1.5. Uzroci smrti

Prema [4], postoje razni uzroci smrti koji se primarno dijele na prirodne, iznenadne prirodne i nasilne smrti. Prirodne se smrti definiraju sukladno s vremenskim trajanjem bolesti umrle osobe te se ovisno o tome klasificiraju na iznenadne ili dugotrajne. Pod iznenadne smrti ubrajaju se: bolesti srčano-žilnog sustava (ateroskleroza, upala srčanog mišića, srčani zastoj, bolesti zalistaka, bolesti hipertenzije srca…), središnjeg živčanog sustava (krvarenje u mekim moždanim ovojnicama, ili tkivu, meningitis, novotvorine, epilepsija…), dišnog sustava (astma, tromboembolija, tuberkuloza, novotvorine, upala pluća…), probavnog sustava (ciroza jetre, perforacija crvuljka) i ostale (rijetke bolesti – maligni karcinomi, akutna upala gušterače…). Nasilne smrti dolaze s uvjetom da je uzrok smrti bio nasilan moment koji se izvršio nad umrlom osobom.

2.1.6. Izvješće o smrtnosti prema listi odabranih uzroka smrti

Na [3] prikazuje se cjelokupni statistički pregled mortalitetne statistike u Republici Hrvatskoj prema podatcima iz liste odabranih uzroka smrti tijekom 2019. godine. Korištena dokumentacija izvorni su podatci Državnog zavoda za statistiku za 2019. godinu. Podatke je prikupio i obradio Hrvatski zavod za javno zdravstvo u 2020. godini za prethodnu godinu.

2.2. Prikaz postojećih rješenja i stanje u području za vođenje mortalitetne statistike

2.2.1. Analiza stanja u području

Budući da se svakodnevno evidentiraju i ažuriraju podatci o mortalitetu, dokumentacija i papirologija se neminovno gomila, pa se pojavila potreba za rasterećenjem ljudskog rada u tom području intervencijom digitalizacije. Izvješća se podnose tromjesečno i sustav akumulira podatke na županijskoj razini.

2.2.1.1. Glavni izazovi

Glavni izazovi mortalitetne statistike su prikupljanje, održavanje i praćenje podataka. Prikupljanje se odvija pisanim načinom, odnosno ispunjava se službena dokumentacija na mjestu smrti. Zatim te podatke mrtvozornik održava samostalno, a nakon tromjesečnog razdoblja ih podnosi Hrvatskom zavodu za javno zdravstvo. Dokumentacija iziskuje vremensku i financijsku potporu što znači za svaki novi unos novo vremensko odricanje, za svako tromjesečno razdoblje, novi trošak. Praćenje podataka također ovisi o dokumentaciji budući da se mora proanalizirati svaki unos na papiru čime se troši vremenski resurs.

2.2.1.2. Računalna rješenja na postojeće izazove u smislu vrsta aplikacija i postupaka vođenja statistike

Računalne, odnosno informacijsko-komunikacijske tehnologije u širem smislu omogućuju smanjenje ljudskog rada na način da se smanji postojeća dokumentacija koja bi se zamijenila digitalnim sustavom. Povećala bi se učinkovitost, brzina i pristupačnost podatcima, rasteretio ljudski rad i omogućilo zaposlenicima više slobodnog vremena.

2.2.2. Postojeća programska rješenja

2.2.2.1. Cause of Death Mobile Application

Mobilna aplikacija *Cause of Death* je aplikacija koja služi kao skup brzih instrukcija napravljenih za vođenje mortalitetne statistike između kojih je i kako ispuniti izvješće o Potvrdi o smrti. Dizajnirala ju je američka vladina agencija za prevenciju i kontrolu bolesti (CDC – *Centar for Disease Control and Prevention*) [5].

2.2.2.2. WISQARS Mobile for Phone

Također, aplikaciju WISQARS stvorila je CDC agencija, no ova mobilna aplikacija pronalazi svoju upotrebu u prikazu sveukupnog broja smrti prema ozljedama u Sjedinjenim Američkim Državama. Tako se može dobiti ispis deset vodećih uzroka smrti u Sjedinjenim Američkim Državama, statistika smrtnosti prema spolu, nacionalnosti (odnosno prema federalnim državama), lokaciji, dobnim i etničkim skupinama [6]. Sučelje mobilne aplikacije WISQARS nalazi se na slici 2.1.

Slika 2.1. : Korisničko sučelje mobilne aplikacije WISQARS

2.3. Idejno rješenje vlastite mobilne aplikacije za vođenje mortalitetne statistike

Izrada vlastite mobilne aplikacije za vođenje mortalitetne statistike motiviran je primarno manjkom sličnih aplikacija u tom polju i željom za povećanjem učinkovitosti službe mrtvozorništva kao uslužnog posla. Korisnik unosi parametre osnovnih podataka osobe, kao i detalje smrti iste osobe koji se spremaju u bazu podataka. Na temelju tih podataka bit će prikazana cjelokupna mortalitetna struktura unosa preko grafova i legendi. Također će korisnik imati priliku pregleda svojih unosa te vlastitog lociranja kako bi si olakšao unos mjesta smrti. Poglavlje 3 detaljnije objašnjava prijedlog modela mobilne aplikacije i njezine funkcionalnosti.

3. MODEL MOBILNE APLIKACIJE ZA VOĐENJE MORTALITENE STATISTIKE

3.1. Nužni podatci za vođenje mortalitetne statistike

Nužni podatci za vođenje mortalitetne statistike dijele se u dva dijela: opći podatci i specifikacije smrti. Pod opće podatke podrazumijevaju se podatci koji identificiraju osobu, odnosno esencijalni pojmovi koji pripadaju svakoj osobi. To su: ime, prezime, spol i datum rođenja osobe. Uz njih se vežu podatci o smrti osobe. U njih se ubrajaju: datum, vrijeme, mjesto i uzrok smrti. Ime, prezime i mjesto smrti unose se tekstualnim unosom, odnosno korisnik ih zapisuje izravno na njih predviđeno mjesto. Datum, vrijeme i uzrok smrti, te datum rođenja osobe unose se preko padajućih izbornika i kalendara. Strukturni oblik ovih nužnih podataka za radnu okolinu aplikacije postigao se u suradnji sa zahtjevima korisnika aplikacije.

3.2. Postupak vođenja mortalitetne statistike

Postupak vođenja mortalitetne statistike sastoji se od unosa podataka, tromjesečnog praćenja istih podataka i konačnog podnošenja izvješća o mortalitetu iz baze podataka. Kako navodi [4], mortalitetna statistika je temelj epidemioloških istraživanja, provodi istraživanja o uzrocima smrti prikupljenim različitim demografskim podatcima. Njeni podatci su ključni pokazatelji za istraživanja o povijesti bolesti te procjeni svih medicinskih metoda koje se provode radi poboljšanja zdravlja. Prije podnošenja izvješća bitno je obaviti točan i potpun unos svih podataka, kao i pregled tih podataka. Na kraju treba sagledati i izvršiti analizu statističkih grafova koji nam prikazuju pregled zdravstvenog stanja u društvo i pokazuju što činiti.

3.3. Funkcionalni zahtjevi mobilne aplikacije

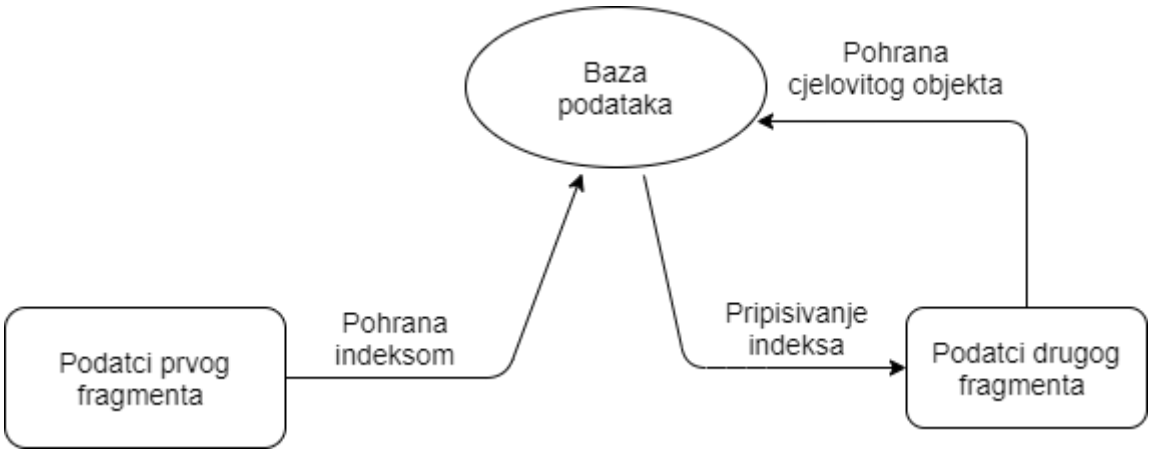
Funkcionalni zahtjevi opisuju očekivane usluge, ono što bi sustav trebao raditi ili se od njega očekuje [7]. Naglasak je na pojedinačnim ponašanjima sustavima koji vode uspješnosti cjelokupnog programa.

3.3.1. Unos podataka

Unos podataka odvija se preko korisničkog sučelja i ono predstavlja temelj mobilne aplikacije. Korisnik unosi sedam nužnih podataka. To su označni podatci poput imena, prezimena, spola i datuma rođenja, te podatci posebnog značenja kao što su datum, vrijeme, mjesto i uzrok smrti. Podatci se unose preko XML komponenti: *EditText, Spinner, DatePicker, TimePicker* i *RadioButton* ubrizganih u aplikaciju pomoću *Java* programskog jezika.

3.3.2. Pohrana podataka

Uneseni se podatci mrežno pohranjuju na bazu podataka koja se nalazi na Internetu. Pohrana se odvija u dva dijela. Prvi se dio podataka pohranjuje iz prvog fragmenta (*Opći podatci*), a zatim se drugi dio podataka nadovezuje na njih iz drugog fragmenta (*Specifikacije smrti*). Ovaj način pohrane podataka ostvario se povezivanjem ključeva (indeksa) spremljenih podataka. Dohvaća se indeks iz baze podataka koji se spremio kao redni broj spremljenih podatka prvog fragmenta te se podatci iz drugog fragmenta pripisuju odgovarajućem objektu tog određenog indeksa. Nužno je napomenuti kako je bitno ostvariti posredni objekt (model) koji će znati koji će se pripadajući podatci iz prvog fragmenta povezati s podatcima iz drugog fragmenta. Naposljetku se svi podatci zajedno spremaju pod jedan objekt baze podataka. Proces pohrane podataka vidljiv je na slici 3.1.



Slika 3.1. : Dijagram tijeka pohrane podataka u bazu podataka

3.3.3. Određivanje lokacije putem *Google Maps*

Određivanje lokacije putem *Google Mapsa* ostvaruje se pomoću API poziva koji su nužni da bi korisnik mogao dobiti vlastitu lokaciju što preciznije i točnije. API pozivi bit će detaljnije objašnjeni u nadolazećem potpoglavlju 4.3.3. Korisnik se locira tako što pritisne tipku *Pogledaj mapu* na fragmentu *Specifikacije smrti*. Pored API poziva, obavezno je implementirati i dopuštenja poput: dopuštenje mrežnom povezivanju te finom i grubom lociranju. Lociranje je uspješno ako se pojavi zeleni marker s korisnikovom lokacijom.

3.3.4. Prikaz pohranjenih podataka

Kako bi korisnik imao što bolje korisničko iskustvo, bitan dio funkcionalnosti aplikacije jest i prikaz unesenih, odnosno pohranjenih podataka. To je posebni dio aplikacije i prikazan je u zasebnoj aktivnosti aplikacije. Nakon što se aplikacija povezala s bazom podataka, pritiskom na gumb *Rezultati* na fragmentu *Specifikacije smrti*, korisnik dobiva listu pohranjenih podataka.

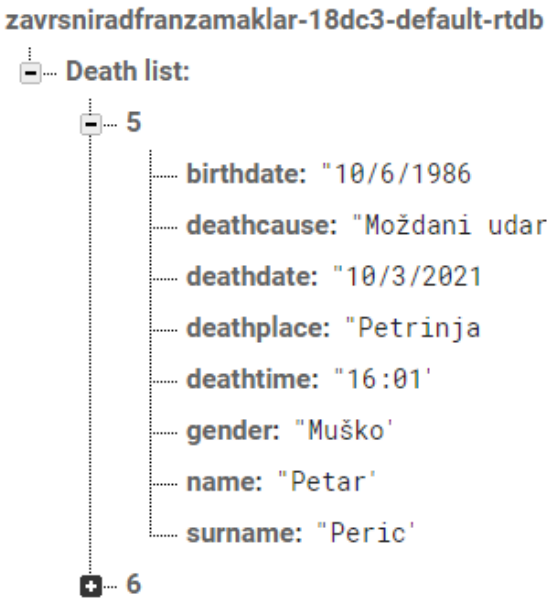
Podatci se drže unutar XML komponente *RecyclerView* koji zajedno s vlastitim adapterom omogućava mjesto u „listi“ svakom podatku i njegovim atributima.

