Mobilna iOS aplikacija za istraživanje znamenitosti grada podržano proširenom stvarnošću i postupcima strojnog učenja

Jurić, Jakov

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

Sveučilišni studij

MOBILNA iOS APLIKACIJA ZA ISTRAŽIVANJE ZNAMENITOSTI GRADA PODRŽANO PROŠIRENOM STVARNOŠĆU I POSTUPCIMA STROJNOG UČENJA

Diplomski rad

Jakov Jurić

Osijek, 2021.

SADRŽAJ

1. UVOD

Proširena stvarnost (engl. *Augmented Reality, AR)* pojam je koji se u današnje vrijeme sve češće pojavljuje, a primjena tehnologije proširene stvarnosti pronalazi se u svim područjima života. Unatoč činjenici da je pojam proširene stvarnosti već odavno poznat, tek sada zbog naglog napretka tehnologije proširena stvarnost postaje sve dostupnija i korisnija javnosti. U prošlosti su gotovo sve primjene proširene stvarnosti bile vrlo teško ostvarive, jer su zahtijevale stupanj tehnološke razvijenosti koji u to vrijeme nije postojao. Danas je tehnologija naprednija, a ljudi imaju mogućnost kupnje pametnih mobilnih uređaja koji bez poteškoća obrađuju zadatke proširene stvarnosti. Upravo zbog napretka tehnologije danas možemo vidjeti ogroman broj uređaja, aplikacija te igara koje koriste proširenu stvarnost u poslovne, edukativne ili zabavne svrhe. Iako je tehnologija napredovala, proširena stvarnost je daleko od savršene, postoje mnogi problemi koji utječu na mogućnost i preciznost simuliranja proširene stvarnosti.

U diplomskom radu potrebno je korištenjem mobilnih tehnologija, proširene stvarnosti i postupaka strojnog učenja omogućiti i potaknuti javnost na istraživanje grada i njegovih znamenitosti. Potrebno je predložiti model i arhitekturu mobilne aplikacije za iOS platformu koja će korištenjem prethodno navedenih tehnologija omogućiti ugodno korisničko iskustvo pri razgledavanju te istraživanju grada. Također, treba opisati mogućnosti i način korištenja tehnologija, jezika, alata i programskih okvira potrebnih za izradu opisane aplikacije. Nadalje, treba programski ostvariti opisanu mobilnu aplikaciju te ispitati i analizirati funkcionalnost aplikacije.

Struktura ovog diplomskog rada je sljedeća. Drugo poglavlje rada opisuje što je proširena stvarnost, ukratko je opisana povijest razvoja ove tehnologije te je načinjena usporedba proširene, virtualne i mješovite stvarnosti. Treće poglavlje opisuje trenutne tehnologije proširene stvarnosti koje su dostupne za upotrebu te su opisane razne postojeće implementacije i istraživački radovi proširene stvarnosti. Četvrto poglavlje predstavlja idejno rješenje i model, te opis tehnologija koje su korištene za razvoj mobilne aplikacije, opisani su razvojni alati i korišteni programski jezik. U petom poglavlju detaljno je opisano programsko rješenje koje prati glavne funkcionalnosti iz idejnog rješenja. Šesto poglavlje prolazi kroz razvijenu aplikaciju, a u njemu su analizirani klasifikacijski modeli, prikazani su rezultati ručnih i jediničnih testova te su izloženi nedostatci i moguća poboljšanja ostvarenog rješenja.

2. PROŠIRENA STVARNOST

Proširena stvarnost je tehnologija koja omogućuje kreiranje i pozicioniranje virtualnih objekata u stvarnom svijetu. Jednostavnije rečeno ova tehnologija kao što i pojam kaže proširuje stvarnost raznim virtualnim opisima. Tehnologija proširene stvarnosti postoji već neko vrijeme, a prvi ovakav funkcionalni sustav nastao je još u 1992. godini vođen Američkim ratnim zrakoplovstvom (engl. *United States Airforce)* pod nazivom *Virtual Fixtures.* Sustav *Virtual Fixtures* služio je za poboljšanje ljudskih performansi tijekom obavljanja raznih direktnih i daljinskih zadataka, a sastojao se od nosivog robotskog egzoskeleta, posebnih naočala koje su služile za fokusiranje pogleda na robotske ruke te računalno generiranih virtualnih objekata koji su usmjeravali korisnika (Slika 2.1 iz [14] i [15]).

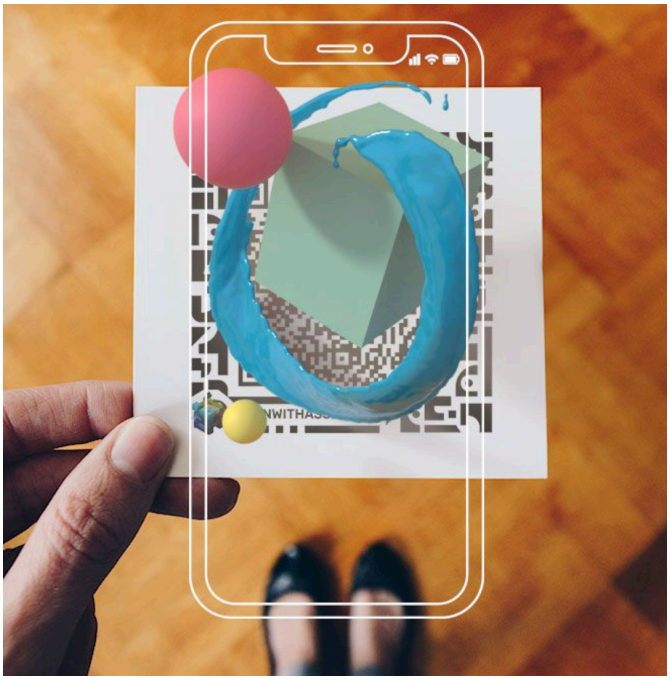
Slika 2.1. *Prikaz sustava Virtual Fixtures* [14]

Tijekom godina proširena stvarnost se implementirala i pokušala razvijati u različitim područjima medicine, vojske, terapije i edukacije, ali nikada nije zaživjela. Pojavom pametnih naočala i razvojem mobilne tehnologije proširena stvarnost se polako stabilizira i uvodi u svakodnevni život ljudi.

2.1. Podjela proširene stvarnosti prema pozicioniranju

Kada je riječ o načinu pozicioniranja i praćenja virtualnih objekata u stvarnom svijetu, proširenu stvarnost možemo podijeliti na dvije glavne skupine: *marker-based tracking* i *markerless tracking*.

Marker-based tracking predstavlja vrstu stvaranja virtualnih objekata uz pomoć jedinstvenih kartica, odnosno podloga (markera) za koje se virtualni objekti vežu. Pametni uređaj položajem, oblikom i veličinom podloge određuje veličinu i oblik virtualnog objekta kojeg stvara nad tom podlogom. Primjer *marker-based* praćenja prikazan je na slici 2.2 [16].



Slika 2.2. *Primjer marker-based praćenja* [16]

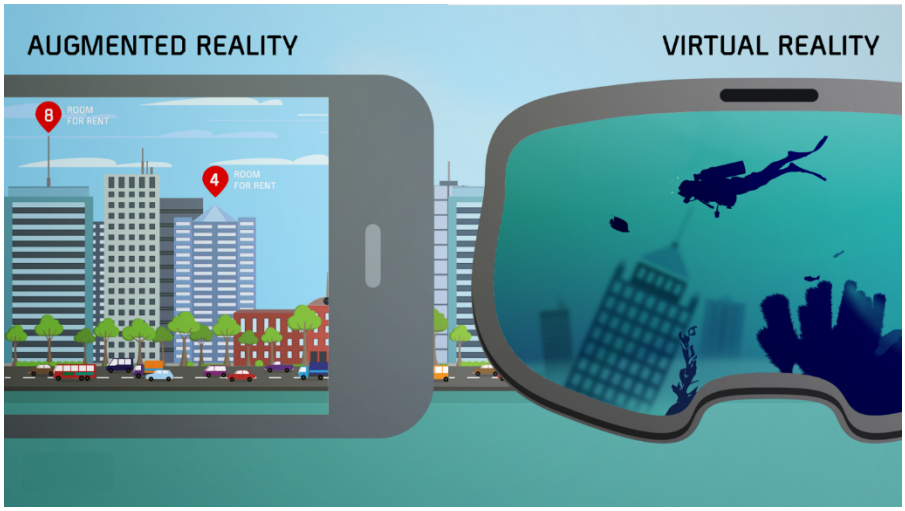
Markerless tracking je vrsta praćenja u kojem virtualni objekti nisu vezani za podlogu tj. karticu, već se smještaju na određenu poziciju u stvarnom svijetu. Ovakvim načinom praćenja virtualni objekti se mogu pozicionirati gotovo bilo gdje i na bilo kojoj podlozi, nisu ograničeni na stalni položaj nad definiranom plohom te se korisniku pruža bolja preglednost virtualnog objekta. Primjer ovakvog praćenja virtualnih objekata prikazan je na slici 2.3 [17].

Slika 2.3. *Primjer markerless praćenja* [17]

2.2. Proširena, virtualna i mješovita stvarnost

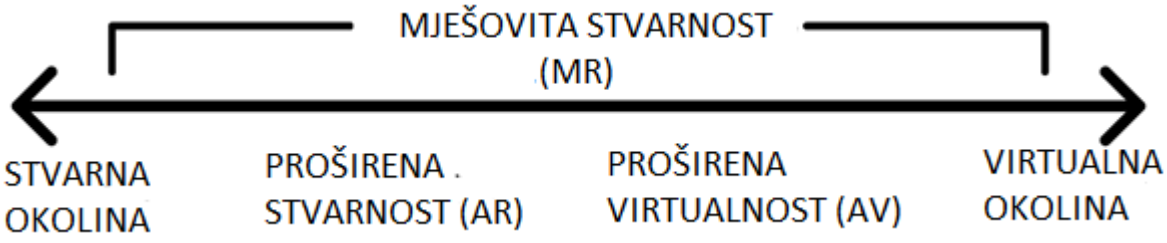
Osim spomenute proširene stvarnosti često se mogu čuti pojmovi poput virtualne i mješovite stvarnosti pa čak i produžena ili povećana stvarnost (engl. *Extended Reality).* Pojmovi unatoč sličnim imenima opisuju različite koncepte tehnologije.

Virtualna stvarnost (engl. *Virtual Reality, VR)* pojam je koji opisuje tehnologiju koja simulira umjetno, odnosno virtualno okruženje koje korisnika potpuno izdvaja iz stvarnog svijeta. Kako bi se korisnik izdvojio iz stvarnog svijeta najčešće se za ovakvu tehnologiju koristi neka vrsta VR naočala. Često se pojmovi virtualne i proširene stvarnosti miješaju te mogu djelovati zbunjujuće za nekoga tko nije upoznat s ovom tehnologijom. Najjednostavnije rečeno virtualna stvarnost izdvaja korisnika iz stvarnog svijeta i simulira potpuno virtualni svijet koji može i ne mora biti sličan stvarnom, dok proširena stvarnost ne izdvaja korisnika iz stvarnog svijeta, već dodatno opisuje stvarnu okolinu s virtualnim objektima [40]. Razlika između virtualne i proširene stvarnosti ilustrirana je na slici 2.4.



Slika 2.4. *Razlika između proširene i virtualne stvarnosti* [18]

Mješovita stvarnost (engl. *Mixed Reality, MR)* je prema [19] pojam koji kombinira elemente virtualne stvarnosti i proširene stvarnosti u jedinstveno iskustvo, drugim riječima mješovita stvarnost se ne odvija isključivo u virtualnom ili stvarnom svijetu, već predstavlja hibridno iskustvo. Spektar tehnologija, tj. stvarnosti koje spadaju pod mješovitu stvarnost prikazan je na slici 2.5.



Slika 2.5. *Spektar virtualnog i stvarnog svijeta [19]*

Zadnji pojam koji se može čuti je produžena ili povećana stvarnost, ali zbog nedostatka hrvatskih riječi ovaj pojam se često izmjenjuje s mješovitom stvarnošću. Naime, riječ je samo o zajedničkom skupu tehnologija virtualne, proširene i mješovite stvarnosti.

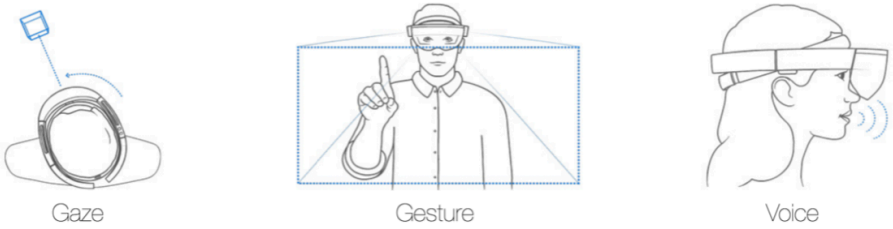
3. POSTOJEĆE TEHNOLOGIJE I ISTRAŽIVANJA PROŠIRENE STVARNOSTI

Tehnologija je u današnje vrijeme iznimno napredovala, već sada postoje mnogobrojni uređaji koji omogućuju implementiranje proširene stvarnosti u raznim područjima industrije i života općenito. Nažalost većina uređaja nije namijenjena za širu javnost i svakodnevni život ljudi, a razlog tomu je cijena tehnologije. Trenutno najpoznatiji uređaji koji koriste proširenu stvarnost su pametne naočale (engl. *smart glasses)*, a koriste ju kao alat za dodatno opisivanje i interakciju sa stvarnim svijetom. Dva najpoznatija primjera takvih naočala su Microsoft HoloLens i Google Glass.

3.1. Pametne naočale

Microsoft HoloLens

HoloLens su pametne naočale razvijene od strane Microsofta i reklamirane su kao naočale za mješovitu stvarnost. Prva inačica ovih naočala nastala je još u 2016. godini i na tržište je stavljena po cijeni od 3000 američkih dolara [1]. Naočale pomoću holografske leće i mnogobrojnih senzora pokreta i audio senzora omogućuju korisnicima interakciju s elementima proširene stvarnosti, bez potrebe za korištenjem dodatnih kontrolera. Interakcija s virtualnim objektima se vrši pomoću pogleda (engl. *gaze)*, odnosno točke ili kocke na sredini zaslona koja prati pokrete očiju (engl. *eye tracking)*, raznim gestikulacijama ruke (engl. *gestures*) te putem glasovnih naredbi, što je prikazano na Slici 3.1 [2].