3.3.5. Prikaz statističke analize podataka

Uz pohranjene podatke, postoji i još jedna esencijalna funkcionalnost aplikacije koja je bitna stavka rada aplikacije – prikaz statističke analize podataka. Zajedno s pregledom pohranjenih podataka, može se na istoj aktivnosti vidjeti i grafovi koji predstavljaju statističku obradu podataka. Postoje dvije statističke analize: specijalna statistika i statistika umrlih po određenom mjesecu. Klasifikacija specijalne statistike dijeli se na statistiku po spolu i uzroku smrti dok vremenski obrađena statistika predstavlja cjelokupnu sliku mortaliteta kroz godišnje razdoblje, kao i po pojedinim mjesecima. Statistike se oslikavaju grafovima (*BarChart*) uz njihove legende i svojstvene vrijednosti. Nakon toga se tim vrijednostima mogu analizirati željeni uzorci podataka. Korištena metoda za obradu podataka je metoda uvećanja, odnosno kada se u bazi podataka pronađe željeni podatak koji zadovoljava uvjet inkrementiranja, dodaje se u skup i naposljetku se dobiva ogledna slika stanja mortaliteta.

3.3.6. Struktura baze podataka

Slika 3.2 prikazuje strukturu baze podataka. Baza se ostvaruje uslugom *Firebase Realtime Database*. Subjekt *Death list* predstavlja tablicu koja drži druge entitete, odnosno svojstvene indekse po kojima se raspoređuju podatci u bazu podataka. Tako se za svaki novi korisnikov unos pripisuje novi indeks. Imena atributa određena su objektom i tako se pohranjuju uz vrijednosti.



Slika 3.2. : Struktura baze podataka u *Firebase*

3.4. Nefunkcionalni zahtjevi na mobilnu aplikaciju

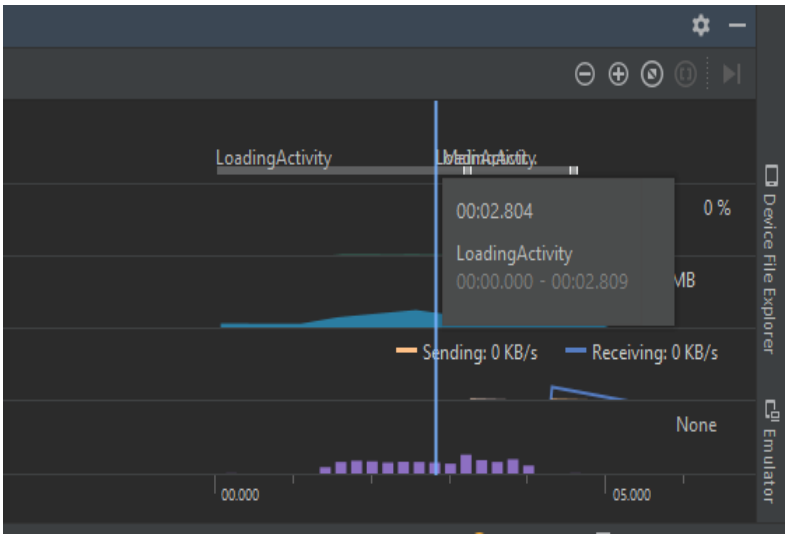
Nefunkcionalni zahtjevi definiraju se kao temelj pri izradi dizajna aplikacije jer određuju specifikacije proizvoda koje će biti prezentirane. Oni nameću ograničenja i objedinjuju svojstva kojima će sustav funkcionirati. U njih se ubrajaju: mogućnost korištenja (engl. *usability*), promjene (engl. *modifiability*), održivosti performansa (engl. *maintainability*), sigurnosni zahtjevi (engl. *reliability, security*), skalabilnosti (engl. *flexibility*), podudarnosti i prenosivosti (engl. *adaptability*) [8].

U sljedećim potpoglavljima bit će obrađeni nefunkcionalni zahtjevi na mobilnu aplikaciju za obradu i vođenje mortalitetne statistike.

3.4.1. Performanse

Izvedba aplikacije, kao i njezine odlike u brzini i aktivnosti mogu se odrediti prema vremenu odziva, odnosno vremenu kojeg aplikaciji treba da izvrši određeni zadatak. Prilikom pokretanja, aplikacija se ne bi trebala učitavati više od tri sekunde za prikaz početnog zaslona. Također korisnik ne bi trebao imati nikakve smetnje niti vremenske odgode prilikom unosa podataka.

Android Profiler usluga je *Android Studioa* koja omogućuje praćenje performansa aplikacije. Na slici 3.3 može se primijeniti kako je vremenski odziv pokretanja aplikacije do učitavanja početnog zaslona nešto manji od tri sekunde što je u granicama dobre kvalitete aplikacije. Ujedno je i potrošnja memorije (zelena boja) i energije (ljubičasta boja) vrlo malena što je u skladu sa zahtjevima aplikacije



Slika 3.3. : Obrada izvedbe aplikacije *Android Profilerom*

3.4.2. Sigurnosni zahtjevi

Aplikacija bi trebala biti osigurana minimalnim potrebama kako bi se sačuvao njezin integritet od vanjskih, a i unutarnjih narušavanja. Također se na taj način osigurava njezina pouzdanost da će funkcionalno obraditi zahtjeve koje se pred nju postavljaju.

Mobilna aplikacija za vođenje mortalitetne statistike koristi razne upite korisnika za dopuštenja prema uređaju. To su dopuštenje za dohvaćanje internetske veze, korištenje geografske lokacije uređaja. No postoje i dopuštenja koja su navedena u samoj aplikaciji kao što su: potreba za cjelovitim unosom podataka za pohranu u bazu podataka, povezivanje na bazu podataka *Firebase,* nemogućnost preuzimanja podataka zbog nedostatka podataka iz baze.

3.4.3. Prenosivost i podudarnost

Pod prenosivost i podudarnost aplikacije podrazumijeva se kompatibilnost aplikacije s ostalim uređajima i njezina jednolika i konstantna izvedba na svakom od njih pojedinačno. Također, prema [9], temeljna gradivna jedinica programske prenosivosti je aplikacijsko programsko sučelje (*Application Programming Interface* – API). API predstavlja odnos klijenta s dobavljačem željene usluge. Tako kada više dobavljača odluče implementirati iste API značajke, korisnici tih značajka mogu premještati program s jedne platforme na drugu.

3.4.4. Ostali nefunkcionalni zahtjevi

Pored izvedbe, sigurnosnih zahtjeva, prenosivosti i podudarnosti mobilne aplikacije, bitno je spomenuti i ostale nefunkcionalne zahtjeve mobilne aplikacije za vođenje mortalitetne statistike. Mogućnost korištenja aplikacije iznimno je olakšana signalizacijom ukomponiranom u aplikaciji. Iznad svakog unosa, objašnjeno je što se traži od korisnika, a tipke su naznačene drugačijom bojom kako bi se što bolje istaknule. Također je jednostavan i tok navođenja korisnika kroz aplikaciju jer nema kompleksnih prozorčića niti padajućih izbornika koji bi mogli zbuniti korisnika. To čini aplikaciju pristupačnu različitim profilima osobama.

3.5. Dizajn korisničkog sučelja i korisničko iskustvo

Korisničko sučelje dio je interaktivnog računalnog sustava koji komunicira s korisnikom. Dizajn korisničkog sučelja sadrži bilo koji dio sustava koji je vidljiv korisniku. Ono postaje sve veći i veći dio programskog računalnog sustava te se čak i neki problemi sve više rješavaju korisničkim sučeljem nego sklopovljem [10]. Dizajn korisničkog sučelja stvorio se pomoću programskog alata *FluidUI* o kojem će biti više objašnjeno u potpoglavlju 4.1.1.

3.5.1. Elementi korisničkog sučelja

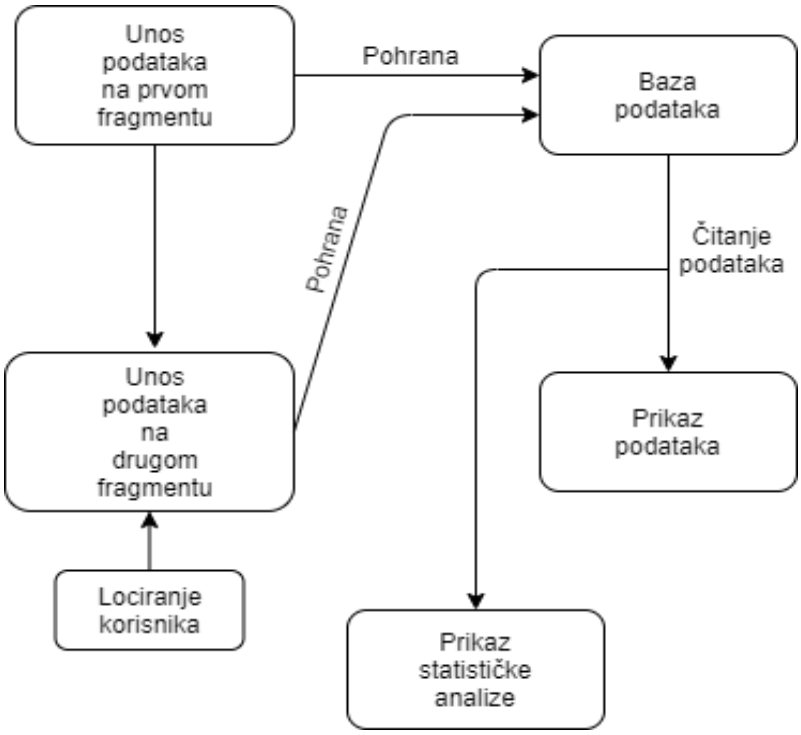
Korisničko sučelje mobilne aplikacije za vođenje mortalitetne statistike sastoji se od mnoštva dijelova koji zajedničkom kompozicijom obogaćuju korisničko iskustvo. Aplikacija započinje početnim zaslonom koji odvaja formalni dio aplikacije. Zatim slijedi dijaloški okvir uz detalje dobrodošlice korisniku i kreatorskih zasluga. Nakon toga korisnik se upoznaje s formalnim dijelom aplikacije – unosom i pohranom na prvom fragmentu, a slijedno i na drugom fragmentu aplikacije. Naposljetku, pritiskom na tipku *Rezultati* korisnik može vidjeti ispis svojih unosa unutar malih spremnika i grafičku statističku obradu istih unosa.

3.5.2. Očekivano korisničko iskustvo i pokazatelji korisničkog iskustva

Primarni pokazatelji korisničkog iskustva ogledaju se u uspješnosti obuhvaćanja zahtjeva korisnika mobilne aplikacije. Shodno tome dolazi zadovoljstvo i kvalitetna povratna informacija korisnika. Takvo se korisničko iskustvo i očekuje jer su elementi korisničkog sučelja mobilne aplikacije prilagođeni korisniku, njegovim zahtjevima i što boljem korisničkom dojmu.

3.6. Struktura mobilne aplikacije prema idejnom rješenju

Slika 3.4 pokazuje tijek aktivnosti u mobilnoj Android aplikaciji za vođenje mortalitetne statistike prema idejnom rješenju opisanom u ovom poglavlju, a koji je programski razrađen u poglavlju 4.



Slika 3.4. : Struktura aplikacije za vođenje mortalitetne statistike prema idejnom rješenju

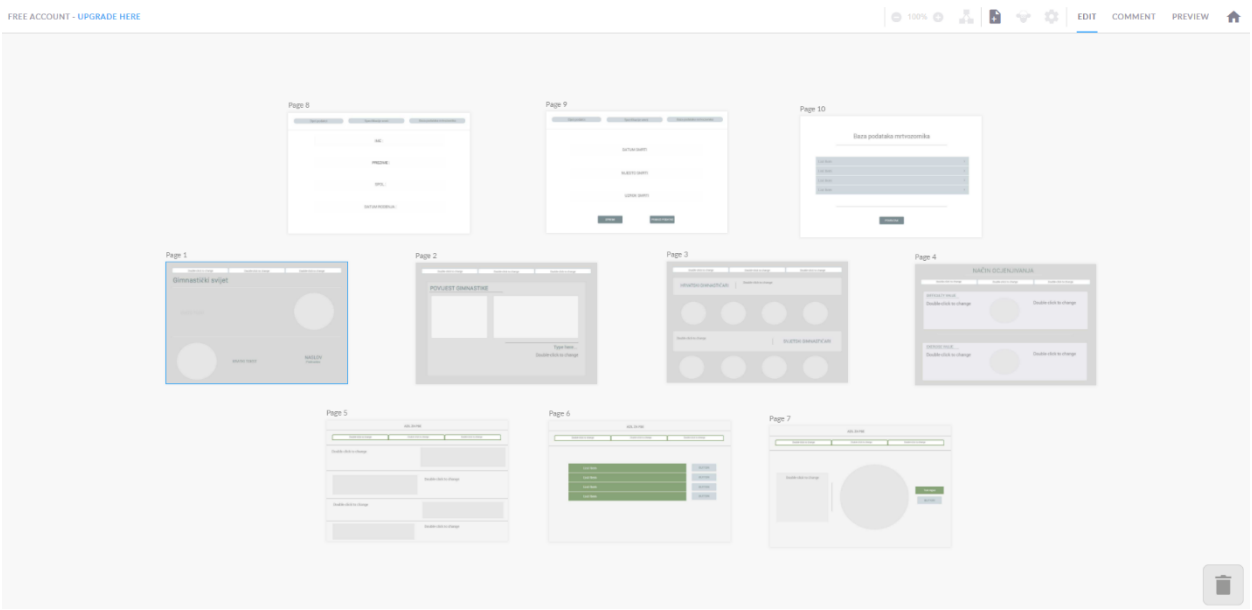
4. PROGRAMSKA IZVEDBA MOBILNE APLIKACIJE

4.1. Programske tehnologije, okoline i jezik korištene za izradu aplikacije

Mobilna Android aplikacija za vođenje mortalitetne statistike ostvarena je u programskom sučelju *Android Studio*. Idejni prikaz mobilne aplikacije stvorio se internetskom aplikacijom *FluidUI* dok se korisničko sučelje ostvarilo XML označnim jezikom. Korisničko sučelje i programsko sučelje povezuje se *Java* programskim jezikom. *Java* je objektno orijentirani jezik koji udružuje različite funkcionalnosti unutar aplikacije poput povezivanja na bazu podataka *Firebase*, API ključeve koji omogućuju lociranje i pregled *Google Mapsa*. U sljedećim će se potpoglavljima detaljno razraditi navedene komponente programske izvedbe mobilne aplikacije.

4.1.1. FluidUI

FluidUI je, prema [11], internetska usluga koja omogućuje korisnicima stvaranje idejnoga pred pregleda njihovih aplikacija. Pred pregled je moguće uređivati, dijeliti i izvesti u obliku slike. Uslugu provodi *Fluid Software d.o.o.* u 190 različitih država u svrhu razvoja poboljšanja i modificiranja vizije njihovih korisnika.



Slika 4.1. : Sučelje razvoja projekta u *FluidUI*

4.1.2. Operacijski sustav Android

Operacijski sustav Android mobilni je sustav kojeg je razvila *Android, Inc*. Međutim, *Google* je stekao prava 2005. godine i sve do danas razvija sustav sukladno sa svojim uslugama. Operacijski sustav primarno je razvijen za uređaje s mogućnošću dodirne interakcije sa zaslonom. Izvorni kod Android operacijskog sustava izdan je u obliku otvorenog izvornog koda kako bi se omogućio vanjski utjecaj korisnika na razvoj operacijskog sustava [12].

Razvoj operacijskog sustava Android podržava programski jezik *Java*. Prva se SDK (*Software Development Kit* – Paket za razvoj programa) verzija – Android 1.0 u javnu upotrebi izdala 2008. godine. Najnovija je verzija Android 11.0 i koristi se od 2020. godine. Prema [13], SDK nudi skup alata, biblioteka, važnih dokumentacija, primjera kodova, procesa i instrukcija koje omogućuju programeru napraviti programsko rješenje aplikacije na određenoj platformi. Zajedno s API, koji su skup građevnih blokova za stvaranje pojedinih aktivnosti, omogućuju kvalitetno sučelje razvoja aplikacije operacijskim sustavom Android. Arhitektura Android operacijskog sustava sastoji se od pet različitih slojeva, a to su: aplikacije, aplikacijski okvir, biblioteke, *Android Runtime* i *Linux* jezgra.

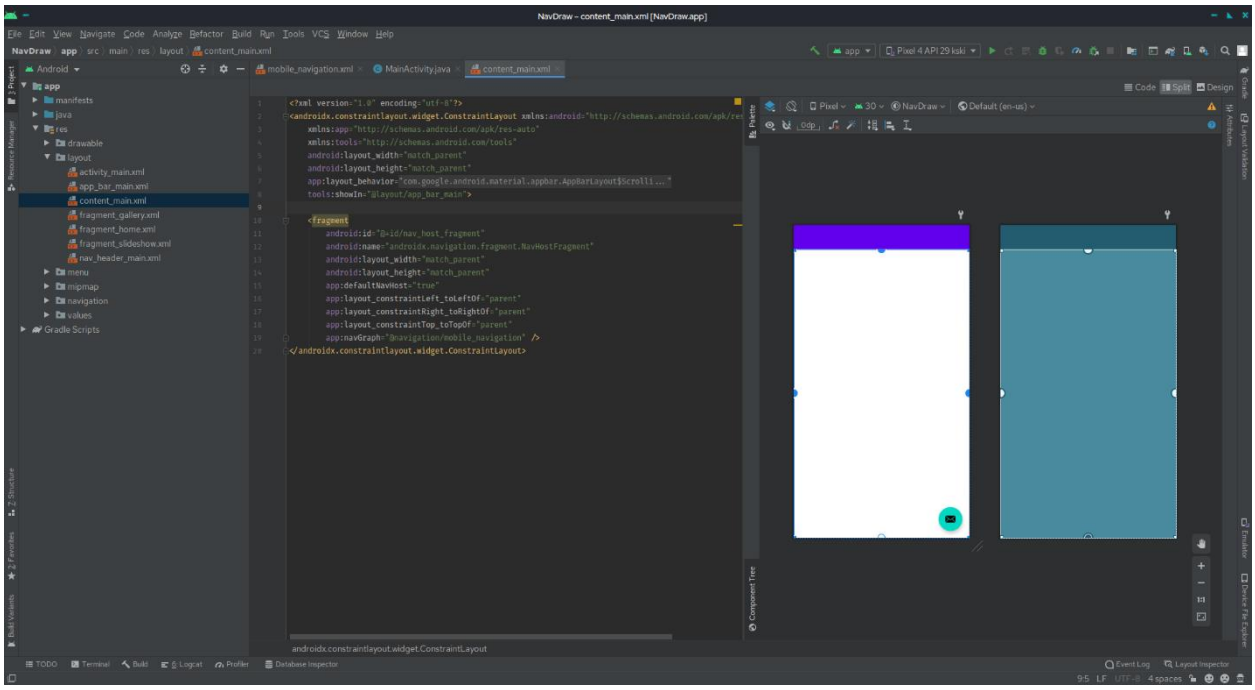
Odlike su Android operacijskog sustava: brzina i protočan prijenos podataka, korištenje usluga *Google Maps*, jezična podržanost sa skoro 100 jezika, virtualni asistent (*Google assistant*), otvorena sposobnost razvoja aplikacija *Android Studiom*, velika raširenost na različitim mobilnim uređajima [14].

4.1.3. Android Studio

Sukladno [15], *Android Studio* službena je integrirana razvojna okolina (IDE – *Integrated Development Environment*) za razvoj Android aplikacija, a temelji se na IntelliJ IDEA razvojnom sučelju. *Android Studio* nudi mnoštvo inovacija koje poboljšavaju produktivnost izgradnje Android aplikacija kao što su: brzi *Emulator* (virtualni uređaj) ispunjen različitim izborom značajki za korištenje, jedinstvena razvojna okolina, razni dodaci (*GitHub*, *Firebase*), alati za ispitivanje i pronalaženje grešaka u kodu, ugrađena platforma *Google* oblak. *Android Studio* podržan je na operacijskom sustavu *Windows, macOS* i *Linux*. Na slici 4.2 može se vidjeti sučelje *Android Studia*.

Prema zadanim postavkama, svaki projekt u *Android Studio* sadrži jedan ili više modula s izvornim kodom resursa i podataka. Svi moduli su vidljivi pod *Gradle Scripts* i oni sadrže sljedeće mape:

- ***Manifest*** – sadrži datoteku „AndroidManifest.xml“ gdje se nalaze svi podatci o aplikaciji kao što su identitet, ikona, aktivnosti, konfiguracije…
- ***Java*** – uz test kod „JUnit“, sadrži i podatke *Java* izvornog koda koje predstavljaju java klase aplikacije
- ***Res*** – sadrži sve resurse koji nisu isprogramirani kao što su XML izgledi (*layouts*), UI tekstovi (*strings*) i slike.



Slika 4.2. : Sučelje *Android Studio*

4.1.4. XML

Extensible Markup Language, skraćeno XML, opisuje klase podataka zvane XML dokumenti i djelomično opisuje ponašanje računalnog programa koji upravlja njima. XML dokumenti napravljeni su od pohranjenih jedinica zvanih entiteti koji sadrže raščlanjene i neraščlanjene podatke. Raščlanjeni podatci sastoje se od znakova. Neki od njih formiraju označne podatke, dok neki formiraju oznaku. Oznaka šifrira opis izgleda pohranjenog dokumenta i njegovu logičku strukturu. Ovaj označni jezik pruža mehanizam koji nameće ograničenja na izgled pohrane i logičku strukturu. Program koji čita XML dokument naziva se *XML processor* i pruža čitanje njegovog sadržaja i strukture [16].

4.1.5. Java

Java je programski jezik u općoj namjeni, podudaran, klasno utemeljen objektno orijentirani jezik. Dizajniran je da bude jednostavan kako bi mnogi programeri mogli steći znanje i tečnost rada u njemu. Relativno je visoko-razinski jezik koji sadrži automatsko upravljanje memorijom [17].

Služi kao računalna platforma za razvoj aplikacije. Sadrži skup računalnih programa i specifikacija koje prevode kod, izvršavaju ga, otklanjaju pogreške i sintetiziraju ga u cjelinu. Pretežito se koristi na računalima, igraćim konzolama, znanstvenim super-računalima, mobitelima. Platformu je razvio James Gosling radeći u američkoj tehnološkoj tvrtki Sun Microsystems, a kasnije ju je otkupila *Oracle Corporation* [18].

Navodeći [19], *Java* je iznimno prenosiv jezik koji se može pokrenuti jednako na bilo kojem računalu bez obzira na značajke sklopovlja ili operacijskog sustava. Jedina je bitna stavka da mora posjedovati *Java* prevoditelj. Pored prenosivosti, važna je prednost *Jave* sigurnost od zloćudnih programa. Na taj se način omogućuje korisniku sigurno pokretanje *Java* aplikacije (*Java applet*) koja je preuzeta s Interneta jer *Java* sigurnosne značajke onemogućuju aplikacijama (*applets*) pristup računalnom tvrdom disku ili mrežnoj vezi. Tako su *Java* aplikacije ili *applets* centar razvoja *Java* programskim jezikom, a predstavljaju male *Java* programe koji su ugrađeni s HTML-om Internetske stranice. Služe za pokretanje animacija i zvukova na Internetu kako bi korisničko iskustvo bilo što kvalitetnije.

4.1.6. Firebase

Prema [20], *Firebase* je *Googleova* usluga za razvoj, rast i poboljšanje korisnikove mobilne aplikacije. Sadrži skup alata, značajki i usluga koje omogućuju korisniku olakšano iskustvo razvoja mobilne aplikacije. Pod ovim se podrazumijevaju usluge analitike i obrade, provjere sigurnosnih podataka, baze podataka i pohrane podataka. Usluga se održava na oblaku i skalira u skladu sa zahtjevima korisnika. Nužno je jedino implementirati *Google* SDK pakete kako bi se omogućila povezanost između poslužitelja (*Google*) i aplikacije (korisnik).

Firebase omogućuje pregršt proizvoda s različitim funkcionalnostima poput sigurnosnih mjera, vođenja statistike i pohrane podataka, a neki od njih su:

- *Realtime Database*
- *Analytics*
- *Performance Monitoring*
- *Cloud Storage*

4.1.7. Google Maps

Google Maps predstavlja orijentacijsko-navigacijsku aplikaciju za računalne i mobilne uređaje koju je razvio *Google*. Mape omogućuju korisniku slijedne navigacijske naredbe do odredišta uz 2D i 3D satelitski prikaz, te isto tako i informacije o javnom prometu. Također pružaju mogućnost takozvanog *Street View* izbora koje prikazuje stvarne slike ulica i prostora. *Google Maps* koristi GPS uređaja koji mora biti uključen kako bi usluga funkcionirala. Uz to, dohvaća *Googleovu* vlastitu stanicu i Wi-Fi lokacijske usluge [21].

4.2. Programski predložak mobilne arhitekture

Kako navodi [22], mobilna je arhitektura skup svih strukturnih elemenata, sučelja i njihovih ponašanja od kojih se sustav sastoji. Može se reći kako je ono kostur za izgradnju programa i cjelovitosti mobilne aplikacije. Ona određuje njezinu izvedbu u kvaliteti, performansama i trajnosti. Složenost izgradnje visokokvalitetne arhitekture ovisi o veličini aplikacije. Ispravno sastavljena arhitektura omogućuje uštedu budućeg vremena, resursa i troškova. Shodno ovome, adekvatna se mobilna arhitektura ne bi trebala oslanjati na vanjske biblioteke i resurse jer posjeduju vlastita ograničenja čime bi se ograničila i korisnikova aplikacija. Njihova je svrha poboljšanje i optimizacija, a ne stvaranje i ovisnost. Stoga se dijagram aplikacijske arhitekture ne otkriva drugim operacijskim sustavima ili bazama podataka kako bi mogao funkcionirati neovisno o njima.