Slika 3.1. *Načini interakcije s virtualnim objektima korištenjem HoloLens-a [2]*

Namjena je usmjerena poduzećima pa se tako među raznim aplikacijama pametnih naočala ipak ističu dvije najkorisnije, a to su *Dynamics 365 Remote Assist* i *Dynamics 365 Guides*.

Remote Assist je aplikacija koja omogućuje poduzećima direktan kontakt i suradnju s tehničkim stručnjacima neovisno o lokaciji. Stručnjaci mogu pratiti bilo koju situaciju uživo preko pametnih naočala koje koriste radnici i tako uz pomoć proširene stvarnosti pružiti pomoć i usmjeriti radnike na efikasno otklanjanje poteškoća.

Guides aplikacija je vrlo slična, ali je više usmjerena na edukaciju zaposlenika bez potrebe dodatnih stručnjaka ili edukatora. Aplikacija postepeno vodi radnike kroz razne poslovne procese, a uz pomoć integriranih slika, videa i virtualnih objekata cijeli proces postaje interaktivan i zabavan. Osim spomenutih integracija, aplikacija također pruža mogućnost analiziranja podataka zaposlenika, te na taj način pruža jedinstven uvid u korisnost poslovnih procesa [3].

Google Glass

Glass je vrsta pametnih naočala razvijena od strane Google X tima. Naočale nisu osmišljene s proširenom stvarnošću kao glavnim ciljem, već su zamišljene kao nosivi pametni mobilni uređaj. Naočale su dizajnirane drugačije od Microsoft-ovog HoloLens-a, zaslon za prikazivanje svih dodatnih informacija nalazi se samo na jednoj strani naočala s ciljem da se korisnika što manje izdvaja iz stvarnog svijeta. Osim zaslona, glavni dijelovi naočala su dodirna ploha (engl. *touchpad*) koja korisnicima omogućuje interakciju sa sučeljem te kamera kako je prikazano na sljedećoj slici (Slika 3.2).

Slika 3.2. *Google Glass* [4]

Unatoč nižoj cijeni pametne naočale *Glass* nisu zaživjele pa se tijekom godina Google također više posvetio poduzećima sa svojim *Enterprise Edition* naočalama. Neke od značajki kojima se

ističu su brzina procesora, kvaliteta videa i prenošenja slike (engl. *streaming)* te prethodno spomenuti mali zaslon koji ne zauzima cijelo vidno polje korisnika [5].

Iako su originalne namjene Google Glass i Microsoft HoloLens drugačije, oba uređaja koriste elemente proširene stvarnosti kako bi poboljšali korisničko iskustvo. Velika razlika i nedostatak Google Glass-a je manji broj mogućnosti za interakcijom s virtualnim elementima koji su većinom statični.

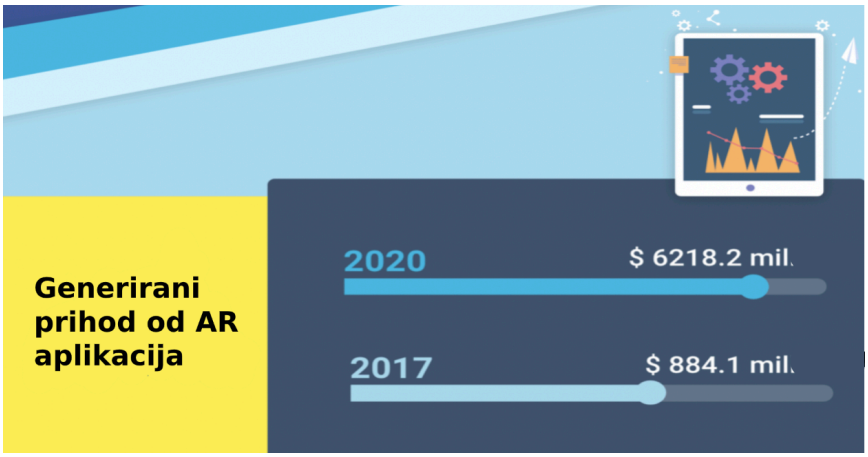
3.2. Proširena stvarnost u mobilnoj industriji

Razvoj pametnih naočala pruža jedinstvenu priliku isprobavanja raznih mogućnosti proširene stvarnosti, ali zbog velikih troškova proizvodnje, cijene krajnjeg proizvoda i slabe potražnje uređaji poput pametnih naočala nisu prihvaćeni od strane javnosti. No mobilna tehnologija i pametni telefoni zbog svoje dostupnosti polako postaju glavna platforma za eksperimentiranje s proširenom stvarnošću. Osim dostupnosti, pametni telefoni se ističu i zbog većih kapaciteta baterije, bolje povezanosti i već postoji ekosustav mobilnih aplikacija u koji se lako mogu dodati nove AR aplikacije koje mogu ili ne moraju implementirati dodatne uređaje poput pametnih naočala [7].

Kako raste popularnost i traženost za produktima proširene stvarnosti tako raste i broj aplikacija i programa. Razlozi popularnosti takvih aplikacija su brojni, ali najočitiji razlog je povećanje angažiranosti korisnika i bolja mogućnost obrade i pamćenja informacija (Slika 3.3).

Slika 3.3. *Pozitivne karakteristike aplikacija s proširenom stvarnošću* [8]

U zadnjih nekoliko godina broj mobilnih aplikacija koje koriste proširenu stvarnost eksponencijalno je porastao, a to potvrđuje i generirani prihod takvih aplikacija u zadnje tri godine, vidljiv na slici 3.4.



Slika 3.4. *Prihod aplikacija koje koriste proširenu stvarnost u zadnje 3 godine* [8]

Kada je riječ o mobilnoj industriji poznato je da su predstavnici Apple i Google sa svojim mobilnim operacijskim sustavima iOS i Android. Ove dvije tvrtke ponudile su dva najpoznatija programska okvira (engl. *framework)* za implementaciju proširene stvarnosti, a to su Apple-ov *ARKit* te Google-ov *ARCore*.

3.3. Istraživanja i radovi

Danas se nerijetko eksperimentira s primjenom tehnologija proširene stvarnosti unutar raznih industrijskih procesa, turizma, vojske pa čak i medicine. Uz razvoj mobilne tehnologije mogućnost eksperimentiranja s proširenom stvarnošću pruža se svakom pojedincu koji posjeduje pametni mobilni uređaj. Zbog toga se i među najkompliciranijim sustavima namijenjenim za simuliranje proširene stvarnosti mogu i dalje pronaći pametni mobilni uređaji i tableti kao dodatna pomagala. U nastavku slijedi nekoliko različitih istraživanja i radova o proširenoj stvarnosti i njenoj mogućoj primjeni u raznim dijelovima života.

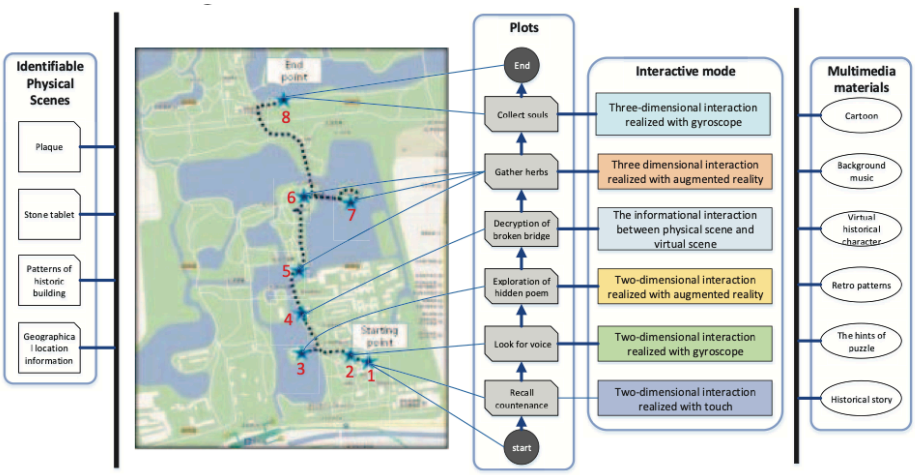
3.3.1. Proširena stvarnost u turizmu

Turizmu je oduvijek pogodovalo razvijanje tehnologije, ubrzalo se i olakšalo putovanje s jednog kraja svijeta na drugi, a orijentacija i pronalazak zanimljivosti nikad nisu bili lakši uz pomoć pametnih mobilnih uređaja. Osim mogućnosti kompasa, navigacijskih mapa i interneta,

mobilni uređaji se u turizmu sve češće koriste uz proširenu stvarnost kako bi se turisti više angažirali te kako bi im se cjelokupno iskustvo poboljšalo.

Primjer ovakve uporabe proširene stvarnosti istražen je i testiran na Pekinškom institutu za tehnologiju, gdje je napravljena interaktivna aplikacija (*MAGIC-EYES*) za istraživanje *Yuanmingyuan* parka koji sadrži brojne kulturne i povijesne ostatke. Primijećeno je da tradicionalnom načinu provođenja turističkih obilazaka parka nedostaje zabave i interakcije turista, zbog čega je napravljena aplikacija koja krši dotadašnja pravila te omogućuje turistima obilazak parka u obliku zabavne igre [20]. Aplikacija započinje generiranjem neobičnog puta kroz park koji se kreće kroz osam ključnih točaka na temelju kojih aplikacija može prepoznati je li osoba stigla na predviđeno mjesto (Slika 3.5). Aplikacija ima šest faza koje su napravljene u obliku misija koje korisnik mora obaviti kako bi uspješno prešao igru, odnosno kako bi uspješno završio obilazak. Korisnicima se pružaju brojne animacije i slike kako bi se objasnila legenda igre i povijest ruševina unutar parka. Osim korisnih informacija, aplikacija pomoću proširene stvarnosti omogućuje „pogled u prošlost“, odnosno generira virtualnu scenu koja predstavlja izgled tog parka u prošlosti.

Obavljeni su i testovi nad par turističkih grupa koji su koristili aplikaciju te su pokazali kako su grupe koje su koristile interaktivnu aplikaciju za obilazak parka generalno bile zadovoljnije i motiviranije za istraživanje parka i njegove prošlosti [20].



Slika 3.5. *Dizajn shema aplikacije MAGIC-EYES* [20]

Osim aplikacija koje su usmjerene na poboljšavanje turističkih obilazaka, postoje mnoge druge aplikacije koje nisu napravljene isključivo s turizmom na umu, ali su njihove funkcionalnosti

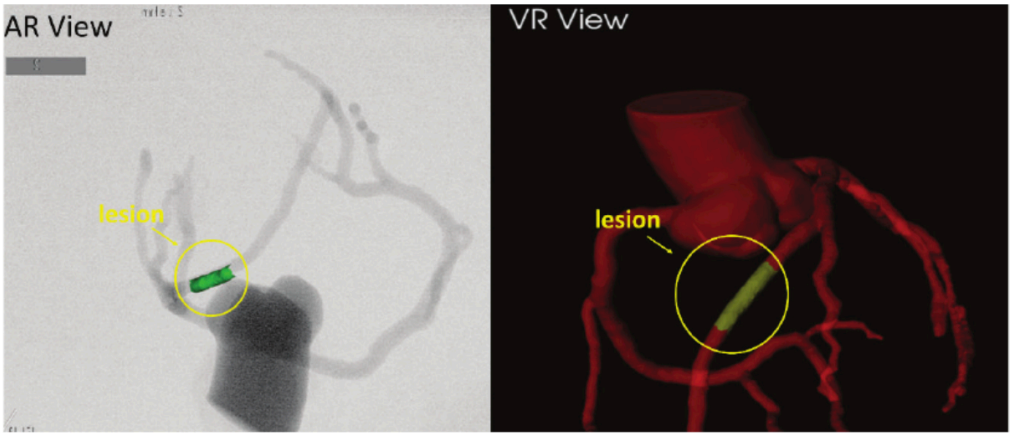
itekako pogodovale turizmu. Jedna od takvih aplikacija je i Google-ova mobilna aplikacija *Google Translate*, koja osim klasičnog prevoditelja ima i opciju korištenja kamere te u kombinaciji s proširenom stvarnošću i strojnim učenjem uživo prevodi bilo koji tekst sa stranog na proizvoljni jezik. Na taj način turistima se olakšava snalaženje u nepoznatim zemljama, smanjuje se mogućnost krivog tumačenja natpisa, omogućuje se čitanje do tada nepoznatih pisama i samim time se na određeni način povećava sigurnost i ugodnost turista u tuđoj zemlji.

Slika 3.6. *Primjer prevođenja s japanskog na engleski u aplikaciji Google Translate* [21]

3.3.2. Proširena stvarnost u medicini

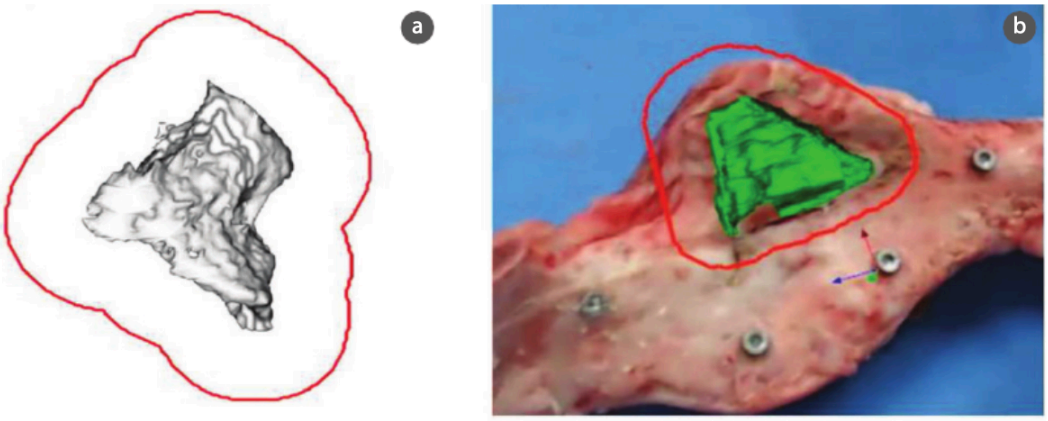
Istraživanja primjene proširene stvarnosti u medicini najčešće su vezana za integraciju unutar operacijske sale ili u edukacijske svrhe za pomoć pri vizualizaciji unutrašnjosti pacijenta, ali zbog određenih nepreciznosti i opasnosti najčešće ovakva istraživanja nisu odobrena za uporabu pri stvarnim operacijama.