Kako naglašava [23], postoje četiri pojma koje svaki programer želi ostvariti u svojoj arhitekturi jer su oni ogledni parametri za ocjenu kvalitete aplikacije, a to su:

- Elastičnost (engl. *scalability*)
- Pouzdanost (engl. *reliability*)
- Održivost (engl. *maintainability*)
- Mogućnost testiranja i ponovne uporabe koda (engl. *code reusability and testability*)

Neki od predložaka mobilnih arhitektura su:

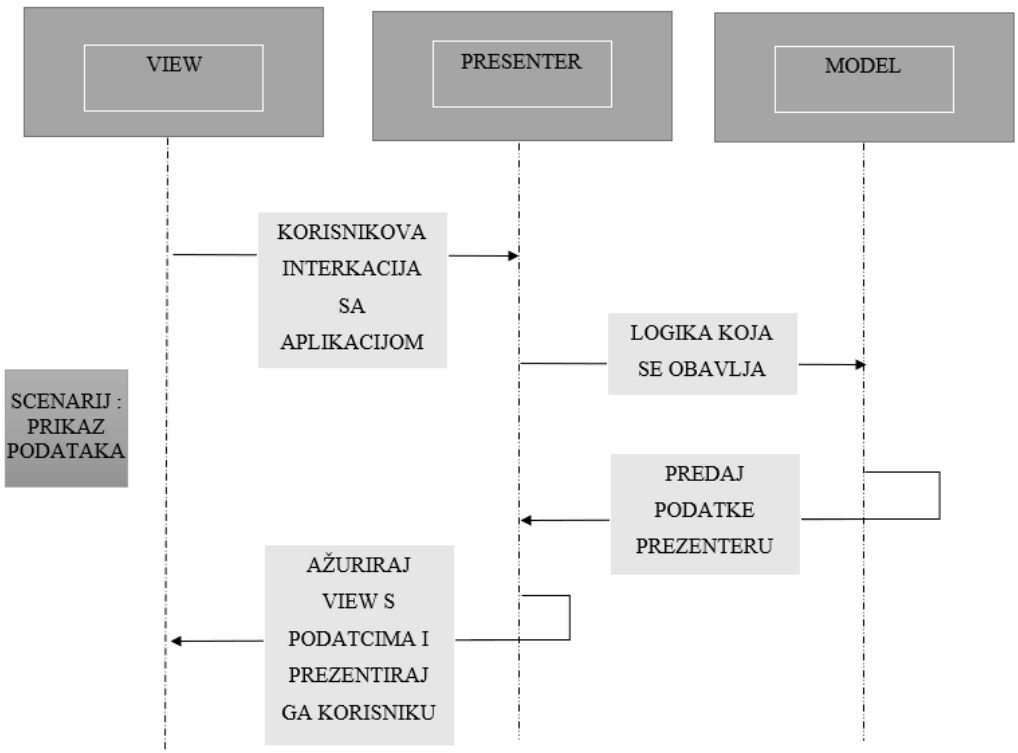
- MVC (*Model-View-Controller*)
- MVP (*Model-View-Presenter*)
- MVVM (*Model-View-ViewModel*)
- MVI (*Model-View-Intent*)

4.2.1. Predložak mobilne arhitekture MVP

Model-View-Presenter (MVP) je oblikovni obrazac za stvaranje predloška mobilne arhitekture kojeg koristi mobilna aplikacija za vođenje mortalitetne statistike. Prema [24], usmjeren je na pružanje čišće razlike između pozadinskih zadataka od aktivnosti (*Activity*), fragmenata (*Fragments*) i pogleda (*Views*) kako bi aplikacije izgledala jednostavnije i održivije. Predložak razdvaja ovu logiku u tri sloja:

1. Pogled (*View*)
2. Model (*Model*)
3. Prezentator (*Presenter*)

Model, prema objašnjenju u [25], posjeduje entitete i usluge koji sadrže podatke koji se spremaju u tablice baze podataka, a koji će se kasnije koristiti u druge svrhe poput njihovog dohvaćanja. Pogled ima odgovornost interakcije s korisnicima i ne sadrži nikakvu logiku, već navodi korisnika u navigaciji aplikacijom prema prezentatoru. Ono često ima reference na svojega prezentatora. Prezentator je posrednik između pogleda i modela. Sva poslovna logika, logika podataka, ispravnost unosa, prihvaćanje događaja iz pogleda, pretvaranje podataka iz pogleda u *Model*, i obrnuto, implementirani su u prezentatora. Slika 4.3 prikazuje uporabu MVP koja je nacrtana po uzoru na literaturu [26].



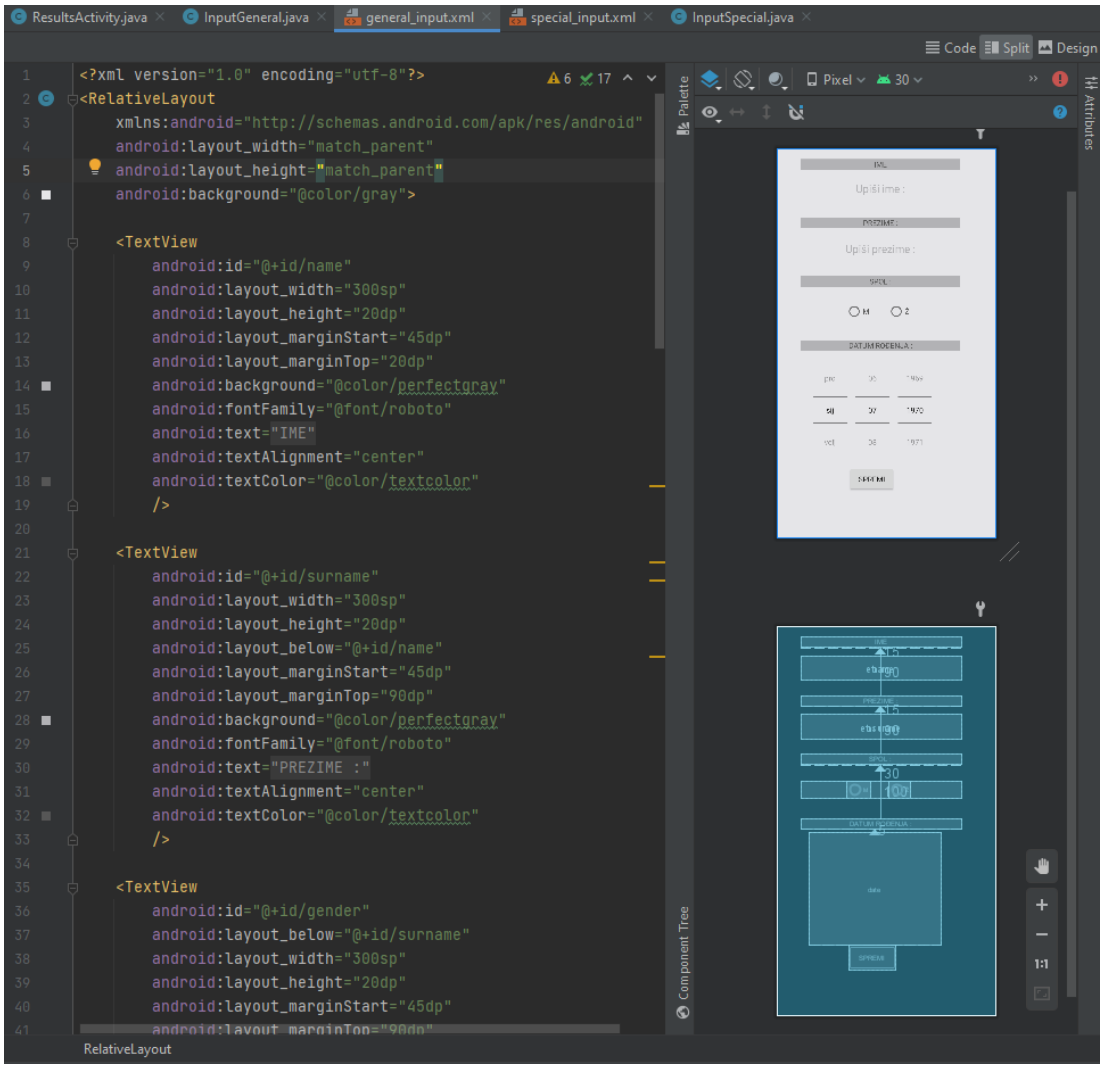
Slika 4.3. : Slijedni dijagram MVP obrasca

4.3. Programsko rješenje na strani korisnika

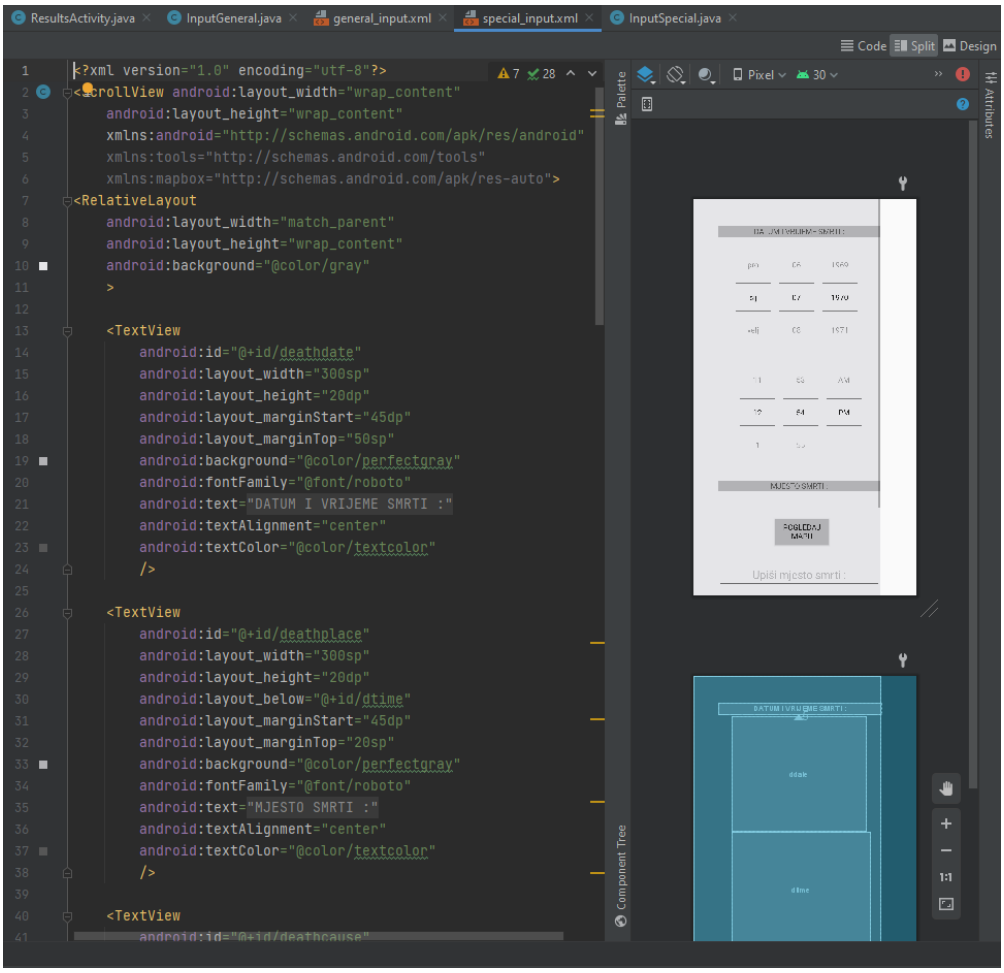
U ovom potpoglavlju bit će prikazani svi dijelovi aplikacije uz njima pripadajući programski kod.

4.3.1. Unos podataka

Unos podataka odvija se u dva različita fragmenta na način da korisnik unese tražene podatke u pogled *EditText*. Na taj se način ostvaruje interakcija između korisnika i aplikacije što podržava MVP. Prvi fragment sastoji se od četiri unosa: IME, PREZIME, SPOL i DATUM ROĐENJA dok se drugi sastoji od: DATUMA I VREMENA SMRTI, MJESTA SMRTI i UZROKA SMRTI. Prvo se cjelokupni skup tih resursa mora napraviti u XML-u kako bi se mogao nakon toga ubrizgati u *Java* klase. Slike 4.4 i 4.5 prikazuju XML mape: „general_input.xml“ i „special_input.xml“.



Slika 4.4. : „general_input.xml“

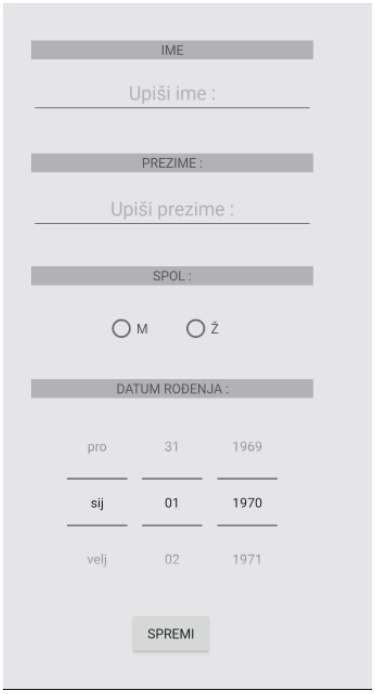


Slika 4.5. : „special_input.xml“

Prikaz svih tih pogleda mora se sada ubrizgati preko svojstvenih *Java* klasa (*InputGeneral* i *InputSpecial*) u metodi *onViewCreated*. Ovo se odvija na način da se pronalazi identifikacijska oznaka *ID* svakoga pogleda i pridružuje se stvorenom atributu klase. Atribut je jednakog tipa kao i pogled u resursima. Na taj način ostvaruje se prikaz svega onoga što je u XML datoteci na zaslon uređaja gdje će biti ostvarena aplikacija. Taj se cjelokupni proces pridruživanja *ID-a* XML datoteke atributu klase naziva proces ubrizgavanja što prikazuje slika 4.6. Slike 4.7 i 4.8 prikazuju kako izgledaju komponente unosa nakon procesa ubrizgavanja.

```
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    name = view.findViewById(R.id.name);
    surname = view.findViewById(R.id.surname);
    gender = view.findViewById(R.id.gender);
    birthdate = view.findViewById(R.id.birthdate);
    etName = view.findViewById(R.id.etname);
    etSurname = view.findViewById(R.id.etnsurame);
    radioGroup = view.findViewById(R.id.group);
    radioButtonMale = view.findViewById(R.id.male);
    radioButtonFemale = view.findViewById(R.id.female);
    datePicker = view.findViewById(R.id.date);
```

Slika 4.6. : Ubrizgavanje resursa u atribute klase



Slika 4.7. : Prikaz ubrizganog resursa „general_input.xml“

Slika 4.8. : Prikaz ubrizganog resursa „special_input.xml“

4.3.2. Pohrana podataka u bazu podataka

Nakon korisnikova unosa na predviđena područja u mobilnoj aplikaciji, potrebno ih je pohraniti. Pohrana se ostvaruje pogledima *Button* koji su se prethodno ubrizgali u *Java* klase. Kada korisnik stisne *Button Spremi*, poziva se metoda *setUpData* gdje se podatci pohranjuju mrežno na usluzi *Firebase* uz svojstveni primarni ključ (na slici 4.9 atribut *maxID*). Ono što je napisano na predviđena mjesta unutar pogleda *EditText* i izbornika za vrijeme, datum, lokaciju, spol i uzrok smrti pretvara se u tekstualni oblik kako bi se moglo pohraniti u bazu podataka. (Slika 4.9)

```
public void setUpData(View view){
    button = view.findViewById(R.id.saveStates);
    reference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if(snapshot.exists()){
                maxID = (snapshot.getChildrenCount());
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(etName.getText().length() == 0 || etSurname.getText().length() == 0){
                Toast.makeText(getContext(), text: "Fill all fields please", Toast.LENGTH_LONG).show();
            }else {
                deathList.setName(etName.getText().toString().trim());
                deathList.setSurname(etSurname.getText().toString().trim());
                deathList.setGender(getGender());
                deathList.setBirthdate(getDate(String.valueOf(datePicker.getDayOfMonth()),
                        String.valueOf(datePicker.getMonth() + 1),
                        String.valueOf(datePicker.getYear())));
                reference.child(String.valueOf(maxID + 1)).setValue(deathList).
                        addOnCompleteListener(new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        Toast.makeText(getContext(), text: "First part of data is saved", Toast.LENGTH_LONG).
                                show();
                    }
                });
            }
        }
    });
}
```

Slika 4.9. : Proces pohrane podataka u prvom fragmentu *„GeneralInput“*

Kako se unos odvija na dva različita fragmenta, podatci se moraju prvobitno pohraniti u *Model* koji se zove *DeathList* kojeg prikazuje slika 4.10. Nakon što se svi podatci pohrane, cjelokupni se objekt podiže na *Firebase* sa svim svojim atributima što je prikazano slikom 4.11. To pogoduje predlošku MVP jer prezentator (metoda u klasama) obavlja logiku nad modelom.

```java
package com.example.zavrsniradfranzamaklar.inputs;

import ...

public class DeathList{
    String name;
    String surname;
    String gender;
    String birthdate;
    String deathdate;
    String deathtime;
    String deathplace;
    String deathcause;

    public DeathList(){}

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getSurname() { return surname; }

    public void setSurname(String surname) { this.surname = surname; }

    public String getGender() { return gender; }

    public void setGender(String gender) { this.gender = gender; }

    public String getBirthdate() { return birthdate; }

    public void setBirthdate(String birthdate) { this.birthdate = birthdate; }


    public String getDeathdate() { return deathdate; }

    public void setDeathdate(String deathdate) { this.deathdate = deathdate; }

    public String getDeathtime() { return deathtime; }

    public void setDeathtime(String deathtime) { this.deathtime = deathtime; }

    public String getDeathplace() { return deathplace; }
```

Slika 4.10. : Model *„DeathList“*

Bitno je napomenuti kako se prije svega mora ostvariti povezanost s uslugom *Firebase* što će se kasnije objasniti u potpoglavlju 4.4.3.

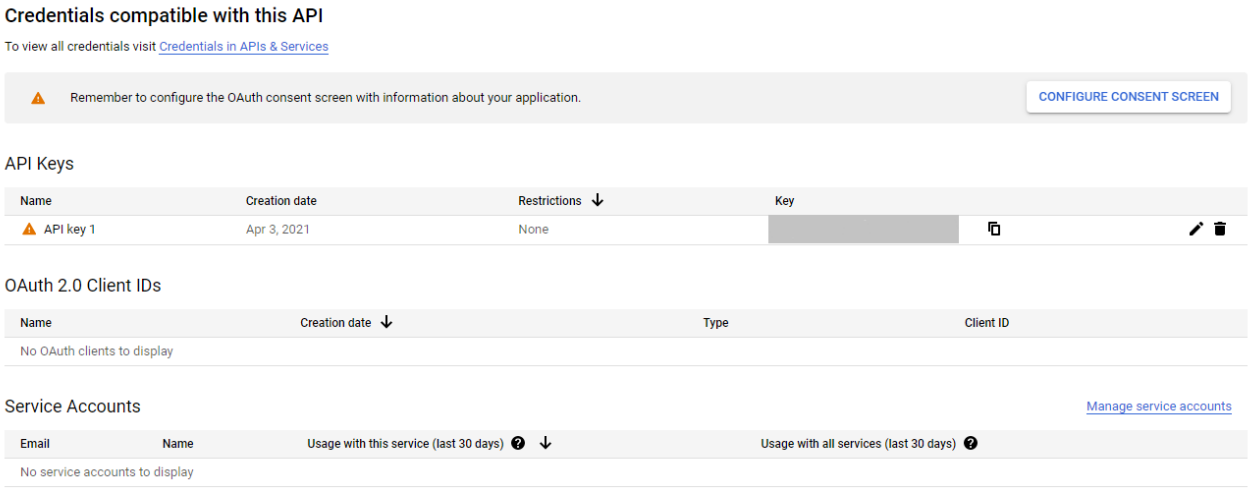
```java
void setUpData(View view){

    submit = view.findViewById(R.id.submit);
    submit.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            deathList.setDeathdate(getDate(String.valueOf(dDate.getDayOfMonth()),
                    String.valueOf(dDate.getMonth() + 1),
                    String.valueOf(dDate.getYear())));
            deathList.setDeathtime(getTime(timePicker.getHour(), timePicker.getMinute()));
            deathList.setDeathplace(etmaps.getText().toString().trim());
            deathList.setDeathcause(getCause());
            reference.addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    DataSnapshot lastsnapshot = snapshot.child(String.valueOf(maxID));
                    lastsnapshot.getRef().setValue(deathList).addOnCompleteListener(new OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull Task<Void> task) {
                            Toast.makeText(getContext(),  text: "Second part of data is saved", Toast.LENGTH_LONG).show();
                        }
                    });
                }

                @Override
                public void onCancelled(@NonNull DatabaseError error) {

                }
            });
        }
    });
}
```

Slika 4.11. : Proces pohrane podataka u drugom fragmentu „*SpecialInput*“

4.3.3. Ostvarenje lociranja

Ostvarenje lociranja korisnika aplikacije također je jedna od mogućnost interakcije s aplikacijom. Korisnik ima mogućnost vlastitog lociranja, orijentacije na usluzi *Google Maps*, pregled raznih drugih usluga koje ga okružuju. Usluge se ostvaraju na temelju *Google* API poziva koji štite *Google* proizvode od nedozvoljenog pristupa ili zloupotrijebe. Ti se pozivi ostvaruju API ključevima koji su, prema [27], jedinstveni alfanumerički znakovi koji povezuju korisnikov *Google* račun s korisnikovim projektom. Ključevi se mogu pristupiti i generirati besplatno, ukoliko korisnik ima *Google* račun, na *Google Cloud Platform* što i prikazuje slika 4.12.

Također, za ostvarenje usluge *Google Maps* trebaju se postaviti dozvole poput: dozvola za pristup internetu, pristup finom i grubom lociranju u datoteci „Manifest.xml“ unutar projekta *Android Studio*.



Slika 4.12. : Ostvareni API ključevi preko usluge *Google Cloud Platform*

Komponenta Google mape implementira se preko klase *SupportMapFragment* koja nasljeđuje klasu *Fragment*, odnosno prilikom izrade XML, oznaci <fragment> se pridružuje ime da podržava *SupportMapFragment*. Time se omogućuje automatsko upravljanje životnim ciklusom pogleda mape.

Metoda *getMapAsync(OnMapReadyCallback)* automatski pokreće sustav mape i njezin pogled tako da je njezina implementacija u *Java* klasi nužna. Na slici 4.13 se također inicijalizira atribut *fusedLocationProviderClient* kako bi se mogao koristiti za lociranje i dobivanje zadnje lokacije korisnika.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);

    fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);

    mapFragment = (SupportMapFragment)getSupportFragmentManager().findFragmentById(R.id.maps);
    mapFragment.getMapAsync( callback: this);
    backButton = findViewById(R.id.back);
    backButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { endIt(); }
    });
}
```

Slika 4.13. : Stvaranje mape i lokacije unutar metode *onCreate*

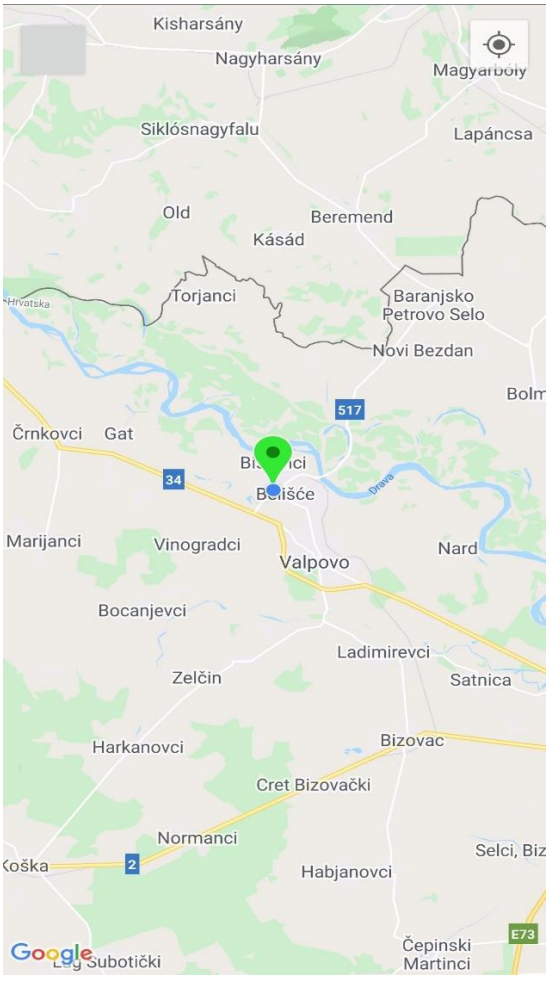
Slika 4.14 predstavlja metodu *onMapsReady(GoogleMap)* koja daje rezultat i povratnu informaciju o tome je li mapa spremna za korištenje. To je atribut kojeg prima metoda *getMapAsyncs*. Kada je mapa spremna za korištenje, na nju se mogu dodati još i „markeri“ – oznake koje označuju lokaciju na mapi. Kada se lociranje uzrokuje (korisnik zatraži vlastitu lokaciju), poziva se konstruktor markera u kojeg se smješta dužina *(Longitude)* i širina *(Latitude*) posljednje lokacije. Na taj se način dobivaju podatci o lokaciji korisnika aplikacije. Postupak stvaranja markera prikazan je na slici 4.15, dok se slikom 4.16 prikazuje zaslon ostvarenja lokacije korisnika.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mGoogleMap = googleMap;
    mGoogleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);

    locationRequest = new LocationRequest();
    locationRequest.setInterval(120000);
    locationRequest.setFastestInterval(120000);
    locationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);

    if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M){
        if(ContextCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED){
            fusedLocationProviderClient.requestLocationUpdates(locationRequest,mLocationCallback, Looper.myLooper());
            mGoogleMap.setMyLocationEnabled(true);
        }else {
            checkLocationPermission();
        }
    }else {
        fusedLocationProviderClient.requestLocationUpdates(locationRequest, mLocationCallback, Looper.myLooper());
        mGoogleMap.setMyLocationEnabled(true);
    }
}
```

Slika 4.14. : Metoda *onMapsReady*

```
LocationCallback mLocationCallback = onLocationResult(locationResult) → {
        List<Location> locations = locationResult.getLocations();
        if(locations.size() > 0){
            Location location = locations.get(locations.size() - 1);
            lastLocation = location;
            if(currentLocationMarker != null){
                currentLocationMarker.remove();
            }

            LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());
            MarkerOptions markerOptions = new MarkerOptions();
            markerOptions.position(latLng);
            markerOptions.title("Current Position");
            markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN));
            currentLocationMarker = mGoogleMap.addMarker(markerOptions);

            mGoogleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, zoom: 11));
        }
};
```

Slika 4.15. : Stvaranje markera korisnikove lokacije



Slika 4.16. : Prikaz ostvarenja lociranja korisnika

4.4. Programsko rješenje na strani poslužitelja

4.4.1. Prikaz rezultata pohranjenih u bazu podataka

Rezultati korisnikova unosa koji su pohranjeni u bazu podataka prikazuju se preko posebne nove aktivnosti u klasi *ResultsActivity*. Aktivnost omogućuje korisniku pregled vlastitih unosa uz sve njihove detalje. Slika 4.17 predstavlja ostvarenje prikaza rezultata. Logika se obavlja u pozadini kako bi se korisniku omogućila samo vizualna prezentacija podataka. To ide u korist MVP arhitekturi gdje se postojeći pogled mijenja u skladu s korisnikovim zahtjevima, a prezentator mu omogućuje prikaz i obavlja pozadinsku logiku.