Primjer ovakvog istraživanja je primjena tehnologije proširene stvarnosti u svrhe kardijalne kirurgije koja se najčešće bazira na 2D rendgenskim snimkama te iskustvu kirurga. Zbog toga se mogu pojaviti pogreške prilikom operiranja koje su uzrokovane krivim tumačenjem rendgenske slike zbog nezgodnog položaja pacijenta tijekom snimanja ili nejasnoće slike. Kao rješenje problema predloženo je korištenje proširene stvarnosti koja uz rendgenske slike kreira virtualni 3D model anatomije pacijenta te se tako pružaju jasnije informacije kirurgu (Slika 3.7 iz [22]).



Slika 3.7. *Primjer generiranog modela sustava proširene stvarnosti za kardiokirurgiju* [22]

Sličan primjer je navigacijski sustav proširene stvarnosti za resekciju tumora kosti koji također predstavlja kompliciran kirurški zahvat zbog međusobne isprepletenosti krvnih žila i živaca. Sustav je osmišljen za intuitivnu vizualizaciju koja bi dodatno pripremila kirurga za operaciju, a koristi običan tablet za praćenje pacijenta i kirurških alata. Aplikacija simulira virtualni objekt na položaju tumora te ga tako čini lakše uočljivim kirurgu tijekom operacije ili pregleda (Slika 3.8). Uspoređivanjem rezultata na testnom modelu pokazano je da kirurški zahvat obavljen uz pomoć tehnologije proširene stvarnosti pokazuje preciznije rezultate od zahvata obavljenog s tradicionalnim pristupom [22].

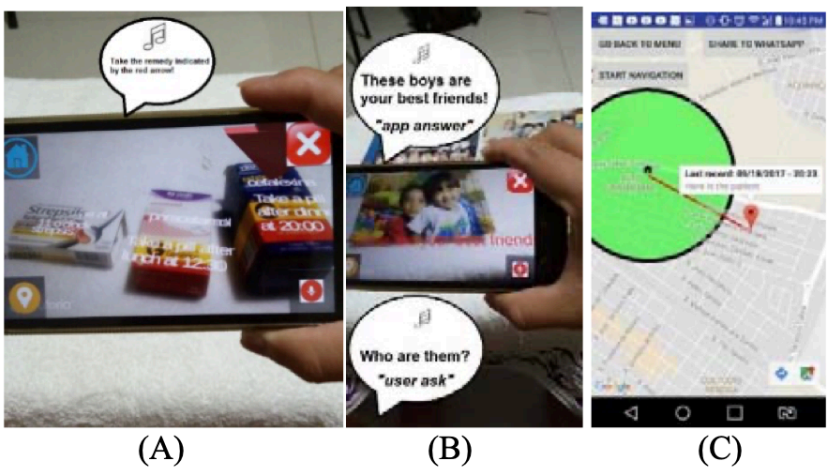


Slika 3.8. *Prikaz (a) 3D modela tumora i (b) AR prikaza tumora na testnom modelu* [22]

Na brazilskom sveučilištu izrađena je aplikacija koja umjesto usmjeravanja na operacijsku salu, tehnologiju proširene stvarnosti fokusira na pomoć oboljelima od Alzheimerove bolesti. Aplikacija je zamišljena da radi uz dodatnu pomoćnu tehnologiju (engl. *assistive technology*) poput pametnih naočala kako bi uz mobilni uređaj i glasovne naredbe olakšala život oboljelima.

Osim oboljelima aplikacija je osmišljena za pomoć skrbniku tako da prati lokaciju oboljelog u slučaju gubljenja u vanjskom prostoru. Aplikacija potiče bolesnike na terapeutske vježbe koje smanjuju negativne oblike ponašanja oboljelih od Alzheimerove bolesti, poput agresije, frustriranja i depresije [23]. Uz vježbe aplikacija također daje vizualne podsjetnike oboljelima za uzimanje i prepoznavanje ispravnih lijekova u pravodobno vrijeme (Slika 3.9). Aplikacija uz kombinaciju glasovnih naredbi omogućuje oboljelom da ispituje pitanja o slikama koje su u njegovoj okolini te mu pruža povratne informacije poput vremena kada je fotografija uslikana, tko su osobe na fotografiji i slično.

Aplikacija nije testirana na velikom broju ljudi, ali je zaključeno kako aplikacija olakšava određene aktivnosti poput uzimanja lijekova te olakšava uporabu starijim osobama pružanjem mogućnosti glasovnih naredbi.

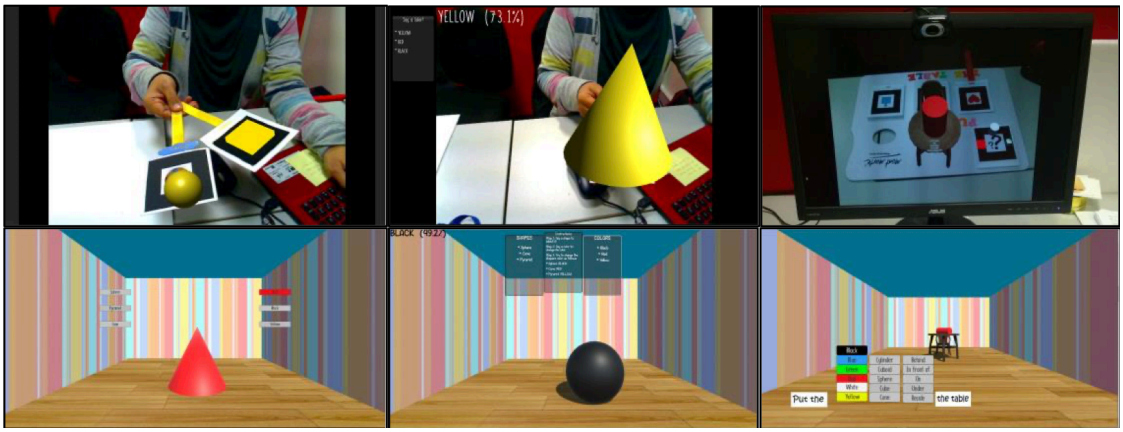


Slika 3.9. *Prikaz aplikacije i mogućnosti (A) prepoznavanja lijekova, (B) analiziranja slika i (C) praćenja lokacije oboljelog* [23]

3.3.3. Proširena stvarnost kao pomoć pri učenju

Danas se u obrazovnim institucijama sve češće koristi tehnologija kao alat za olakšano i efektivnije prenošenje znanja. Veliki problem standardnog načina prenošenja znanja je što se u većim grupama profesori teško mogu prilagoditi svakom pojedincu i njegovoj razini znanja, dok se alati poput edukacijskih programa mogu lako prilagoditi svakom pojedincu. Upravo to potvrđuju i mnogobrojna testiranja raznih edukativnih programa koja koriste tehnološki potpomognute programe, a u jednom takvom testiranju pokazano je da od 30 takvih programa njih čak 20 pokazalo je pozitivne promjene u učenju [24].

Primjer ovakve aplikacije je i program *TeachAR*, alat namijenjen za djecu od 4 do 6 godina koji služi kao pomoć pri učenju osnova engleskog jezika. Aplikacija se bazira na učenju engleskih naziva za boje, oblike i prostorne odnose, a interakcija je osnovana na glasovnim naredbama koja olakšava djeci korištenje aplikacije. Osim glasovnih naredbi, djeci s poteškoćama u govoru omogućeno je korištenje AR markera, koji osim što aktiviraju prikazivanje elemenata proširene stvarnosti služe i za interakciju s virtualnim objektima. Tako se na primjer putem AR markera aktivira pojavljivanje virtualnog 3D objekta kojeg dijete mora glasovnom naredbom prepoznati ili ukoliko ima govornih poteškoća, mora pokazati drugi AR marker s ispravnom riječi koja opisuje objekt. Testiranja su pokazala kako se djeci generalno sviđa učenje potpomognuto proširenom stvarnošću te su pokazali veću znatiželju za učenjem [25]. Primjer sučelja i korištena aplikacije *TeachAR* prikazan je na slici 3.10.



Slika 3.10. *Prikaz aplikacije TeachAR* [25]

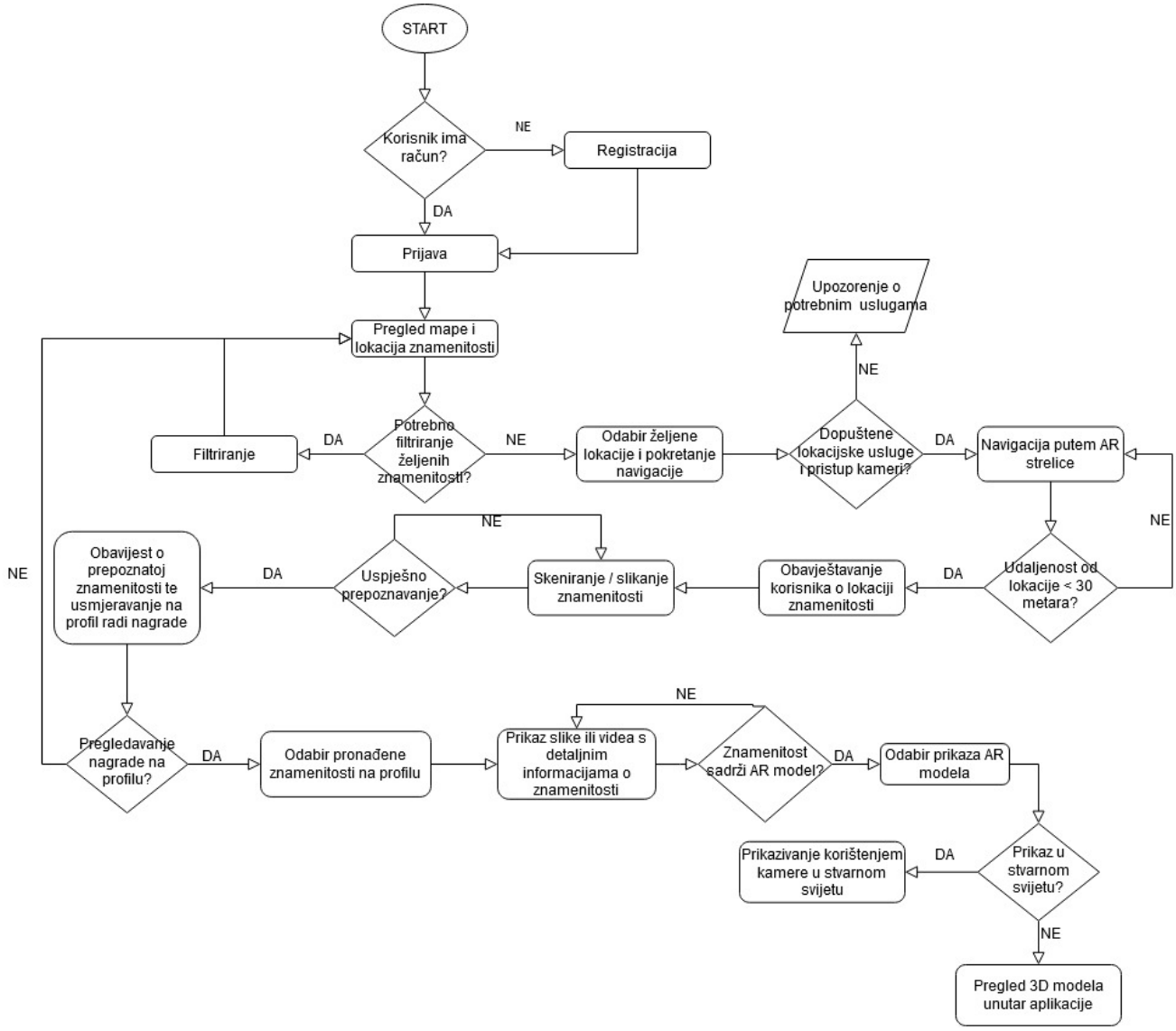
4. IDEJNO RJEŠENJE I MODEL MOBILNE APLIKACIJE

Osijek je grad koji nudi mnoštvo zanimljivih lokacija i znamenitosti od kojih većina ima bogatu povijest ili barem zanimljive činjenice koje su na određeni način dio kulture grada. Nažalost velik broj građana i turista ne znaju za priče iza tih znamenitosti, a razlog je najčešće nezainteresiranost za proučavanjem ili nepoznavanje povijesti grada. Upravo zbog toga je ideja ovog rada i aplikacije *AROsijek,* uporabom tehnologije ponovno zainteresirati ljude, a pogotovo one mlađe, na istraživanje grada, njegove povijesti i zanimljivosti koje nudi.

U ovom poglavlju prikazan je dijagram zamišljenog toka aplikacije, navedeni su glavni funkcionalni i pripadajući nefunkcionalni zahtjevi aplikacije te su navedene i ukratko opisane potrebne tehnologije, alati i programski okviri korišteni za ostvarenje zahtjeva aplikacije.

4.1. Dijagram toka aplikacije i zahtjevi

U nastavku slijedi prikaz dijagrama toka korištenja aplikacije *AROsijek* koji je prvobitno izrađen radi lakšeg vizualiziranja i izrade mobilne aplikacije te zbog toga ne sadrži sva stanja završne aplikacije. Dijagram predstavlja mogući scenariji korisničkog korištenja u kojem se prolazi kroz većinu stanja aplikacije.



Slika 4.1. *Dijagram toka aplikacije AROsijek (flowchart)*

Osim dijagrama toka određeni su i glavni zahtjevi koje aplikacija mora poštivati kako bi bila funkcionalna, ali i kako bi se poboljšalo korisničko iskustvo. U tablici 4.1 prikazani su neki glavni funkcionalni zahtjevi te njihovi odgovarajući nefunkcionalni zahtjevi.

FUNKCIONALNI ZAHTJEVI	NEFUNKCIONALNI ZAHTJEVI
Mogućnost autentificiranja korisnika	Jasne povratne poruke korisniku u slučaju grešaka
Navigacija korisnika	Upute za ispravnu navigaciju i povratne poruke u slučaju problema, te način za njihovo otklanjanje. Navigacija treba biti ostvarena uporabom AR tehnologije
Prikaz lokacija zanimljivosti i znamenitosti	Karta treba imati mogućnost filtriranja vrste znamenitosti te brzi pregled znamenitosti uz sliku i kratki opis
Pregled AR modela stražnjom kamerom	Mogućnost pregleda modela u stvarnom svijetu uporabom kamere, mogućnost pregleda unutar aplikacije u slučaju nepovoljnih vanjskih uvjeta
Prepoznavanje znamenitosti uporabom modela strojnog učenja	Potrebne povratne poruke o dodatnim informacijama i (ne)uspješnosti prepoznavanja, vremenski period prepoznavanja treba biti manji od 15 sekundi. Modeli ne smiju zauzimati previše mjesta u memoriji uređaja

Tablica 4.1. *Zahtjevi aplikacije AROsijek*

4.2. Potrebne tehnologije i usluge za ostvarenje ideje

Kako bi se zamišljeni tok aplikacije ostvario potrebne su određene tehnologije i usluge koje omogućuju ispravno funkcioniranje aplikacije. U sljedećim potpoglavljima opisani su alati i usluge potrebne za ostvarenje pozadinskog sloja (engl. *backend)* i baze podataka aplikacije, opisana je vrsta stojnog učenja kao i alat korišten za izradu modela potrebnih za mogućnost prepoznavanja znamenitosti te programski okvir za rukovanje s proširenom stvarnošću.