Slika 4.17. : Prikaz rezultata preko klase *ResultsActivity*

Dizajn rasporeda pregleda podataka programski se ostvario *RecycleView* komponentom. Komponenta stvara okvir prostora za popunjavanje podataka, a usluga koja drži i upravlja podatcima je *RecycleAdapter*. Kad god se promijeni prostor unutar adaptera, ono se uz pomoć samoinicijativne metode *notifyDataSetChanged()* ažurira i prilagođava prostor novom obliku. Kako se povlače podatci iz baze podataka, tako se *RecycleAdapter* ostvaruje i formira. Podatci se dohvaćaju na objektu kojeg stvara *Firebase* referenca s podrškom metode *getValue* koja vraća

model spremljen na bazu podataka u cjelovitom obliku. Taj se model tada sprema u polje liste podataka (*ArrayList*) koje popunjava *RecyclerAdapter*. (Slika 4.18)

```
public void setUpRecycler(){

    recyclerView = findViewById(R.id.recycleView);
    recyclerView.setHasFixedSize(true);
    recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
    deathLists = new ArrayList<>();
    recyclerAdapter = new RecyclerAdapter( context: this, deathLists);
    recyclerView.setAdapter(recyclerAdapter);
    reference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            for (DataSnapshot dataSnapshot : snapshot.getChildren()){
                DeathList deathList = dataSnapshot.getValue(DeathList.class);
                deathLists.add(deathList);
            }
            setUpBars(deathLists);
            recyclerAdapter.notifyDataSetChanged();
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
    returnButton = findViewById(R.id.btnBack);
    returnButton.setOnClickListener(v -> endIt());
}
```

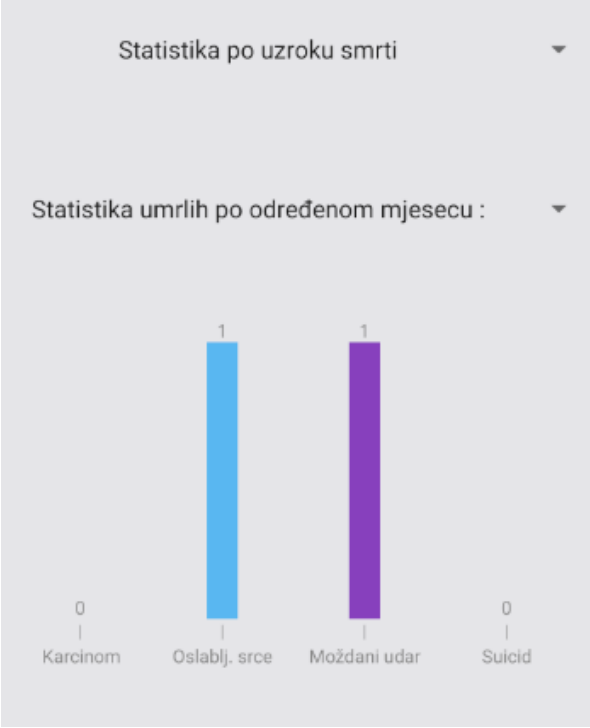
Slika 4.18. : Ostvarenje komponente *RecyclerView*

4.4.2. Prikaz statističke analize rezultata

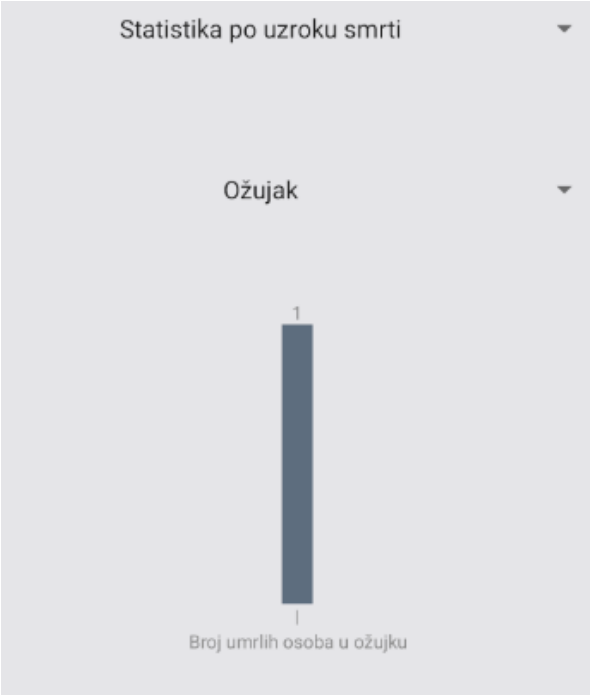
Poput rezultata korisnikovog unosa, statistička obrada tih istih podataka ostvaruje se unutar *ResultsActivity* aktivnosti, a slika 4.19 predstavlja programsko ostvarenje u *Javi*. Obrađuju se dvije statističke grupe: statistika po spolu i statistike po uzroku smrti, te jedna zasebna vremenska statistika. Grafički su prikaz stupci različitih boja koje signaliziraju varijacije između entiteta. Stupci se stvaraju na temelju porasta, odnosno po vrijednosti brojača. Brojači se povećavaju skladno po broju entiteta u modelima. Kada prođu svi modeli, formiraju se stupci metodom *addBar()* na objektu *BarChart* uz argumente imena stupca, vrijednosti rasta i boje. Naposljetku se prikaz stupaca realizira metodom *startAnimation*(). Kako se i spominje, dvije su statističke grupe specijalne statistike i jedna vremenske statistike. Prva, statistika po spolu, ima dva uvjeta. Prvi je postoji li atribut spol u modelu s vrijednošću „Žensko“. Ako postoji, povećava se brojač za tu vrijednost koji će kasnije poslužiti u formiranju stupca za ženski spol. Nasuprot njemu je stupac za muški spol i njegov uvjet koji diktira postoji li u modelu atribut spol s vrijednošću „Muško“. Provjeru pruža predefinirana metoda *equals*() koja provjerava podudarnost teksta predanog kao argument i vrijednosti atributa objekta nad kojim se metoda poziva. Analogno se logika ostvaruje i za drugu i treću statističku grupu – statistika po uzroku smrti i vremensku statistiku koja se ostvaruje na temelju različitih mjeseci. Prikazi statistika predstavljeni su slikama 4.20 i 4.21.

```
void setUpBars(ArrayList<DeathList> deathLists){
    mBarChart = findViewById(R.id.barchart);
    primarySpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
            if (deathLists.size() > 0) {
                if (parent.getItemAtPosition(position).equals("Statistika po spolu")) {
                    mBarChart.clearChart();
                    if (!flag) {
                        for (int i = 0; i < deathLists.size(); i++) {
                            if (deathLists.get(i).getGender().equals("Muško")) {
                                femaleValue++;
                            } else if (deathLists.get(i).getGender().equals("Žensko")) {
                                maleValue++;
                            }
                        }
                        mBarChart.addBar(new BarModel( _legendLabel: "Ženski spol", femaleValue,  _color: 0xFF873F56));
                        mBarChart.addBar(new BarModel( _legendLabel: "Muški spol", maleValue,  _color: 0xFF56B7F1));
                        mBarChart.startAnimation();
                        flag = true;
                    } else {
                        mBarChart.clearChart();
                        for (int i = 0; i < deathLists.size(); i++) {
                            if (deathLists.get(i).getGender().equals("Žensko")) {
                                femaleValue++;
                            } else if (deathLists.get(i).getGender().equals("Muško")) {
                                maleValue++;
                            }
                        }
                        mBarChart.addBar(new BarModel( _legendLabel: "Ženski spol", femaleValue,  _color: 0xFF873F56));
                        mBarChart.addBar(new BarModel( _legendLabel: "Muški spol", maleValue,  _color: 0xFF56B7F1));
                        mBarChart.startAnimation();
                    }
                    maleValue = 0;
                    femaleValue = 0;
                } else if (parent.getItemAtPosition(position).equals("Statistika po uzroku smrti")) {
                    if (!flag) {
                        for (int i = 0; i < deathLists.size(); i++) {
                            if (deathLists.get(i).getDeathcause().equals("Karcinom")) {
                                cancerValue++;
                            } else if (deathLists.get(i).getDeathcause().equals("Oslabljeno srce")) {
                                hstrokeValue++;
```

Slika 4.19. : Stvaranje stupaca u klasi *ResultsActivity*



Slika 4.20. : Prikaz statistike po uzroku smrti



Slika 4.21. : Prikaz vremenske statistike

4.4.3. Povezivanje na bazu podataka

Spajanje na *Firebase* bazu podataka odvija se pozadinski, izvan okvira korisničkog sučelja. U pozadini se obavljaju sljedeći koraci:

1. Ugrađivanje SDK paketa koji služe za povezivanje Google usluga s aplikacijom

2. Konfiguracije pravila baze podataka – pod ovim se podrazumijeva je li dopušteno zapisivanje i čitanje iz baze podataka

3. Programerski dio koji treba pozvati metodu dohvaćanja instance na *FirebaseDatabase* objektu

4. Stvoriti referencu *DatabaseReference* na *FirebaseDatabase* objektu uz metodu *getReference*() s argumentom naziva tablice baze podataka. Tada je veza između aplikacije i *Firebase* uspostavljena (Slika 4.22).

```
DatabaseReference reference =
        FirebaseDatabase.
        getInstance("https://zavrsniradfranzamaklar-18dc3-default-rtdb.firebaseio.com/").
        getReference( path: "Death list:");
```

Slika 4.22. : Spajanje na bazu podataka *Firebase*

5. KORIŠTENJE I ISPITIVANJE MOBILNE APLIKACIJE

5.1. Način korištenja mobilne aplikacije

Prvi i osnovni korak korištenja aplikacije je instalacija aplikacije na mobilni uređaj korisnika. Korisniku se pojavljuje početni zaslon učitavanja koji odvaja formalni dio aplikacije. Dolaskom na korisničko sučelje, korisniku se pokazuje dijaloški okvir dobrodošlice i kreatorskih zasluga. Nakon njega korisnik je u mogućnosti interakcije s aplikacijom.

Glavi dio aplikacije sastoji se od tri dijela: prvi fragment s osnovnim unosima, drugi fragment s unosima specifikacije smrti i treći zaslon s prikazima rezultata unosa i statističkom obradom. Prva dva fragmenta povezana su zajedničkom aktivnosti i životnim ciklusima fragmenata dok je treći zaslon odvojena aktivnost. Iznad fragmenata nalazi se navigacijska traka preko koje se može doći iz jednog fragmenta u drugi, a i jednostavnim se potezima preko zaslona mogu pokrenuti transformacije koje će zamijeniti fragmente. Kada se na prvom fragmentu unesu svi željeni podatci, potrebno je pohraniti podatke na tipku *Spremi*. Zatim se unos obavlja na drugom fragmentu i pohranjuju podatci na isti način. U tom su trenutku podatci pohranjeni u bazi podataka. Tada korisnik može pregledati svoje unose ako stisne tipku *Rezultati*. Otvara se nova aktivnosti zajedno s malim prozorčićima koji drže informacije o podatcima unutar baze podataka. Ispod podataka nalaze se padajući izbornici koji daju korisniku mogućnost odabira statističke analize. Nakon njegova odabira, pokreće se animacija grafova koji daju sliku statističke obrade.

Aplikacija se zaključuje korisnikovim povratkom na prethodni fragment gdje korisnik može ponovno unositi željene podatke.

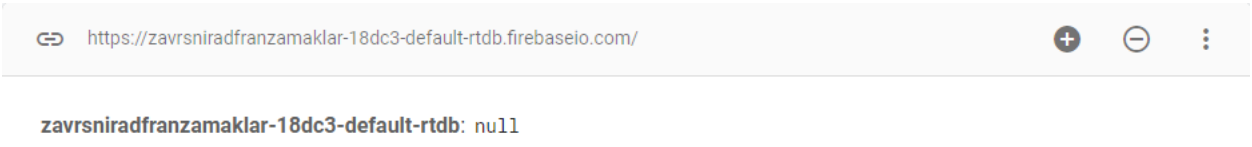
5.2. Ispitivanje mobilne aplikacije

5.2.1. Uvjeti ispitivanja mobilne aplikacije

Kako bi se prikazale što bolje funkcionalnosti aplikacije u kojima se ogleda i kvaliteta aplikacije, postavljaju se uvjeti ispitivanja mobilne aplikacije. Prvi je i glavni uvjet uspješnosti aplikacije – unos i pohrana podataka. Taj je uvjet esencijalan i diktira kronološku funkcionalnost ostalih stavki. Zatim slijedi uvjet prikaza podataka te kako se aplikacija ponaša kada u bazi nema podataka, a kada ima. Naposljetku stoji uvjet rada statističke analize i njezine obrade podataka. Ovaj se uvjet odlikuje u ocjeni kvalitete i ispravnosti obrade unesenih podataka, odnosno hoće li se podatci obraditi u skladu s očekivanjima. U sljedeća će se tri potpoglavlja razraditi ispitni slučajevi koji pokrivaju ove uvjete funkcioniranja mobilne aplikacije.

5.2.2. Ispitni slučaj 1

Prvi ispitni slučaj ispituje praznu bazu podataka i kako se aplikacija ponaša sukladno tome slučaju. Slika 5.1 predstavlja praznu bazu podataka, a slika 5.2 predstavlja funkcionalnost aktivnosti *ResultsActivity*.



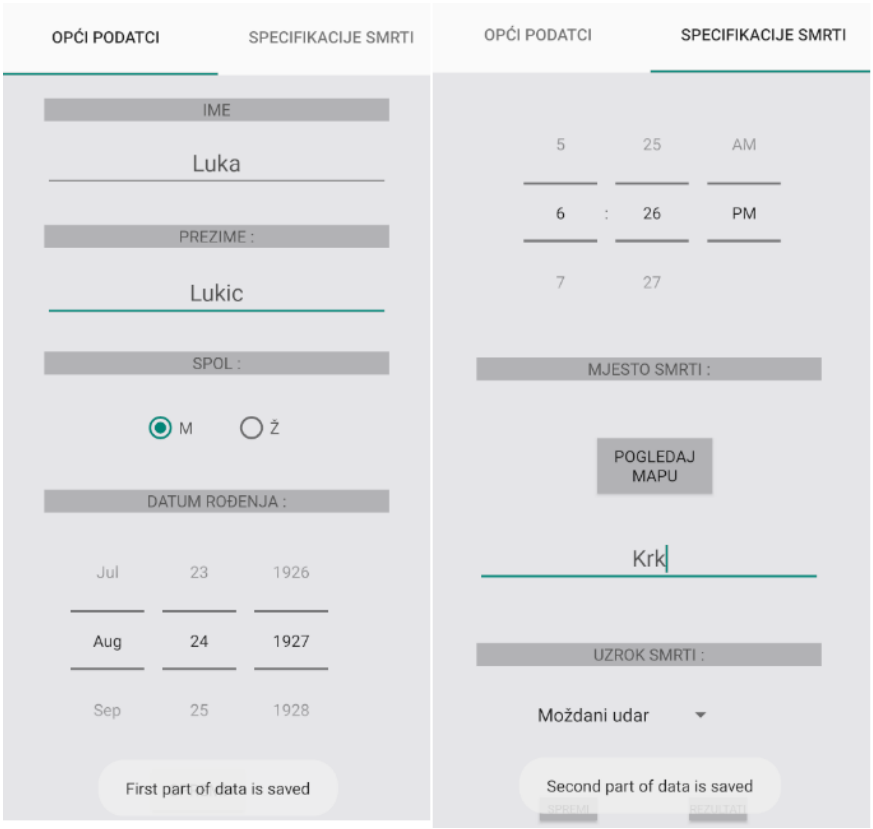
Slika 5.1. : Prazna baza podataka

Slika 5.2. : Ostvareni prikaz aktivnosti *ResultsActivity*

Ostvareni prikaz aktivnosti *ResultsActivity* je prazan spremnik i kada se pokuša odabrati statistička metoda, javlja se poruka da nema podataka u bazi podataka.

5.2.3. Ispitni slučaj 2

Drugim ispitnim slučajem provodi se analiza unosa jednog podatka. Prvo se unose opći podatci preminule osobe, a zatim specifikacije okolnosti smrti. Slike 5.3 i 5.4 prikazuju postupke provedbe ispitnog slučaja 2 kao i njegov konačni rezultat.

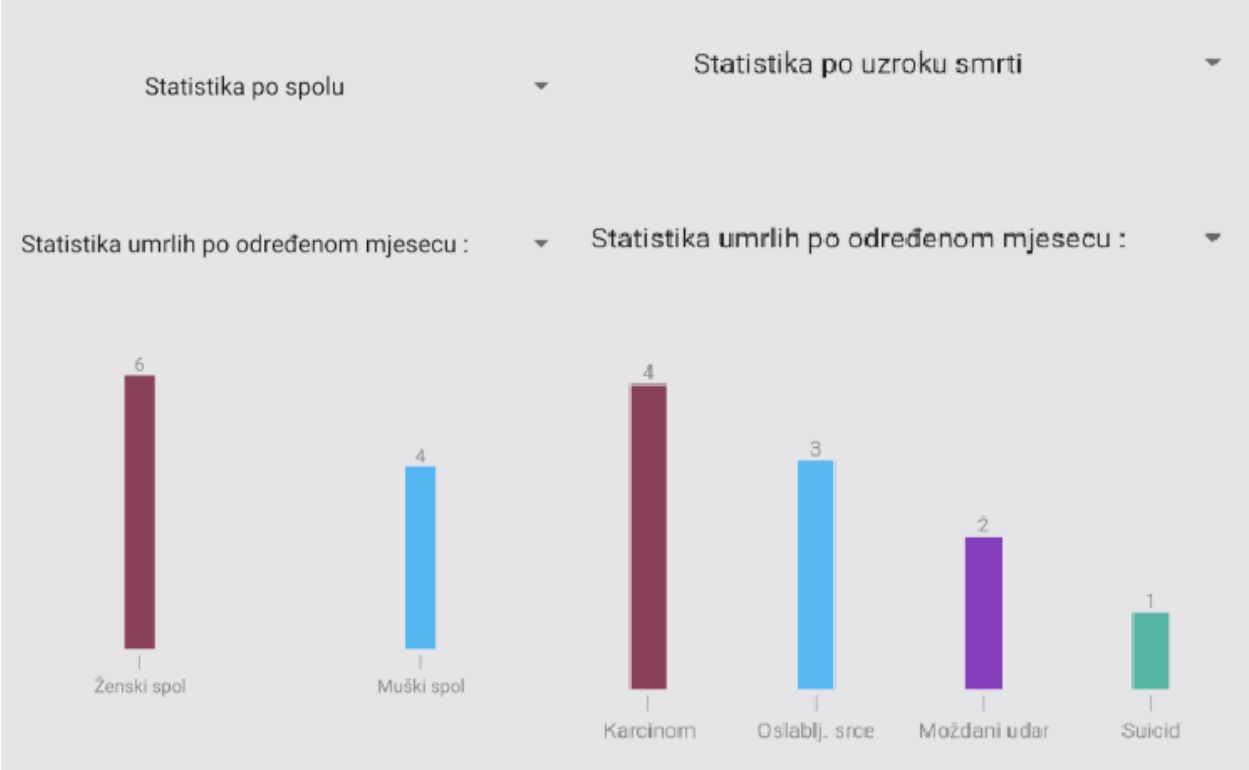


Slika 5.3. : Unos podataka za ispitni slučaj 2

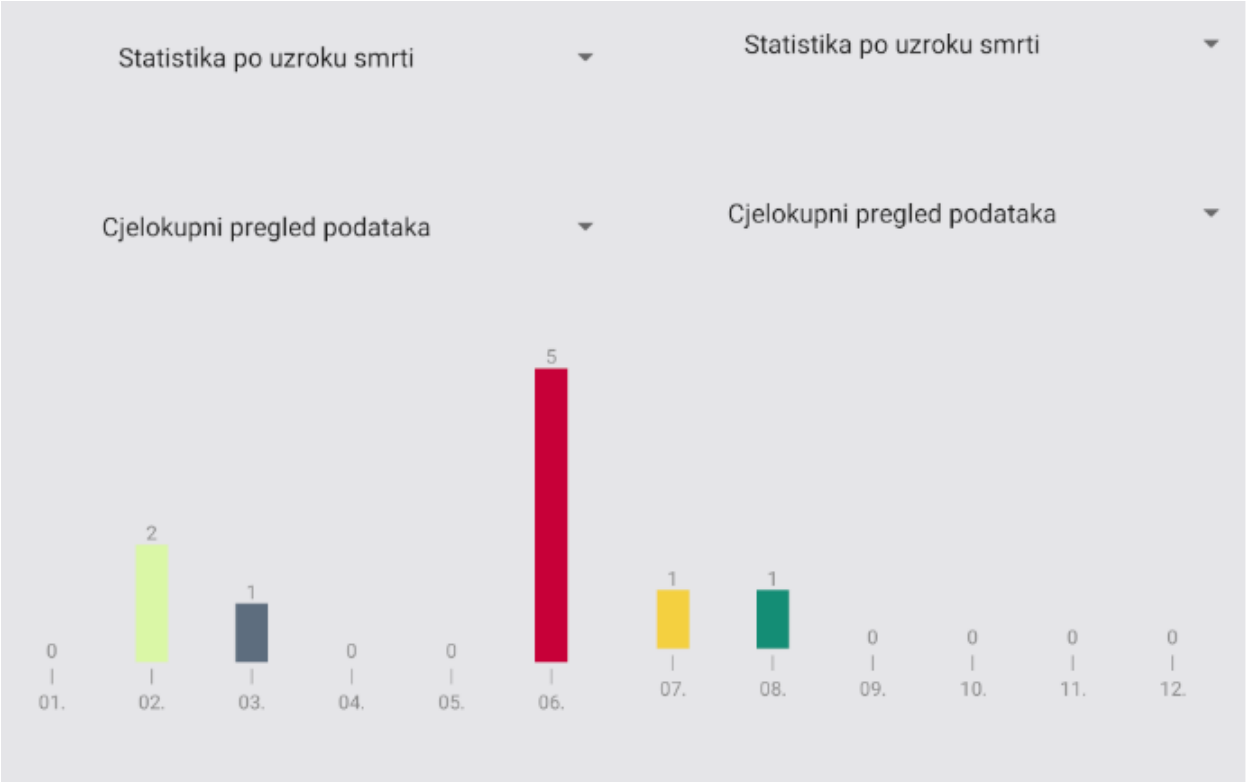
Slika 5.4. : Ispis rezultata u aktivnosti *ResultsActivity*

5.2.4. Ispitni slučaj 3

Trećim ispitnim slučajem ispituje se funkcionalnost statističke obrade pohranjenih podataka. Pohranjuje se uzorak od 10 podataka. Provodi se upotreba tipki *Statistička analiza prema spolu*, *Statistička analiza prema uzroku smrti* unutar komponente *Specijalna statistika* i *Cjelokupni pregled podataka* unutar komponente *Statistika umrlih po određenom mjesecu*. Navedeno je prikazano na slikama 5.5 i 5.6.



Slika 5.5. : Funkcionalnost statističke obrade po spolu i uzroku smrti



Slika 5.6. : Funkcionalnost statističke obrade umrlih po određenom mjesecu

5.3. Analiza i obrada podataka

5.3.1. Analiza uspješnosti testnih slučajeva

Sva tri ispitna slučaja uspješno su provedena. Prvim ispitnim slučajem provjerava se sigurnost funkcioniranja aplikacije kada nema podataka u bazi. Time se osigurava postojanje integriteta aplikacije bez vanjskog čimbenika, odnosno podataka. Drugim ispitnim slučajem ispituje se valjanost glavne funkcije aplikacije dok se trećim ispituje ispravnost rada statističke analize.

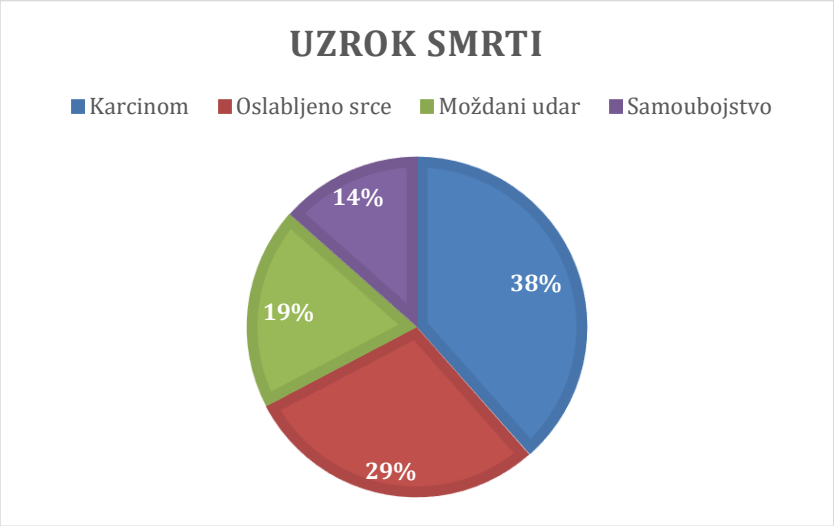
5.3.2. Statističke analize podataka

U sljedećim se potpoglavljima statistički analiziraju podatci koji su uneseni u bazu podataka. Na temelju odnosa skupnog udjela prikazat će se mortalitetno stanje prema spolu i uzroku smrt. U trećem je poglavlju ovog završnog rada najavljena statistička analiza po metodi uvećavanja koja se koristila u statističkog obradi i primjeni analize podataka. Za uzorak se uzeo skup od 10 unesenih podataka u bazi podataka.

5.3.2.1. Statistička analiza prema uzroku smrti

Slika 5.7 prikazuje statističku analizu prema uzroku smrti. Može se primijetiti da je uzrok smrti kod većine preminulih prirodan uzrok smrti (86%) te da su najviše preminulih osoba oboljele od

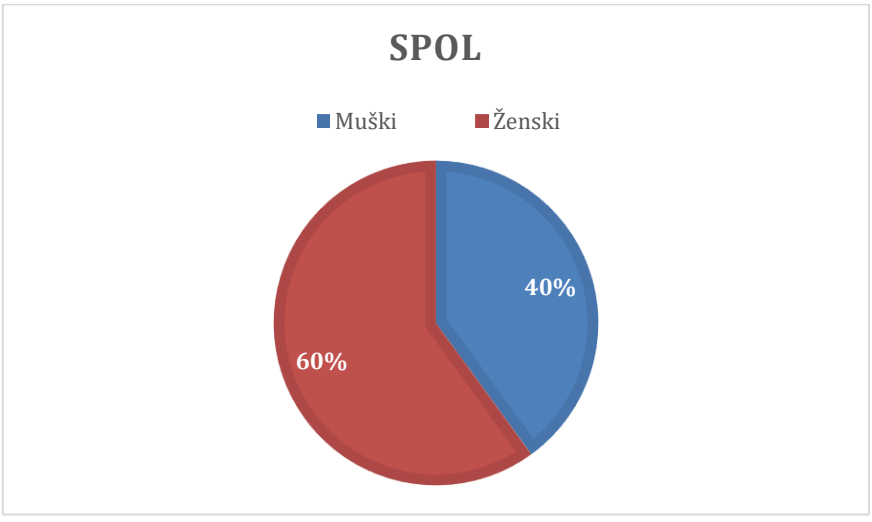
raka. Budući da su većina unesenih podataka osobe starije dobi, ovaj statistički prikaz nije devijacija i najveći postotak uzroka smrti je reprezentativno opravdan. Prema [3] (Izvješće o smrtnosti iz 2019.) od raka je umrlo 26% osoba, najviše 2019. godine te je time potkrijepljena ova analiza.



Slika 5.7. : Statistička obrada preminulih prema uzroku smrti

5.3.2.2. Statistička analiza smrtnosti prema spolu

Statistička obrada smrtnosti prema spolu prikazana je na slici 5.8. Iz priloženog se vidi viša stopa smrtnosti ženskog spola nego muškog. Također [3] navodi i prikazuje da su osobe ženskog spola preminule više, u omjeru s 51,05% osoba umrlih ženskog spola naspram muškaraca - 48,95%. Ovaj statistički prikaz odstupa od svjetskih statistika koje pokazuju da su češće preminule osobe muškarci. Tako američka analitika koja je obradila 2019. skup od 100000 osoba predstavlja odnos umrlih prema spolu kao 58,37% za muški spol, a 41,63% za ženski spol [28].