4.2.1. Baza podataka

Aplikacija za ispravan rad zahtijeva bazu podataka u koju se spremaju neophodne informacije o korisnicima poput informacija o otkrivenim znamenitostima, slike korisnika i slično. Uz bazu podataka potreban je i pozadinski sloj koji će rukovati autentifikacijskim zahtjevima korisnika te omogućiti sučelje za dodatno manipuliranje nad kreiranim računima. Zbog toga je za svrhu ove aplikacije odabrana besplatna platforma *Firebase*.

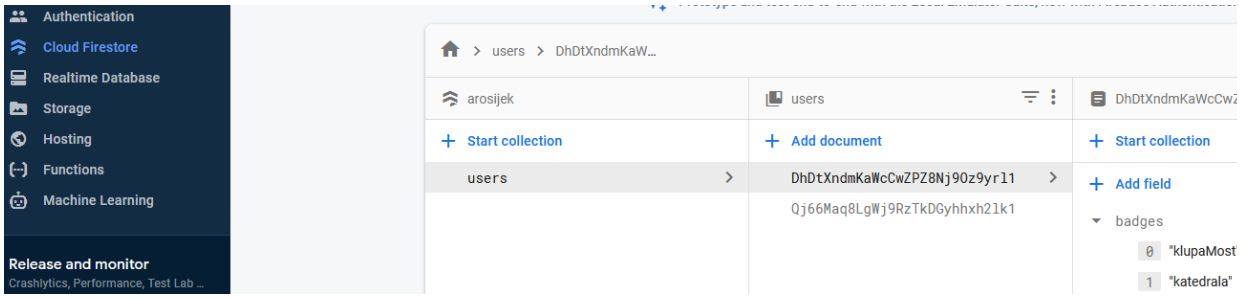
Firebase je Googleova mobilna platforma koja nudi veliki broj usluga za razvoj i održavanje mobilnih i web aplikacija. Firebase omogućuje brojne usluge koje su sasvim dovoljne za

ostvarivanje prethodno definiranih zahtjeva, a u nastavku je opisana svaka korištena Firebase usluga i njena uloga u aplikaciji *AROsijek.*

Firebase Auth - je usluga koja nudi olakšanu integraciju autentifikacijskih usluga unutar mobilnih i web aplikacija. Uslugu je vrlo lako postaviti u aplikaciji te uz nju dolazi i jednostavno web sučelje koje nudi mogućnosti pregleda i manipulacije korisničkih računa, ali i postavljanja uvjeta autentificiranja (npr. podržava li se prijava emailom, socijalnim mrežama, korisničkim imenom...). Kako bi se Firebase Auth implementirao potrebno je kreirati projekt na Firebase stranici, povezati aplikaciju putem *iOS bundle* identifikatora te ubaciti *FirebaseAuth* biblioteku koja sadrži sve potrebne funkcije za autentificiranje unutar aplikacije.

Firebase Firestore je usluga koja pruža pojednostavljenu bazu podataka u obliku kolekcija umjesto klasičnog JSON (*JavaScript Object Notation*) zapisa koji se može pronaći u ostalim bazama podataka. Firestore je više strukturiran, lako skalabilan te omogućuje dohvaćanje samo onih podataka koji nam trebaju umjesto povlačenja svih vezanih objekata iz JSON stabla poput standardnih baza podataka [32]. Na Firebase web sučelju također postoji jednostavno sučelje za pregled i manipuliranje kolekcijama koje se mogu opisati kao direktoriji u koje razvrstavamo potrebne podatke (Slika 4.2).

Firebase Storage kako bi se omogućilo spremanje većih podataka poput slika, korištena je usluga Storage jer Firestore ne podržava spremanje podataka koji su veći od 1 MB. Storage je vrlo slično organiziran kao i Firestore, ali nema kolekcija već su direktoriji hijerarhijski posloženi te se nazivaju djecom ili derivacijama (engl. *child).*



Slika 4.2. *Primjer Firebase web sučelja te mogućih usluga*

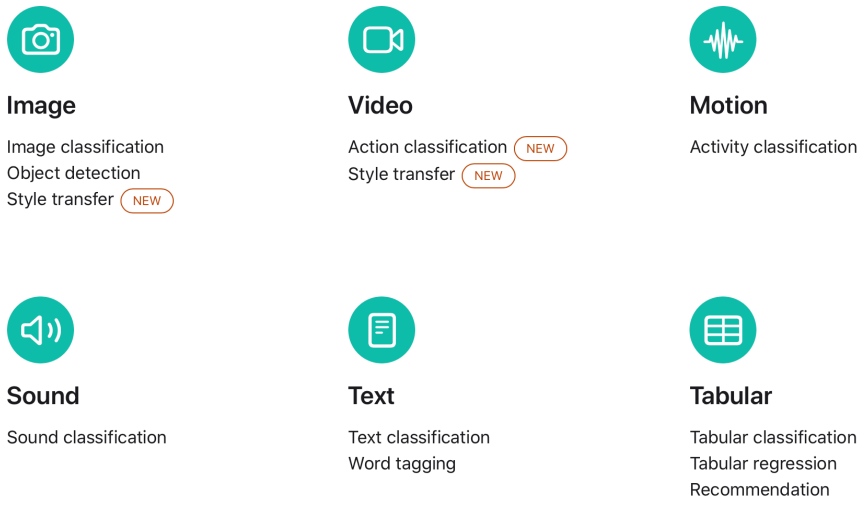
4.2.2. Strojno učenje i klasifikacija

Jedan od funkcionalnih zahtjeva aplikacije je mogućnost prepoznavanja znamenitosti i zanimljivosti putem kamere, a samim time podrazumijeva se neki oblik strojnog učenja. Aplikacija

AROsijek ima tri glavne skupine objekata koji se prepoznaju strojnim učenjem, a to su kipovi, građevine te skupina „istraži“ koja se sastoji od mješavine različitih objekata. U svakoj skupini postoje jasno predefinirane kategorije (klase) koje se trebaju prepoznati, a to su primjerice osječka konkatedrala u skupini građevine ili spomenik Miroslavu Krleži u skupini kipovi.

Jasno je da svaki objekt ima samo jednu klasnu oznaku te su unaprijed poznate vrijednosti tih klasa, stoga je izabran postupak nadziranog strojnog učenja, klasifikacija slika. Kako bi se klasifikacija omogućila potrebno je prvo stvoriti klasifikacijski model na temelju kojeg će se budući objekti klasificirati. Za svrhu kreiranja klasifikacijskog modela koristi se Apple-ov alat *CreateML*.

CreateML - je dio XCode programskog okruženja, a podržava nekoliko različitih tipova podataka kao što su slike, tekst i audio zapisi. CreateML koristi iOS infrastrukturu te su zbog toga modeli najčešće vrlo mali i lako se mogu integrirati unutar aplikacije [28]. Modeli se vrlo lako stvaraju i ne zahtijevaju poznavanje programskog koda, ali zbog dobre potpore za programski jezik Swift modeli se mogu kreirati, trenirati i dodatno poboljšavati i putem koda unutar aplikacija. Primjeri podržanih tipova podataka i vrste strojnog učenja prikazane su na slici 4.3.



Slika 4.3. *Popis tipova podataka koje CreateML podržava te mogućnosti treniranja [29]*

Za korištenje modela unutar aplikacije je bitno da je model u *.mlmodel* formatu, a ukoliko se koriste drugi alati za kreiranje modela strojnog učenja mogu se iskoristiti *Python* ekstenzije koje omogućuju pretvaranje modela u ispravni format. Trenutno podržani modeli koji se mogu

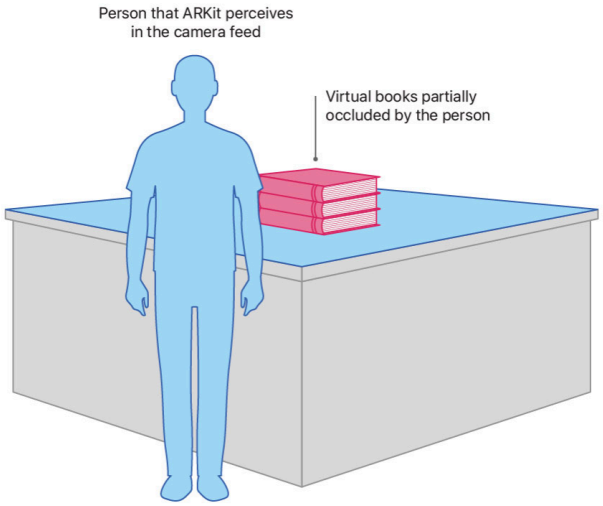
pretvoriti putem ekstenzija su: *PyTorch*, *TensorFlow*, *Multi-Backend Keras*, *ONXX Conversion* te *Caffe Conversion [30].*

4.2.3. Proširena stvarnost

Najvažnija funkcionalnost aplikacije je mogućnost korištenja proširene stvarnosti u svrhe pregledavanja virtualnih 3D modela u stvarnom svijetu te kao pomoć pri navigaciji korisnika. Kako bi se omogućila implementacija proširene stvarnosti potrebno je odabrati programski okvir predviđen za korištenje takve tehnologije. Zbog toga što je aplikacija predviđena za iOS uređaje koristi se prethodno spomenuti nativni Apple-ov programski okvir *ARKit*.

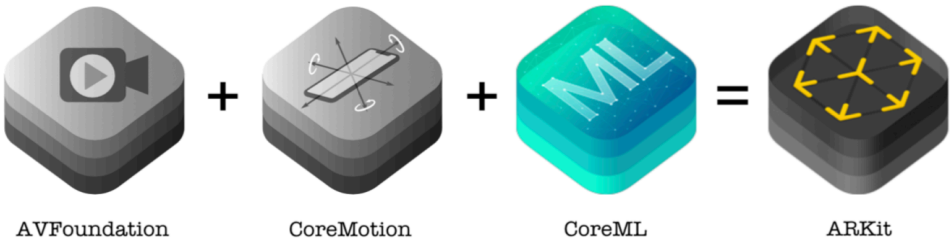
ARKit je razvojni okvir za izradu mobilnih aplikacija koje koriste proširenu stvarnost. Najjednostavnije rečeno ovaj okvir omogućuje programerima dodavanje raznih 2D i 3D virtualnih objekata u stvarnom vremenu na trenutnu sliku tj. video korisnikove kamere. Ovaj razvojni okvir poseban je po svojoj optimiziranosti, ali i mnoštvu funkcionalnosti koje nudi.

Omogućava praćenje i prepoznavanje 2D i 3D objekata, a uz to i izradu 3D točaka koje se kasnije mogu koristiti kao vlastiti referentni virtualni objekti. Također, podržava zajedničke sesije tj. dijeljena iskustva (engl. *collaborative sessions),* praćenje i prepoznavanje lica i pokreta tijela (engl. *face tracking, motion tracking)* što se može iskoristiti kao dodatni način interakcije. Jedna od najpoznatijih karakteristika je pak skrivanje objekata / osoba (engl. *people occlusion)* koja pomoću raznih procesa strojnog učenja omogućuje uvjerljivo pozicioniranje virtualnih objekata u stvarnom prostoru [2]. Primjerice ukoliko osoba stoji iza nekog virtualnog objekta, ta osoba neće biti vidljiva i obrnuto (Slika 4.4).



Slika 4.4. *Primjer skrivanja osoba/virtualnih objekata (people occlusion) [9]*

ARKit koristi funkcionalnosti već gotovih Apple-ovih okvira *CoreMotion*, *AVFoundation* i *CoreML*. CoreMotion služi za očitavanje položaja i kretnji uređaja u stvarnom svijetu, a te se informacije zatim spajaju sa stvarnom slikom kamere korištenjem AVFoundation okvira. Uz pomoć CoreML okvira omogućeno je lako prepoznavanje objekata i interakcija sa istim (Slika 4.5 iz [2]).