Slika 5.8. : Statistička obrada preminulih prema spolu

5.4. Skupna analiza rada mobilne aplikacije

Sinteza rada mobilne aplikacije za vođenje mortalitetne statistike ogleda se u njezinoj analizi pojedinačnih komponenti čije su se funkcionalnosti predstavile kroz ovaj rad. Svi očekivani i zadani funkcionalni zahtjevi su ostvareni, a aplikaciji, zajedno s nefunkcionalnim zahtjevima, nude kompetentnost, svrhu i kvalitetno korisničko iskustvo.

Android Profilerom prikazale su se performanse aplikacije te da je vremenski odziv od tri sekunde ogled dobre izvedbe mobilne aplikacije. Osim toga, sigurnosnim zahtjevima kao što su dopuštenja i ograničenja, omogućena su pravilna funkcionalna svojstva aplikacije. Aplikacija je testirana s dva uređaja. Jedan je virtualni *Google* uređaj (*Emulator* – Pixel 3a), a drugi HUAWEI P30 Pro. Uređaji su dali podjednake rezultate te su se sve aplikacijske funkcionalnosti identično ostvarile na oba uređaja. Signalizacijom se ostvarilo lakše navođenje korisnika u korištenju aplikacije čime se omogućila veća pristupačnost istoj. Značaj aplikacije ogleda se u činjenici da je jednako mogu koristiti stariji i mlađi korisnici. Naglasak je postavljen i na estetskom izgledu korisničkog sučelja čiji se idejni prikaz ostvario koristeći programski alat *FluidUI*. Predložena struktura mobilne aplikacije ostvarena je prema idejnom rješenju te su svi zadaci, metode, klase, resursi i atributi funkcionalni i postojani.

Mobilna Android aplikacija za vođenje mortalitetne statistike kompaktna je i svrsishodna aplikacija. Objedinjuje skup traženih parametara, funkcionalne i nefunkcionalne zahtjeve i modele, estetski dizajn i korisničko sučelje te izvedbu cjelovitog zadatka u kohezivnu kompoziciju. Aplikacija je također fleksibilna, podložna ažuriranjima i nadogradnjama koji će se prilagođavati sukladno korisnikovim novim zahtjevima i potrebama.

6. ZAKLJUČAK

Mortalitet je izravni pokazatelj općeg demografskog stanja i njegovog utjecaja na zdravstvo. Odraz kvalitete i pokazatelj načina života svake države ogleda se između ostalog i u činjenici koliko se država bavi stopom mortaliteta. Stopa mortaliteta izuzetno je važan pokazatelj općeg stanja u društvu jer visoki postotak smrtnosti u nekoj državi govori u prilog lošoj zdravstvenoj skrbi, velikoj socijalnoj neujednačenosti odnosno niskim životnim standardom. Prema stopi mortaliteta mogu se poduzimati i odgovarajuće mjere u cilju zaštite zdravlja ljudi, osiguravanju boljih uvjeta života, proaktivnih mjera u smislu demografske učinkovitosti, a sve kako bi se uravnotežio odnos između nataliteta i mortaliteta. Posljednja tri desetljeća Republika Hrvatska ima veliki pad nataliteta, primarno zbog prirodnog pada uzrokovanog starošću, a 2020. godine stopu smrtnosti dodatno je pogoršala pandemija COVID - 19.

Ovim završnim radom ostvarena je mobilna Android aplikacija za vođenje mortalitetne statistike. Razvijena je s ciljem olakšavanja unosa, održavanja i praćenja podataka o mortalitetu, te uvođenja digitalne dokumentacije kao zamjena za dosadašnje Zakonom i Pravilnikom propisane očevidnike. Aplikacija omogućuje korisniku da na jednostavan način unosi i pohranjuje podatke o preminulim osobama. Korisnik iste može se služiti upisanim podatcima na praktičan način koristeći pri tome raspoložive podatke za prikaz rezultata unosa i statističku obradu. Time se digitalizirala aktivnost korisnika i pojednostavio način obavljanja djelatnosti mrtvozorništva. Pored navedenog osnovnog razloga za razvoj ove aplikacije, drugi razlog je i pandemija COVID - 19 koja je dodatno povećala stopu smrtnosti na globalnoj razini i otvorila potrebu za unaprjeđenje mortalitete statistike.

Usvojena znanja prikazana u ovom završnom radu primijenjena su za ostvarenje cilja izrade mobilne Android aplikacije. Ona je izrađena primjenom odgovarajućih programskih alata, u razvojnoj okolini *Android Studio* uz pomoć programskog jezika *Java* i označnog jezika XML. Baza podataka omogućena je programskim alatom *Firebase* te se funkcionalnost aplikacije proširila uslugom *Google Maps*. Dovršena aplikacija u potpunosti odgovara ideji planiranog zadatka. Ispitana je korištenjem u tri praktična slučaja. Ostvareni rezultati ispitivanja pokazuju njezinu ispravnu i odgovarajuću funkcionalnost u radu. Ova aplikacija je početni model koji će se unaprjeđivati i usavršavati daljnjim zahtjevima korisnika.

LITERATURA

[1] A. Aguirre, *The Use of Mortality Statistics in Health Planning,* dostupno na: https://www.stat.fi/isi99/proceedings/arkisto/varasto/agui0857.pdf [Pristupljeno 2.8.2021.]

[2] M. Erceg, *Odjel za mortalitetnu statistiku, O odjelu*, 10. veljače 2021., dostupno na: https://www.hzjz.hr/sluzba-epidemiologija-prevencija-nezaraznih-bolesti/odjel-za-mortalitetnu-statistiku-2/ [Pristupljeno 2.8.2021.]

[3] M. Erceg., A. Miler Knežević*, Izvješće o smrtnosti prema listi odabranih uzroka smrti u 2019.*, Uvod, Stranica 3, Rujan 2020. dostupno na: https://www.hzjz.hr/wp-content/uploads/2021/01/Bilten__Umrli-_2019-2.pdf [Pristupljeno 2.8.2021.]

[4] 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, Prosinac 2011. [Prisutpljeno 7.8.2021.]

[5] Cause of Death Quick Reference Guide – Apps on Google Play, dostupno na: https://play.google.com/store/apps/details?id=gov.cdc.iiu.anubis&hl=en_US&gl=US [Pristupljeno : 6.8.2021.]

[6] CDC Launches Health App that Displays Public Morbidity and Mortality, iMedicalApps, dostupno na: https://www.imedicalapps.com/2015/11/cdc-health-app-morbidity-mortality/ [Pristupljeno : 7.8.2021.]

[7] A. Faisandier, *System Opportunities and Requirements, vol. 2*, Sinergy'Com, Francuska, 2015 [Pristupljeno 22.8.2021.]

[8] M. A. Mabrok, M. Efatmaneshnik, , M. J. Ryan, *Intergrating Nonfunctional Requirements into Anxiomatic Design Methodology*, IEEE System Journal, 1-11., Kolovoz 2015. [Pristupljeno 22.8.2021.]

[9] D. Rowley, *The Business of Application Portability*, StandardView 4, 80-87, Lipanj 1996 [Pristupljeno 6.9.2021.]

[10] R. J. K. Jacob, *User Interface*, Encyclopedia of Computer Science, 1821-1826, Siječanj 2003. [Pristupljeno 22.8.2021.]

[11] FluidUI Terms and Conditions, dostupno na: https://www.fluidui.com/terms [Pristupljeno 4.7.2021.]

[12] Android Operating System dostupno na: What You Need to Know: https://www.investopedia.com/terms/a/android-operating-system.asp [Pristupljeno 4.7.2021.]

[13] What is the Difference Between an API and an SDK?, dostupno na: https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/ [Pristupljeno 4.7.2021.]

[14] Android Operating System : Introduction, Features & Its Applications, dostupno na: https://www.elprocus.com/what-is-android-introduction-features-applications/ [Pristupljeno 4.7.2021.]

[15] Meet Android Studio, dostupno na: https://developer.android.com/studio/intro [Pristupljeno 4.7.2021.]

[16] T. Bray, J. Paoli, C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0. W3C Recommendation*, Veljača 1998., dostupno na: http://www.renderx.com/~renderx/Demos/fo2html/xml.pdf [Pristupljeno 5.9.2021.]

[17] J. Gosling, B. Joy, G. Steele, G. Brache, A. Buckley, *The Java Language Specification, Java SE 8 Edition*, Ožuljak 2015., dostupno na : https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf [Pristupljeno 5.9.2021.]

[18] What Is Java? Definition, Meaning & Features of Java Platforms, Guru99, dostupno na: https://www.guru99.com/java-platform.html [Pristupljeno 5.7.2021.]

[19] H. Austerlitz, *Data Acquistion Techniques Using PCs (Second Edition)*, 13. poglavlje, 13.1.7 Java, 2003. [Preuzeto sa stranice Science Direct, dostupno na: https://www.sciencedirect.com/topics/computer-science/java-programming-language, pristupljeno : 5.7.2021.]

[20] D. Stevenson, *What is Firebase? The Complete Story, Abridged*., 2018 [Preuzeto sa stranice Medium, dostupno na: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0, pristupljeno 6.7.2021.]

[21] Definition of Google Maps, dostupno na: https://www.pcmag.com/encyclopedia/term/google-maps [Pristupljeno 6.7.2021.]

[22] Your Basic Guide to Mobile App Architecture, Intellectsoft Blog, dostupno na: https://www.intellectsoft.net/blog/mobile-app-architecture/ [Pristupljeno 7.7.2021.]

[23] MVC vs MVP vs MVVM vs MVI. Choosing and Architecture for Android App, Mobindustry Software Engineering, dostupno na: https://www.mobindustry.net/mvc-vs-mvp-vs-mvvm-vs-mvi-choosing-an-architecture-for-android-app/ [Pristupljeno 7.7.2021.]

[24] Y.Zhang, Y. Luo, *An Architecture and Implement Model for Model-View-Presenter pattern, 2010 3rd International Conference on Computer Science and Information Technology*, 2010, Kina, IEEE

[25] Android Handbook | Project architecture / MVP (deprecated), dostupno na: https://infinum.com/handbook/books/android/project-architecture/mvp-deprecated [Pristupljeno 7.7.2021.]

[26] MVP Architectural Pattern, dostupno na: https://ducmanhphan.github.io/2019-08-05-MVP-architectural-pattern/ [Pristupljeno 7.7.2021.]

[27] Using API Keys, dostupno na: https://developers.google.com/maps/documentation/javascript/get-api-key [Pristupljeno 11.7.2021.]

[28] Number of Deaths per 100,000 Population by Gender, KFF, dostupno na: https://www.kff.org/other/state-indicator/death-rate-by-gender/?currentTimeframe=0&selectedDistributions=overall--male--female&sortModel=%7B%22colId%22:%22Location%22,%22sort%22:%22asc%22%7D [Pristupljeno 27.8.2021.]

SAŽETAK

Kako idejno osmisliti i u praktičnom dijelu provesti stvaranje mobilne Android aplikacije za vođenje mortalitetne statistike bio je cilj ovog završnog rada. Na temelju opisanog rješenja mobilne aplikacije, definicija i aktivnosti obavljanja djelatnosti mrtvozorništva, predložen je model mobilne aplikacije za vođenje mortalitetne statistike. Istodobno, opisana je arhitektura aplikacije MVP i korištene programske tehnologije (programski jezik *Java*, označni jezik XML, usluge *Google Maps* i *Firebase*). Kroz završni rad objašnjene su sve funkcionalnosti aplikacije, od unosa osobnih podataka umrle osobe, podataka o smrti osobe, s naglaskom na datum smrti, spol osobe i uzrok smrti koji predstavljaju bazu za statističku obradu podataka. Pored unosa i statističke obrade, korisniku je omogućena pohrana podataka preminule osobe, lociranje mjesta smrti i prikaz unosa na zaslon aplikacije. Provjerom ispravnosti unosa podataka i njihovim ispitivanjem u tri slučaja korištenja, potvrđeno je da aplikacija radi i može se koristiti u svrhu za koju je ostvarena.

Ključne riječi: geolokacija, mobilna Android aplikacija, mortalitetna statistika, predložak programske arhitekture MVP.

ABSTRACT

Title: **Mobile Android Application for monitoring the human mortality rate**

The principal objective of this final paper was a development of a mobile Android application for monitoring the human mortality rate. An application model comprises features based on described ideal solutions of mobile application and functions of the coroner. Also, the paper puts forward a model of application architecture MVP and all required software technologies (programming language *Java*, markup language XML, utilities from *Google Maps* and *Firebase)*. Throughout the paper, it explains functionalities from data input which consists of a dead person's personal and death information. The boldest emphasis is on a person's death date, gender, and death cause since it is a needed data for monitoring. Furthermore, users can record data, locate themselves and display data from the database. The summary of the application functionalities showed how three usage cases confirmed the quality of the application's capabilities.

Keywords: geolocation, mobile Android application, mortality rate, MVP architectural pattern.

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 Valpovu te ju završava 2018. godine. Iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, smjer preddiplomski studij Računarstvo.

_____

Fran Zamaklar

PRILOZI

[1] Prilog 1. Datoteka ZR_Konacna_Verzija_Fran_Zamaklar.docx završnog rada

[2] Prilog 2. Datoteka ZR_Konacna_Verzija_Fran_Zamaklar.pdf završnog rada

[3] Programski kod/projekt mobilne aplikacije