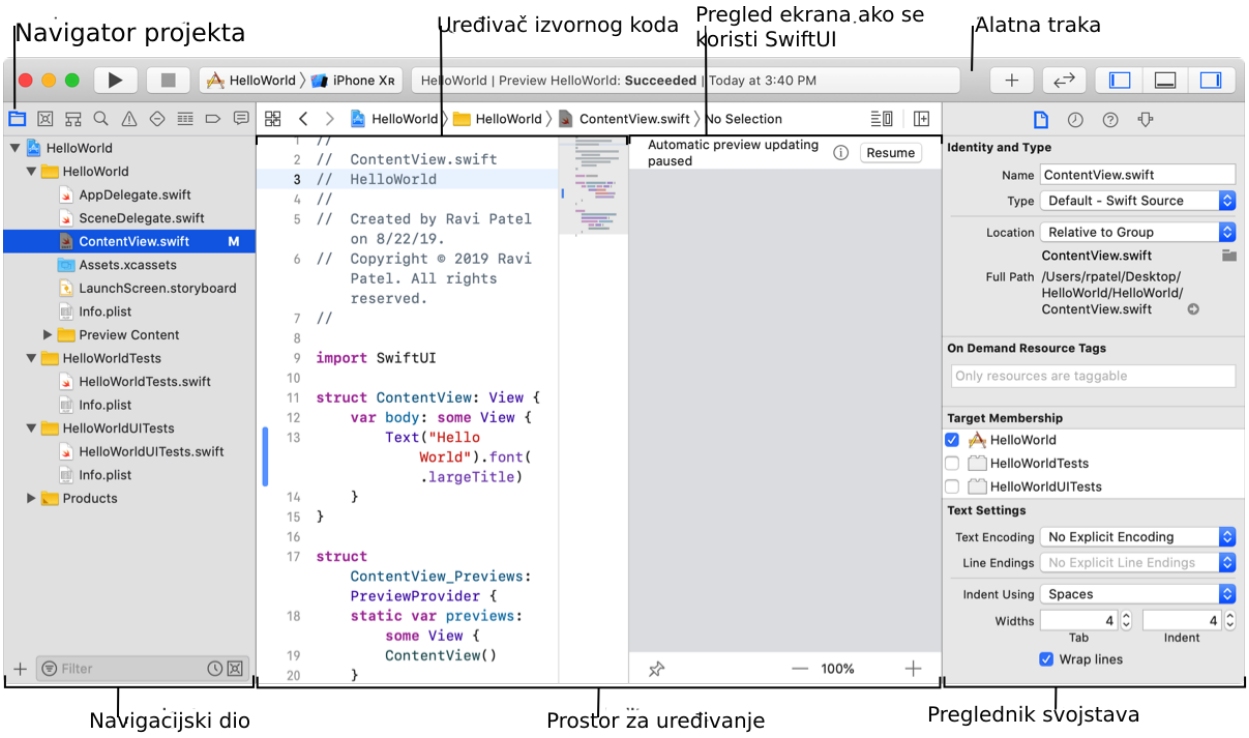
Slika 4.5. *Programski okviri koji čine ARKit* [2]

4.2.4. Razvojno okruženje XCode i programski jezik Swift

XCode predstavlja Apple-ovo integrirano razvojno okruženje (engl. *Integrated Development Environment, IDE)* i koristi se za razvoj i kreiranje aplikacija za sve Apple-ove proizvode. XCode nudi velik broj mogućnosti, ali i alata koji pomažu prilikom razvoja, testiranja i optimiziranja

aplikacije. Razvojno okruženje podržava programiranje koristeći Objective-C, Swift te SwiftUI programske jezike [26]. Za razvoj ove aplikacije korištene inačice XCode-a su 11.6 te inačica 12.

XCode je moguće koristiti samo na MacOS operacijskom sustavu te zbog toga zahtijeva i posjedovanje Mac računala. Razvojno okruženje pruža mogućnost pravljenja projekata koji sadrže sve potrebne resurse za razvoj aplikacija, ali također postoji opcija testiranja programskih kodova nazvanih *Playgrounds*. Brzi pregled glavnog sučelja XCode alata prikazan je na slici 4.6.



Slika 4.6. *Izgled XCode glavnog sučelja [26]*

Swift je programski jezik razvijen uz pomoć Apple-a, a osim što omogućava razvoj aplikacija za Apple uređaje, također nudi potporu za pisanje serverskog, odnosno poslužiteljskog koda. Vrlo je siguran jezik, brz i intuitivan jer sliči engleskom govornom jeziku.

Swift zaobilazi neke česte probleme ostalih programskih jezika jer se temelji na nekoliko osnovnih pravila:

1. Varijable se uvijek inicijaliziraju prije upotrebe
2. Polja i indeksi polja se uvijek provjeravaju jesu li unutar granica kako bi se zaobišle *out-of-bounds* pogreške
3. Cijeli (engl. *integer*) brojevi se provjeravaju za prekoračenje kapaciteta

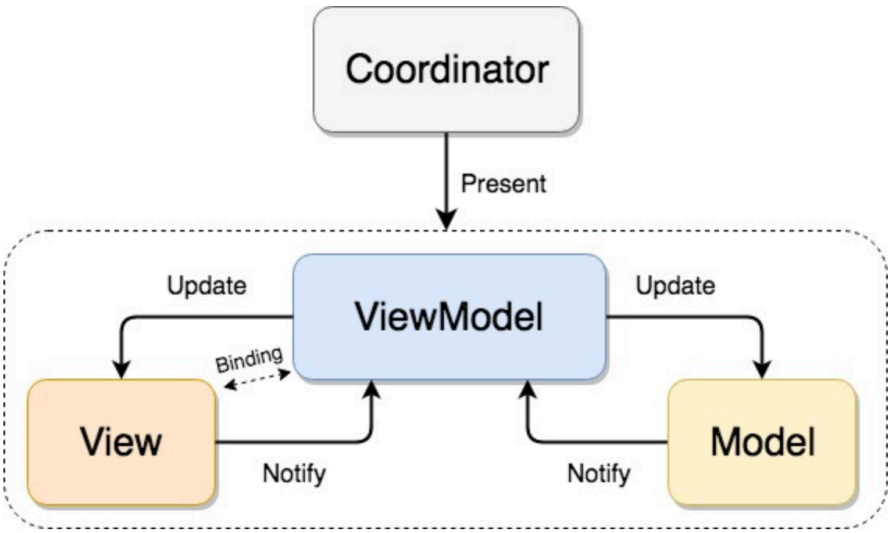
4. *Optional* vrijednost osigurava da se *nil* vrijednosti moraju zasebno rukovati
5. Memorija se rukuje automatski [27]

Prva dokumentacija i inačica Swift programskog jezika objavljena je još 2014. godine, a danas je posljednja inačica 5.3. te je upravo to inačica koja se koristila za razvoj ove aplikacije.

4.2.5. Programska arhitektura

Programska arhitektura aplikacije zasniva se na MVVM-C (*engl. Model-View-ViewModel-Coordinator*) obrascu. U ovom obrascu *model* sadrži potrebne modele podataka za ispravan rad poslovne logike, *view* predstavlja definirane pojedinačne elemente grafičkog sučelja kao i njihove kombinacije, *viewmodel* sadrži poslovnu logiku koja raznim metodama i uslugama manipulira podacima iz *modela* i na taj način utječe na sadržaj *view*-ova te *koordinator* koji upravlja tokom aplikacije, određuje koji *view*-ovi će se prikazivati te služi za lakše predavanje usluga *viewmodel*-u (engl. *dependency injection*).

Ovaj programski obrazac olakšava praćenje i kontroliranje toka aplikacije jer omogućuje kreiranje posebnih koordinatora za svaki poveći dio toka pa tako autentificiranje ima zaseban koordinator koji je zadužen za cijeli proces autentificiranja, dok profil ima koordinator koji je zaslužan za tok svih mogućih akcija na profilu korisnika. Svi pojedinačni koordinatori zatim su ujedinjeni u glavnom *Main* koordinatoru koji je ujedno zaslužan za kontroliranje toka cjelokupne aplikacije. U nastavku na slici 4.7 prikazan je međusobni odnos elemenata u MVVM-C obrascu.



Slika 4.7. *Prikaz arhitekture MVVMC* [31]

5. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE

U sljedećim poglavljima prikazano je programsko rješenje aplikacije *AROsijek* koje prati arhitekturni obrazac MVVM-C naveden u idejnom rješenju prethodnog poglavlja. Detaljno su opisani svi dijelovi programskog koda koji su podijeljeni u četiri glavne skupine koje odgovaraju prethodno definiranim funkcionalnostima aplikacije. Svaka skupina detaljno pojašnjava poslovnu logiku zasluženu za ostvarenje odgovarajuće funkcionalnosti te su opisani i potrebni podatkovni modeli kao i njihova uloga u poslovnoj logici.

5.1. Autentificiranje i pohrana podataka

Kako bi korisnici aplikacije mogli pratiti svoj napredak tijekom otkrivanja znamenitosti i zanimljivosti grada, a da se pri tome podaci ne spremaju na mobilni uređaj i bespotrebno zauzimaju memoriju, potrebno je osigurati bazu podataka u koju će se spremati svi korisnikovi podaci. Osim baze podataka potrebno je osigurati i neki oblik autentificiranja koji će omogućiti kreiranje korisničkog računa preko čijeg će se jedinstvenog identifikatora spremati i dohvaćati svi potrebni podaci.

Autentificiranje je ostvareno uporabom biblioteke *Firebase Auth* spomenute u idejnom rješenju, a prvobitno se treba stvoriti Firebase projekt i povezati ga sa iOS bundle identifikatorom kako je pojašnjeno unutar 4. poglavlja. Nakon kreiranja projekta i dodavanja odgovarajuće biblioteke moguće je koristiti sve funkcionalnosti vezane za autentificiranje korisnika, ali kako bi se funkcije oblikovale u jednostavniji oblik te kako bi se omogućila njihova višekratna uporaba kroz aplikaciju potrebne funkcije iz *Firebase*-ove biblioteke grupirane su u zasebnu autentifikacijsku uslugu koja obavlja svu potrebnu logiku i lako se predaje *viewmodel*-ima. Za ostvarenje autentifikacijske usluge prvo su kreirani *enum* modeli rezultata prema kojima će se određivati uspješnost autentifikacijskih zahtjeva prikazani na slici 5.1.

```
public enum AuthenticationResult<Value> {
    case success(Value)
    case failure(AuthenticationError)
}

public enum AuthenticationError: String {
    case alreadyExists = "User with that e-mail already exists."
    case networkError = "There was a problem with the Firebase network, please try again later."
    case emptyResult = "There was an error while creating the account, please try again."
    case general = "There was a problem with the authentication server, please try again later."
    case wrongInformation = "Wrong e-mail or password, please try again."
    case noInternet = "It seems like there is no internet, check your connection and try again."

    var textDescription: String {
        return self.rawValue
    }
}
```

Slika 5.1. *Enum modeli rezultata autentifikacijskih zahtjeva*

Kao što prethodna slika prikazuje model *AuthenticationResult* sastoji se od dva moguća ishoda, a to su *success* i *failure* koji imaju odgovarajuće tipove vezane za sebe. Slučaj *success* za sebe ima vezanu vrijednost *Value* koja označava bilo koju vrijednost dok slučaj *failure* za sebe ima vezanu vrijednost tipa *AuthenticationError* koja predstavlja zaseban *enum* model unutar kojeg se nalaze sve potrebne vrste pogrešaka koje se mogu dogoditi tijekom slanja autentifikacijskog zahtjeva. Svaka pogreška uz sebe ima i odgovarajuću poruku s kojom se može obavijestiti korisnika zašto je došlo do greške pa tako se na primjer slučaj *alreadyExists* pojavljuje ukoliko korisnik pokuša kreirati račun s postojećom email adresom. Nakon definiranih ishoda kreiran je protokol koji dodatno ograničava autentifikacijsku uslugu i jasno definira koje će se metode moći pozivati unutar *viewmodel*-a. Prikaz protokola *AuthenticationServiceProtocol* s metodama *register(...) i login(...)* prikazan je u nastavku na slici 5.2.

```
protocol AuthenticationServiceProtocol {
    func register(email: String, password: String, completion: @escaping (AuthenticationResult<User>) -> Void)
    func login(email: String, password: String, completion: @escaping (AuthenticationResult<User>) -> Void)
}
```

Slika 5.2. *Protokol AuthenticationServiceProtocol*

Prilikom kreiranja usluge za autentificiranje nasljeđuje se prethodno definirani protokol što znači da će usluga morati imati vlastitu implementaciju navedenih funkcija za registraciju i prijavu. Osim metoda iz protokola autentifikacijska usluga ima i privatnu dodatnu uslugu *ConnectivityService* za provjeru internetske veze jer *Firebase Auth* metode nemaju automatsku provjeru internetske veze u slučaju pogreške. Slika 5.3. prikazuje definiciju klase *AuthenticationService*, protokolske usluge za registraciju korisnika te konstruktor za kreiranje objekta usluge u koji se predaje spomenuta usluga za provjeru internetske veze.

```swift
class AuthenticationService: AuthenticationServiceProtocol {

    private let connectivityService: ConnectivityServiceProtocol

    init(connectivityService: ConnectivityServiceProtocol) {
        self.connectivityService = connectivityService
    }

    func register(email: String, password: String, completion: @escaping (AuthenticationResult<User>) -> Void) {
        Auth.auth().createUser(withEmail: email, password: password) { [weak self] (result, error) in
            if error != nil {
                self?.handleError(error: error, completion: completion)
                return
            }

            guard let user = result?.user else {
                completion(.failure(.emptyResult))
                return
            }

            completion(.success(user))
        }
    }
```

Slika 5.3. *Usluga za autentificiranje AuthenticationService*

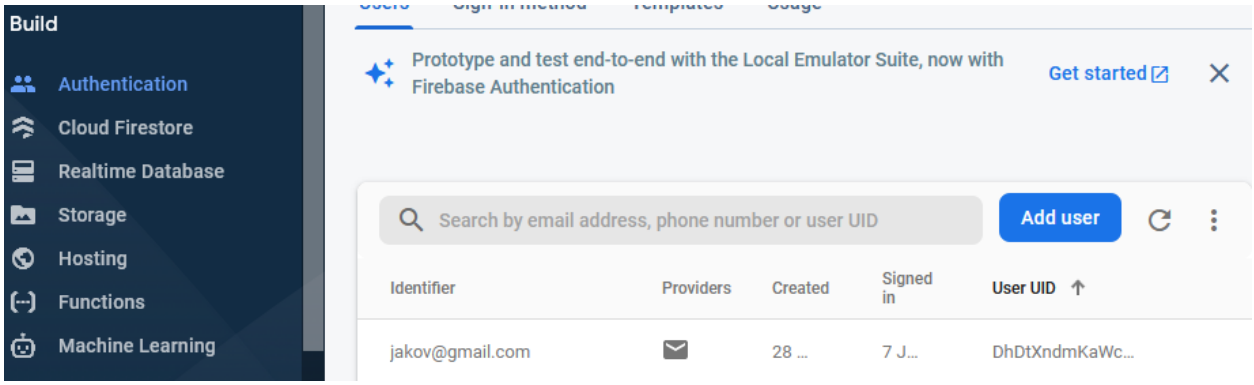
Na prethodnoj slici može se vidjeti implementacija protokolske metode *register*(...) koja prima parametre *email* i *password*, a sadrži i povratnu metodu *completion* preko koje se vraća odgovarajući ishod definiran na slici 5.1. Tijelo metode *register(...)* sastoji se od Firebase-ove metode *createUser(...)* koja ukoliko uspješno kreira korisnika vraća ishod *success* za koji se veže kreirani korisnik, a ukoliko kreiranje korisnika ne uspije vraća se ishod *failure* uz odgovarajući tip i poruku pogreške. Na slici 5.4 može se vidjeti uporaba autentifikacijske usluge i metode *register(...)* unutar *RegisterViewModel*-a koji je namijenjen za obavljanje sve potrebne logike za registraciju.

```swift
    func register(email: String, password: String) {
        onStartedActivity?()
        authenticationService.register(email: email, password: password) { [weak self] (result) in
            switch result {
            case .success(let user):
                print("*** Registered successfully ***")
                print("____________________")
                self?.createUser(email: email, userID: user.uid)
                //self?.saveToken(uid: user.uid)
                //self?.onFinished?()
            case .failure(let error):
                self?.onEndedActivity?()
                self?.onShowErrorMessage?(error.textDescription)
            }
        }
    }
```

Slika 5.4. *Primjer uporabe autentifikacijske usluga unutar RegisterViewmodel-a*

Kao što je vidljivo ukoliko je ishod *success* korisnikov email i jedinstveni identifikator se dodatno pohranjuju u bazu podataka, a ukoliko je ishod *failure* okida se povratna metoda

onShowErrorMessage() u koju se predaje tekst pogreške te se ona prikazuje na zaslonu za registraciju za koji je *RegisterViewModel* vezan. Kreirani korisnici mogu se pregledavati na Firebase web sučelju pod sekcijom *Authentication* prikazanoj na slici 5.5.



Slika 5.5. *Kreirani korisnik u Firebase web sučelju*

Osim autentificiranja i kreiranja korisničkog računa potrebno je i kreirati direktoriji u bazi podataka za svakog kreiranog korisnika kako bi se omogućilo spremanje važnih korisničkih podataka. Za potrebe baze podataka koriste se dvije Firebase usluge spomenute u idejnom rješenju, a to su *Firestore* i *Storage* koje su implementirane na sličan način kao i *Firebase Auth*. *Firestore* se koristi za spremanje malih tekstualnih podataka kao što su email korisnika te ključne riječi koje označavaju otkrivene znamenitosti i zanimljivosti korisnika. S druge strane *Database* se koristi za pohranjivanje podataka koji mogu biti veći od 1MB, a to su primjerice profilne slike korisnika koje se spremaju ukoliko korisnik učita vlastitu sliku s mobilnog uređaja.

Potrebne *Firestore* i *Storage* metode također su grupirane u vlastitu uslugu za komunikaciju s bazom podataka na sličan način kao i usluga *AuthenticationService.* Prvobitno je definiran *enum* model mogućih ishoda (*FirestoreResult)* prikazan na slici 5.6.

```swift
public enum FirestoreResult<Value> {
    case success(Value)
    case failure(FirestoreError)
}

public enum FirestoreError: String {
    case uploadError = "There was an error while uploading your data. Please try again."
    case urlError = "There was a problem gettiing the image URL. Please try again."
    case noInternet = "It seems like there is no internet, check your connection and try again."
    case downloadError = "There was an error while downloading your data. Please try again. "

    var textDescription: String {
        return self.rawValue
    }
}
```

Slika 5.6. *Enum model ishoda usluge za komunikaciju s bazom podataka*

Zatim je definiran protokol *DatabaseServiceProtocol* s četiri deklarirane metode: *createUser(...)* i *getUser(...)* za kreiranje i dohvaćanje korisničkog direktorija i njegovih podataka, *uploadImage(...)* za učitavanje vlastite korisničke profilne slike te metoda *updateBadges(...)* za osvježavanje otkrivenih znamenitosti i zanimljivosti. Definicija protokola može se vidjeti na slici 5.7.

```swift
protocol DatabaseServiceProtocol {
    func createUser(email: String, userID: String, completion: @escaping (FirestoreResult<Any?>) -> Void)
    func uploadImage(image: UIImage, userID: String, completion: @escaping (FirestoreResult<URL>) -> Void)
    func getUser(userID: String, completion: @escaping (FirestoreResult<UserData>) -> Void)
    func updateBadges(userID: String, badgeIdentifier: String, completion: @escaping (FirestoreResult<Any?>) -> Void)
}
```

Slika 5.7. *Definicija protokola DatabaseServiceProtocol*

Nakon definiranja protokola i mogućih ishoda implementirana je usluga za bazu podataka *DatabaseService* koja nasljeđuje prethodno prikazani protokol. Osim protokolskih metoda ova klasa također sadrži konstruktor koji prima uslugu za provjeru internetske veze po uzoru na autentifikacijsku uslugu. Klasa *DatabaseService*, njen konstruktor i metoda *createUser(...)* prikazani su na slici 5.8.

```swift
class DatabaseService: DatabaseServiceProtocol {

    private let db = Firestore.firestore()
    private let storage = Storage.storage().reference()
    private let connectivityService: ConnectivityServiceProtocol

    init(connectivityService: ConnectivityServiceProtocol) {
        self.connectivityService = connectivityService
    }

    func createUser(email: String, userID: String, completion: @escaping (FirestoreResult<Any?>) -> Void) {
        guard connectivityService.isConnected else {
            completion(.failure(.noInternet))
            return
        }

        let data: [String : Any] = ["email" : email,
                                    "badges" : [String]()]
        db.collection("users").document(userID).setData(data) { (error) in
            if let _ = error {
                completion(.failure(.uploadError))
                return
            }
            completion(.success(nil))
        }
    }
```

Slika 5.8. *DatabaseService usluga i metoda createUser(...)*

Na početku metode *createUser(...)* može se vidjeti provjera internetske veze putem usluge *connectivityService* nakon čega se kreiraju podatci uporabom *Firestore* metode *setData()* koja u direktoriju koji odgovara korisnikovom jedinstvenom identifikatoru (engl. *user identification, user ID*) kreira podatke *email* s korisnikovom email adresom i prazno polje *string*-ova koji će se popunjavati nakon što korisnik otkrije znamenitost uporabom metode *updateBadges(...)* deklariranom u protokolu na slici 5.7. Kao i kod autentifikacijske usluge ovisno o uspješnosti zahtjeva za kreiranjem direktorija vraća se *success* ili *failure*. Uporaba ove metode unutar *RegisterViewModel-a* viđena je na slici 5.4, a njena puna implementacija može se vidjeti na slici 5.9.

```swift
private func createUser(email: String, userID: String) {
    firestoreService.createUser(email: email, userID: userID) { [weak self] (result) in
        self?.onEndedActivity?()
        switch result {
        case .success(_):
            print("*** Created user in databse ***")
            print("____________________")
            self?.saveToken(uid: userID)
            self?.onFinished?()
        case .failure(let error):
            self?.onShowErrorMessage?(error.textDescription)
        }
    }
}
```

Slika 5.9. *Implementacija metode createUser(...) unutar RegisterViewModel-a*

5.2. Lokacijske usluge i navigacija

Jedni od funkcionalnih zahtjeva definiranih u idejnom rješenju su i mogućnost prikazivanja lokacija znamenitosti te navigacija korisnika do odabrane znamenitosti. Za ostvarenje karte s lokacijama svih znamenitosti korišten je programski okvir *GoogleMaps* dok se za navigaciju koristi *Apple-ov* nativni programski okvir *CoreLocation* namijenjen za sve lokacijske potrebe.

Kako bi se sva nužna svojstva lokacija znamenitosti grupirala na jedno mjesto kreiran je podatkovni model *LandmarkData* koji sadrži koordinate znamenitosti, naziv, kratki opis, sliku i kategoriju znamenitosti. Svi kreirani podatci znamenitosti i zanimljivosti zatim su grupirani u model *Landmarks* koji sadrži pojedinačne znamenitosti, ali i polje svih znamenitosti radi lakšeg pristupa svim podatcima. Definicija podatkovnih modela i primjer nekoliko kreiranih modela prikazani su na slici 5.10.

```
struct LandmarkData {
    var latitude: Double
    var longitude: Double
    var title: String
    var description: String
    var image: UIImage?
    var category: FilterCategoryType

    var coordinates2D: CLLocationCoordinate2D {
        return CLLocationCoordinate2D(latitude: latitude, longitude: longitude)
    }
}
struct Landmarks {
    static let allLandmarks: [LandmarkData] = [krleza, skoljka, klupaMost, picasso, konkatedrala]

    private static var krleza = LandmarkData(latitude: 45.558779, longitude: 18.6916306, title: "Miroslav Krleža", description: "Kip u spomen
        hrvatskom piscu Miroslavu Krleži", image: UIImage(named: "krleza"), category: .statue)
    private static var skoljka = LandmarkData(latitude: 45.558988, longitude: 18.692848, title: "Školjka", description: "Školjka je poznati simbol
        grada i mjesto okupljanja mladih", image: UIImage(named: "skoljka"), category: .other)
```

Slika 5.10. *Podatkovni modeli LandmarkData i Landmarks*

Na slici iznad može se vidjeti da je svako svojstvo strukture *Landmarks* statično, odnosno uz sebe ima riječ *static* što omogućuje pristupanje tom elementu bez inicijalizacije zasebnog objekta tipa *Landmarks*. Upravo ti podaci koriste se za kreiranje lokacijskih oznaka na Google karti koji su tipa *GMSMarker*, a primjer kreiranja oznaka prikazan je na slici 5.11.

```
private func configureMapMarkers(_ filters: [FilterCategoryType] = []) {
    mapView.clear()
    var allLandmarks = Landmarks.allLandmarks
    if !filters.isEmpty {
        allLandmarks = []
        for filter in filters {
            let filteredLandmarks = Landmarks.allLandmarks.filter({ $0.category == filter })
            allLandmarks.append(contentsOf: filteredLandmarks)
        }
    }
    for landmark in allLandmarks {
        let marker = GMSMarker()
        marker.position = landmark.coordinates2D
        marker.title = landmark.title
        marker.snippet = landmark.description + "\n(Tap to navigate)"
        marker.map = mapView
        marker.opacity = 0.7
    }
}
```

Slika 5.11. *Kreiranje lokacijskih oznaka tipa GMSMarker na Google karti*

Kao što se može vidjeti na prethodnoj slici za svaku znamenitost / zanimljivost kreira se *GMSMarker* oznaka, pozicija se stavlja na koordinate znamenitosti, a pritiskom na lokaciju otvara se informacijski prozor koji se sastoji od naziva, opisa i slike znamenitosti. Implementiranje vlastitog informacijskog prozora obavlja se unutar delegatske metode koja se poziva od strane *Google-ove* karte na svaki pritisak lokacijske oznake, što je vidljivo na slici 5.12.

```
func mapView(_ mapView: GMSMapView, markerInfoWindow marker: GMSMarker) -> UIView? {
    guard let landmark = Landmarks.allLandmarks.first(where: { $0.title == marker.title }) else { return nil }
    let infoView = MapInfoView(frame: CGRect(x: 0, y: 0, width: 300, height: 120))
    infoView.title = landmark.title
    infoView.shortDescription = landmark.description
    infoView.image = landmark.image
    return infoView
}
```

5.12. *Implementacija informacijskog prozora unutar delegatske funkcije*

Već je spomenuto da se u svrhe navigacije koristi programski okvir *CoreLocation* koji se sastoji od mnogobrojnih metoda i svojstava koji su opet radi lakše preglednosti i mogućnosti višekratne uporabe oblikovani u vlastitu lokacijsku uslugu. Lokacijska usluga prati obrazac prethodno opisanih usluga za autentificiranje i bazu podataka, prvo se kreiraju model ishoda i protokol koji je u ovom slučaju nešto veći jer osim metoda za pokretanje i zaustavljanje raznih lokacijskih usluga sadrži i povratne pozive. Povratni pozivi služe kao imitacija delegatskih metoda koje se okidaju unutar usluge, a na taj način se sprječava potreba za postavljanjem delegata unutar svakog *viewmodel-a* koji koristi lokacijsku uslugu. Lokacijski protokol *LocationServiceProtocol* sa svim metodama i povratnim pozivima prikazan je na slici u nastavku (Slika 5.13).

```
protocol LocationServiceProtocol {
    func startUpdatingLocation()
    func stopUpdatingLocation()
    func startUpdatingHeading()
    func stopUpdatingHeading()
    func startMonitoringRegion(coordinates: CLLocationCoordinate2D)
    var onGotHeading: ((Double) -> Void)? { get set }
    var onGotLocation: ((CLLocationCoordinate2D) -> Void)? { get set }
    var onLocationFailure: (() -> Void)? { get set }
    var onMonitoringFailure: (() -> Void)? { get set }
    var onEnteredRegion: (() -> Void)? { get set }
    var onExitedRegion: (() -> Void)? { get set }
    var onDidAuthorize: (() -> Void)? { get set }
}
```

5.13. *Definicija protokola LocationServiceProtocol*

Navigacija se vrši putem pozicioniranja navigacijske strjelice u smjeru odabrane znamenitosti ili zanimljivosti, a da bi se to ostvarilo prvo je potrebno dohvatiti korisnikovu lokaciju. Dohvaćanje korisnikove lokacije obavlja se putem metode *startUpdatingLocation()* koja je vezana za lokacijskog rukovatelja tipa *CLLocationManager*. Na lokacijskom rukovatelju se također mogu odrediti preciznost i udaljenost za dohvaćanje lokacije (vrijednosti *distanceFilter i desiredAccuracy)* kao i vrsta autorizacije potrebna za korištenje lokacijskih usluga (za svrhe ove aplikacije korištena je *requestAlwaysAuthorization* koja traži dopuštenje za konstantno osvježavanje korisnikove lokacije) (Slika 5.14).

```
lazy var locationManager: CLLocationManager = {
    let manager = CLLocationManager()
    manager.delegate = self
    manager.distanceFilter = kCLDistanceFilterNone
    manager.desiredAccuracy = kCLLocationAccuracyBest
    manager.requestAlwaysAuthorization()
    return manager
}()

func startUpdatingLocation() {
    locationManager.startUpdatingLocation()
}
```

Slika 5.14. *Metoda startUpdatingLocation i rukovatelj tipa CLLocationManager*

Kada se uspješno dohvati korisnikova lokacija poziva se delegat metoda *didUpdateLocation* u kojoj se zatim okida povratni poziv *onGotLocation()* definiran unutar lokacijskog protokola, a za njega se vežu posljednje korisnikove koordinate (Slika 5.15).

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    guard let currentLocation = locations.last?.coordinate else { return }
    onGotLocation?(currentLocation)
}
```

Slika 5.15. *Delegat metoda didUpdateLocation i povratni poziv onGotLocation()*

Osim korisnikove lokacije za svrhe navigacije potrebno je dohvatiti i azimut korisnika, odnosno njegovu orijentaciju kako bi se navigacijska strjelica mogla ispravno okretati. Za te svrhe koristi se metoda lokacijske usluge *startUpdatingHeading*() koja je kao i prethodna metoda za dohvaćanje trenutne lokacije vezana za lokacijskog rukovatelja. Prilikom dohvaćanja orijentacije ovaj put se poziva delegat metoda *didUpdateHeading* i okida se povratni poziv *onGotHeading(),* prikazano na slici 5.16.

```
func locationManager(_ manager: CLLocationManager, didUpdateHeading newHeading: CLHeading) {
    let heading = -(newHeading.trueHeading * .pi / 180)
    onGotHeading?(heading)
}
```

Slika 5.16. *Delegat metoda didUpdateHeading() i povratni poziv onGotHeading()*

Nakon dohvaćanja položaja i orijentacije dohvaća se kut između korisnikove lokacije i lokacije odabrane znamenitosti ili zanimljivosti (engl. *bearing*), a izračunava se pomoću formule prikazane na slici 5.17.

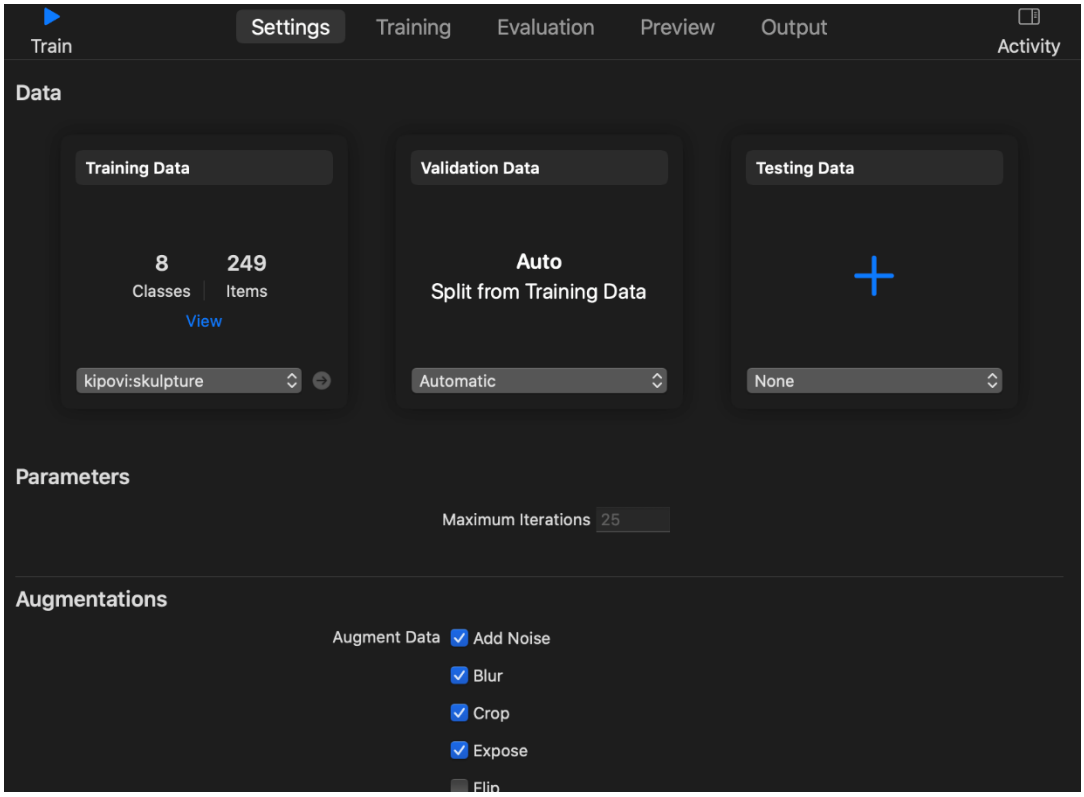
```
private func getRotation(for currentLocation: CLLocationCoordinate2D) {
    var deltaLongitude = location.longitude - currentLocation.longitude
    deltaLongitude = getRadians(from: deltaLongitude)

    let desiredLat = getRadians(from: location.latitude)
    let currentLat = getRadians(from: currentLocation.latitude)

    let y = sin(deltaLongitude) * cos(desiredLat);
    let x = cos(currentLat) * sin(desiredLat) - sin(currentLat) * cos(desiredLat) * cos(deltaLongitude)

    var rad = atan2(y, x)
    if rad < 0 {
        rad += 2 * .pi
    }

    self.locationBearing = rad
}
```

Slika 5.17. *Izračun azimuta (bearing) od trenutne do željene lokacije*

Oduzimanjem vrijednosti dobivene za *bearing* od korisnikovog azimuta dobiva se kut za koji treba zarotirati navigacijsku strjelicu kako bi pokazivala prema odabranoj lokaciji znamenitosti ili zanimljivosti.

5.3. Strojno učenje i klasifikacija znamenitosti

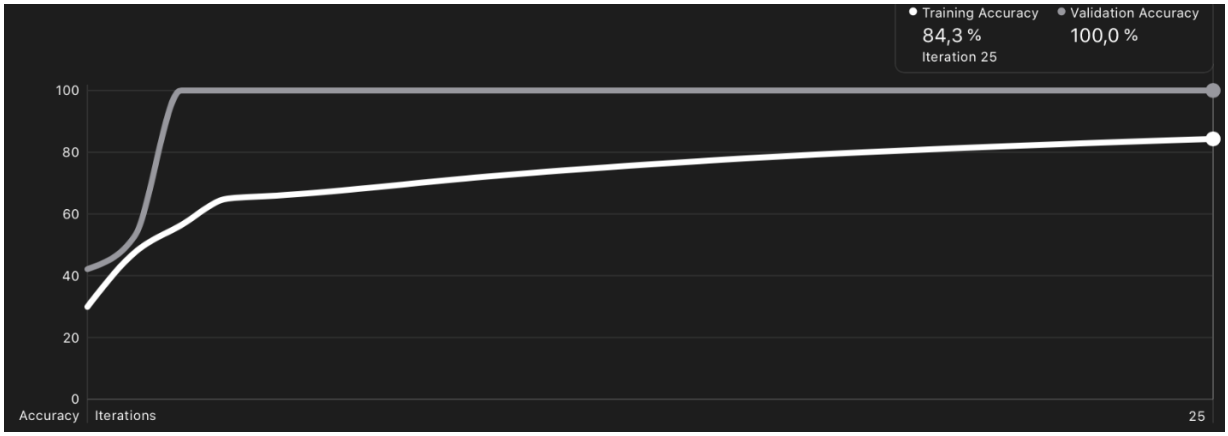
U idejnom rješenju već je spomenuta metoda nadziranog strojnog učenja, klasifikacija slika koja se koristi u svrhe prepoznavanja znamenitosti i zanimljivosti. Kako bi se pripremili odgovarajući modeli za klasifikaciju korišten je prethodno spomenuti alat Create ML, a modeli su podijeljeni u skladu s kategorijama znamenitosti: kipovi, građevine i kategorija istraži. Za kreiranje modela nije potrebno nikakvo programiranje već se sve radi preko jednostavnog sučelja, a primjer sučelja i konfiguracije za treniranje modela prikazan je na slici 5.18.



Slika 5.18. *Primjer konfiguracije za treniranje modela unutar alata CreateML*

Kao što je vidljivo na prethodnoj slici konfiguracija se sastoji od tri skupine podataka, skupina za treniranje podataka (engl. *training data),* validaciju podataka (engl. *validation data)* i testiranje podataka (engl. *testing data*). U primjeru na slici podatci za treniranje su učitani iz baze „kipovi / skulpture“, a sastoji se od osam klasa od kojih svaka odgovara nazivu neke skulpture ili kipa te svaka klasa sadrži trideset ili više slika objekta iz različitih kuteva. Podatci za validaciju su

automatski izabrani od strane alata Create ML dok su podatci za dodatno treniranje ostavljeni prazni jer nisu obavezni za treniranje modela. Tijekom konfiguracije se osim podataka određuju i proširenja (engl. *augmentations*) nad učitanim skupovima podataka kao što su dodatna zamućenja, dodavanje šuma, podrezivanja slike i rotiranje. Svrha proširenja je da povećaju broj podataka za treniranje te samim time povećaju stupanj uspješnosti tijekom korištenja klasifikacijskog modela. Zadnji parametar tijekom konfiguracije odnosi se na maksimalni broj dozvoljenih iteracija jer Create ML nakon raspodjele na podatke za treniranje i validaciju izvlači karakteristike iz svih slika te iterativno trenira model [38]. Povećavanjem broja iteracija može se povećati uspješnost prepoznavanja slika, ali ukoliko se broj iteracija postavi na preveliku vrijednost istrenirani model može donositi lažno pozitivne rezultate. Za svrhe ove aplikacije broj iteracija postavljen je na predefiniranu vrijednost od maksimalno 25 iteracija. U nastavku na slici 5.19 prikazan je graf preciznosti treniranja i validacije tijekom svake obavljene iteracije te krajnja evaluacija uspješnosti treniranja svake klase.



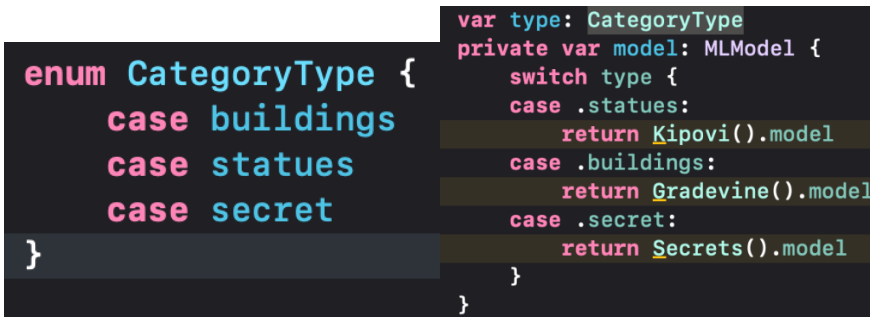
kipovi:skulpture
Nov 18, 2020 at 5:46 PM

Class	Item Count	Precision	Recall
josipStross	1820	82%	74%
klupaMost	1820	90%	96%
krleza	2145	82%	84%
lavoslavRuzicka	1690	85%	74%
majkaiDijete	1950	87%	91%
mijoKispatic	1820	84%	84%
rat	2015	83%	88%
vladimirPrelog	1690	82%	82%

Slika 5.19. *Graf preciznosti (iznad) i evaluacija (ispod) uspješnosti istreniranog modela*

Istrenirani modeli su nakon toga ubačeni u projekt, a kako se ne bi morali učitavati svi modeli odjednom napravljen je *enum (CategoryType)* koji sadržava moguće kategorije znamenitosti te se

na temelju odabranog tipa ujedno određuje i model za učitavanje. Primjer kreiranog *enum-a* za tipove kategorija te odabir odgovarajućeg modela prikazan je na slici 5.20.



Slika 5.20. *Enum CategoryType (lijevo), određivanje odgovarajućeg modela klasifikacije (desno)*

Kako bi se u programskom kodu mogao stvoriti zahtjev za klasificiranje slike ubačene putem korisnikove kamere korišten je nativni programski okvir *Vision* koji osim mogućnosti korištenja Create ML modela za klasifikaciju sadrži i razne algoritme za prepoznavanje značajki lica, barkodova, teksta i općenitih značajki sa slike [39]. Prvo je definiran zahtjev za klasifikaciju (*VNCoreMLRequest*) koji prvobitno učitava odgovarajući model, a zatim provjerava ima li zahtjev rezultate koji su tipa *VNClassificationObservation* jer je upravo to vrsta rezultata koja se dobiva klasifikacijom slika. Ukoliko zahtjev ima rezultata dohvaća se prvi rezultat jer je to rezultat koji ima najveći stupanj sigurnosti te se on prosljeđuje metodi *handleResult(...)*. Opisani zahtjev za klasifikaciju prikazan je na slici 5.21.

```
private lazy var classificationRequest: VNCoreMLRequest = {
    guard let model = try? VNCoreMLModel(for: model) else { fatalError() }
    let request = VNCoreMLRequest(model: model, completionHandler: { [weak self] (request, _) in
        if let results = request.results as? [VNClassificationObservation] {
            let topResult = results[0]
            self?.handleResult(topResult)
        }
    })
    request.imageCropAndScaleOption = .centerCrop
    return request
}()
```

Slika 5.21. *Kreirani zahtjev za klasifikaciju slike VNCoreMLRequest*

Važno je spomenuti kako prethodna slika prikazuje samo definiciju zahtjeva za klasifikaciju dok se zahtjev, odnosno rukovatelj zahtjeva (*VNImageRequestHandler*) poziva na zasebnoj niti (*userInitiated*) koja osigurava da korisnik ne može koristiti aplikaciju dok se obavlja sama

klasifikacija. Rukovatelju se uz zahtjev predaje i slika koju je korisnik prethodno uslikao ili učitao, a pozivanje kreiranog zahtjeva na zasebnoj niti te rukovanje mogućim pogreškama prikazano je u nastavku (Slika 5.22).

```
DispatchQueue.global(qos: .userInitiated).async { [unowned self] in
    let handler = VNImageRequestHandler(ciImage: ciImage, orientation: orientation)
    do {
        try handler.perform([self.classificationRequest])
    } catch {
        DispatchQueue.main.async { [unowned self] in
            self.onEndedActivity?()
            self.onShowErrorMessage?("Error while classifying image, please try again")
        }
    }
}
```

Slika 5.22. *Pozivanje zahtjeva za klasifikaciju na zasebnoj niti userInitiated*

Metoda *handleResult(...)* spomenuta u definiciji zahtjeva za klasifikaciju provjerava zadovoljava li proslijeđeni rezultat postavljenu granicu (engl. *treshold*) sigurnosti za tip kategorije, odnosno modela. Ukoliko rezultat ima stupanj sigurnosti veći ili jednak od zadane granice poziva se metoda *updateUserBadge(...)* koja ažurira korisnikove podatke u bazi podataka kako je i opisano u potpoglavlju 5.1. Ukoliko rezultat ne zadovoljava zadanu granicu sigurnosti korisniku se prikazuje poruka (*onShowErrorMessage)* koja ga obavještava o nemogućnosti prepoznavanja te ga usmjerava na ponovno slikanje znamenitosti. Prikazivanje poruke i ažuriranje podataka prebacuje se na glavnu nit (*DispatchQueue.main*) zbog rukovatelja zahtjeva koji se pozivao na niti koja blokira korisnikovu interakciju (Slika 5.23).

```
private func handleResult(_ result: VNClassificationObservation) {
    print("CLASSIFIED AS: \(result.identifier) WITH CONFIDENCE: \(result.confidence)")
    var confidenceTreshold = treshold
    if result.identifier == "rat" {
        confidenceTreshold = 0.92
    }
    DispatchQueue.main.async { [weak self] in
        if result.confidence >= confidenceTreshold {
            self?.updateUserBadge(badgeIdentifier: result.identifier)
        } else {
            self?.onEndedActivity?()
            self?.onShowErrorMessage?("The image could not be recognized, please try again")
        }
    }
}
```

Slika 5.23. *Prikaz metode handleResult(...)*

5.4. Uporaba proširene stvarnosti

Proširena stvarnost je jedna od glavnih funkcionalnosti aplikacije, a kako je već spomenuto korištena je za prikazivanje navigacijskih elemenata te pregledavanje 3D modela otkrivenih znamenitosti. Simuliranje proširene stvarnosti omogućeno je prethodno opisanim nativnim programskim okvirom *ARKit*.

Kako bi se omogućilo prikazivanje virtualnih objekata i njihovo smještanje u stvarnom svijetu potrebno je prvo postaviti scenu za proširenu stvarnost (*ARSCNView*) te kao njenog delegata postaviti glavni *ViewController* u kojem se vrši navigacija, odnosno prikaz virtualnih elemenata. Na sceni su također postavljene dodatne opcije za automatsko ažuriranje i prikazivanje zadanog svjetla kako bi virtualni objekti izgledali što realnije te kako bi korisnik imao dojam da se svjetlo iz njegove okoline reflektira na virtualnim objektima (Slika 5.24).

```
private lazy var sceneView: ARSCNView = {
    let sceneView = ARSCNView()
    sceneView.delegate = self
    sceneView.automaticallyUpdatesLighting = true
    sceneView.autoenablesDefaultLighting = true
    view.addSubview(sceneView)
    return sceneView
}()
```

Slika 5.24. *Postavljanje scene za proširenu stvarnost (ARSCNView)*

Osim scene kreirana je i konfiguracija proširene stvarnosti kojom je definiran način praćenja pokreta i scene u određenoj sesiji. Konfiguracija također sadrži razne postavke za dodatno kontroliranje scene poput mogućnosti pružanja zvuka, način prikazivanja tekstura okoline, mogućnost detektiranja horizontalnih i vertikalnih površina te mnoge druge opcije. Za potrebe navigacije korištena je konfiguracija *ARWorldTrackingConfiguration* koja nadgleda poziciju i orijentaciju uređaja u stvarnom svijetu koristeći 6DOF (engl. *Six Degrees Of Freedom*) metodu kojom se prati kretanje uređaja u smjeru x,y i z-osi te tri vrste rotacije uređaja (engl. *roll, pitch, yaw).* Također, postavljeno je ishodište koordinatnog sustava (*worldAlignment)* na samu kameru jer se navigacijski elementi moraju nalaziti uvijek ispred korisnika, odnosno ispred kamere uređaja. Prikaz postavljanja konfiguracije i pokretanje unutar sesije prethodno kreirane scene prikazano je na slici 5.25.

```
private func runARSession() {
    let configuration = ARWorldTrackingConfiguration()
    configuration.worldAlignment = .camera
    configuration.providesAudioData = false
    configuration.environmentTexturing = .automatic

    sceneView.session.run(configuration, options: [.removeExistingAnchors, .resetTracking, .resetSceneRecons
}
```

Slika 5.25. *Kreiranje i pokretanje konfiguracije na sceni*

Nakon kreiranja scene i pokretanja konfiguracije moguće je učitati virtualne objekte u scenu pomoću čvorova (engl. *node)* koji su najčešće dio zasebnih scena zvanih *SCNView. SCNView* je dio Apple-ovog programskog okvira *SceneKit* na koji se *ARKit* nadovezuje pa se stoga svi elementi iz *SceneKit-a* ujedno mogu koristiti i u *ARSCNView* sceni. Tako je prvobitno učitana scena u kojoj se nalazi navigacijska strjelica (*ArrowGPS.scn*) te je iz učitane scene izvučen čvor koji sadrži objekt navigacijske strjelice. Zatim su postavljene proporcije objekta (engl. *scale*), određen je položaj objekta na koordinatnom sustavu, spremljena je referenca čvora te je objekt napokon ubačen u glavni čvor (engl. *rootNode*) scene (Slika 5.26).

```
private func loadNodes() {
    // Arrow node
    guard let arrowScene = SCNScene(named: "SceneObjects.scnassets/ArrowGPS.scn") else {
        print("couldn't load arrow scene")
        return
    }
    guard let arrowNode = arrowScene.rootNode.childNode(withName: "arrow", recursively: false) else {
        print("Couldn't load arrow node")
        return
    }
    arrowNode.scale = SCNVector3(0.08, 0.08, 0.08)
    arrowNode.position.z = -1
    self.arrowNode = arrowNode
    self.arrowNode?.isHidden = true
    sceneView.scene.rootNode.addChildNode(self.arrowNode!)
```

Slika 5.26. *Učitavanje objekta (čvora) navigacijske strjelice te ubacivanje u scenu*

Prethodna slika prikazuje učitavanje navigacijske strjelice koja se nalazi u zasebnoj sceni zvanoj *ArrowGPS*, izvlačenje čvora imena *arrow*, skaliranje čvora te postavljanje *z-osi* čvora na -1 što odgovara jednom metru ispred kamere uređaja, jer kao što je spomenuto ishodište koordinatnog sustava se nalazi u kameri. Na sličan način uz dodatno skaliranje i dodavanje vlastitih animacija na čvor, obavlja se učitavanje virtualnog objekta / čvora za lokacijski marker što je vidljivo na slici 5.27.

```
// Location node
guard let locationScene = SCNScene(named: "SceneObjects.scnassets/Location.scn") else {
    print("Couldn't load location scene")
    return
}
guard let locationNode = locationScene.rootNode.childNode(withName: "locationMarker", recursively: false) else {
    print("Couldn't load location node")
    return
}
locationNode.position.z = -1
locationNode.scale = SCNVector3(0.001, 0.001, 0.001)
let animation = SCNAction.rotateBy(x: 0, y: .pi * 2, z: 0, duration: 5)
animation.timingMode = .easeInEaseOut
let loop = SCNAction.repeatForever(animation)
```

Slika 5.27. *Učitavanje čvora lokacijskog markera te dodavanje animacija*

Osim prikazivanja navigacijskih objekata mora se vršiti i rotacija navigacijske strjelice pomoću izračuna spomenutih u potpoglavlju 5.2. kako bi se korisnika usmjeravalo na lokaciju odabrane znamenitosti ili zanimljivosti. Rotacija je obavljena u delegat metodi koja se poziva svake sekunde tijekom prikazivanja *ARSCNView* scene na glavnom *ViewControlleru* zaduženom za navigaciju te se unutar nje postavlja rotacija po y-osi čvora navigacijske strjelice kako je prikazano u nastavku.

```
// MARK: - ARSCNView Delegate extension

extension NavigationViewController: ARSCNViewDelegate {
    func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
        arrowNode?.eulerAngles.y = -Float(viewModel.currentHeading + viewModel.locationBearing + .pi)
    }
}
```

Slika 5.28. *Rotacija navigacijske strjelice po y osi unutar delegat metode*

Kao što je vidljivo iz prethodnih opisa prikazivanje virtualnih objekata proširene stvarnosti tijekom navigacije ne zahtjeva dodatno rukovanje vertikalnih i horizontalnih površina jer su objekti i koordinatni sustav smješteni na kameru uređaja. Pomicanjem i rotiranjem uređaja pomiču se i svi virtualni objekti koji su dodani u scenu što odgovara potrebama navigacije, ali ne i potrebi pregledavanja 3D modela znamenitosti. Korisnik pregledavanjem objekta treba moći obilaziti objekt sa svih strana te pomicati i rotirati uređaj bez da se virtualni 3D model pomiče, drugim riječima koordinatni sustav scene proširene stvarnosti mora se nalaziti u stvarnom svijetu. Kako bi se postiglo željeno ponašanje objekata potrebno je postaviti parametar *worldAlignment* konfiguracije za proširenu stvarnost na vrijednost *gravity* umjesto prethodno postavljene vrijednosti *camera*. Parametar *gravity* postavlja y-os koordinatnog sustava paralelno s gravitacijom, a ishodište se nalazi na poziciji na kojoj je prvobitno pokrenuta sesija za prikazivanje scene proširene stvarnosti. Osim promjene položaja koordinatnog sustava potrebno je detektiranje željene površine (vertikalne / horizontalne) postavljanjem parametra konfiguracije

planeDetection. Detektirane površine stvaraju sidra (engl. *anchors*) s informacijom o položaju i orijentaciji željene površine u stvarnom svijetu na čiju poziciju se dodaje čvor virtualnog objekta. Radi izbjegavanja nepotrebnog konfiguriranja scene, praćenja površina, rukovanja pogreškama te dodatnog navođenje korisnika korišten je Apple-ov programski okvir *QuickLook* koji automatski rukuje svim prethodno spomenutim zahtjevima.

Za omogućavanje *QuickLook* pregleda 3D modela potrebna je provjera sadrži li odabrana znamenitost 3D model proširene stvarnosti, što se iščitava nakon uspješne klasifikacije znamenitosti opisane u potpoglavlju 5.3. Naime nakon uspješnog prepoznavanja objekta osim što se informacija sprema u bazi podataka korisnika, također se putem jedinstvenog identifikatora kojeg sadrži svaka klasa u istreniranim klasifikacijskim modelima kreira *enum* objekt *ClassificationData*. Svaki slučaj unutar *ClassificationData* ujedno odgovara predanom identifikatoru klase, a primjer svih slučajeva vidljiv je na slici 5.29.

```
enum ClassificationData: String {
    // Statues
    case krleza = "krleza"
    case josipStross = "josipStross"
    case klupaMost = "klupaMost"
    case lavoslavRuzicka = "lavoslavRuzicka"
    case majkaiDijete = "majkaiDijete"
    case mijoKispatic = "mijoKispatic"
    case rat = "rat"
    case vladimirPrelog = "vladimirPrelog"

    // Buildings
    case evangelistickaCrkva = "evangelistickaCrkva"
    case dvoracPejacevic = "dvoracPejacevic"
    case katedrala = "katedrala"
    case skoljka = "skoljka"

    // Other
    case katakombeVrata = "katakombeVrata"
    case picasso = "picasso"
    case sfinge = "sfinge"
    case zdenacFontana = "zdenacFontana"
```

Slika 5.29. *ClassificationData enum i mogući slučajevi identifikatora klase*

Na temelju prikazanih slučajeva određeno je koja znamenitost sadrži sliku, video i/ili model proširene stvarnosti te je na taj način omogućeno lako rukovanje svakim klasificiranim objektom. Varijabla *arModel* unutar *enum-a* predstavlja naziv 3D modela za proširenu stvarnost, a ukoliko ta vrijednost ne postoji to označava da znamenitost ne sadrži model već sadrži neku drugu vrstu

medijskog sadržaja poput slike ili videa. Prikaz varijable unutar *ClassificationData enum-a* te način njenog iščitavanja prikazan je na slici u nastavku (Slika 5.30).

```
var arModel: String? {
    switch self {
    case .klupaMost:
        return "bridge"
    case .katedrala:
        return "Konkatedrala"
    default:
        return nil
    }
}
```

Slika 5.30. *Varijabla arModel unutar ClassificationData enum-a*

Provjeravanjem vrijednosti varijabli *arModel* i *videoLink* određen je i tip medijskog sadržaja otkrivene znamenitosti na korisnikovom profilu (*BadgeType*) koji ujedno određuje sadržaj koji će se prikazivati u detaljima te znamenitosti (*image, video* ili *arModel*) kao što je prikazano na slici 5.31.

```
class BadgeInfoViewModel: BaseViewModel {

    let badge: ClassificationData
    let connectivityService: ConnectivityServiceProtocol

    var onNoInternet: (() -> Void)?
    var type: BadgeType {
        if let _ = badge.arModel {
            return .arModel
        } else if let _ = badge.videoLink {
            return .video
        } else {
            return .image
        }
    }
```

Slika 5.31. *Određivanje tipa (type) medijskog sadržaja znamenitosti*

Na temelju prethodno određenog tipa na zaslonu za prikaz detalja znamenitosti (*BadgeInfoViewController*) se dodatno konfigurira sučelje, npr. ukoliko je tip *image* za sučelje se postavlja grafički element za prikaz slike, a ukoliko je tip *video* na sučelje se postavlja element za prikaz videa kao što je prikazano slikom 5.32.

```
switch viewModel.type {
case .image:
    imageView.anchorToSuperview()
case .arModel:
    imageView.anchorToSuperview()
    arButton.anchor(top: (mediaView.topAnchor, 1), trailing: (mediaView.trailingAnchor, 1), size: CGSize(width: 50, height: 50))
case .video:
    videoView.anchorToSuperview()
}
```

Slika 5.32. *Postavljanje grafičkog sučelja na temelju tipa medijskog sadržaja iz viewmodel-a*

Iz prethodne slike vidljivo je da ukoliko znamenitost sadrži model za proširenu stvarnost na sučelje je postavljena slika znamenitosti, a na sliku se dodaje posebno dugme za prikazivanje 3D modela (*arButton*). Kako bi se sadržaj mogao prikazivati kreiran je *QuickLook* rukovatelj za prikazivanje sadržaja (*QLPreviewController*) kojemu se kao delegat i izvor podataka (engl. *data source*) također postavlja trenutni zaslon za prikaz detalja znamenitosti (Slika 5.33).

```
private lazy var previewController: QLPreviewController = {
    let previewController = QLPreviewController()
    previewController.delegate = self
    previewController.dataSource = self
    return previewController
}()
```

Slika 5.33. *QuickLook rukovatelj za prikazivanje sadržaja (QLPreviewController)*

QuickLook rukovatelj definiran je kao *lazy* varijabla što znači da se objekt rukovatelja neće inicijalizirati sve dok se ne pozove unutar programskog koda (npr.pritiskom na kreirani gumb *arButton*). Kako je trenutni zaslon postavljen kao delegat i izvor podataka *QuickLook* rukovatelja, usvojene su i dvije posebne delegatske metode koje su implementirane u zasebnoj ekstenziji klase *BadgeInfoViewController* prikazane u nastavku (Slika 5.34).

```
extension BadgeInfoViewController: QLPreviewControllerDelegate, QLPreviewControllerDataSource {
    func numberOfPreviewItems(in controller: QLPreviewController) -> Int {
        return 1
    }

    func previewController(_ controller: QLPreviewController, previewItemAt index: Int) -> QLPreviewItem {
        let modelUrl = Bundle.main.url(forResource: viewModel.badge.arModel ?? "", withExtension: ".usdz")!
        return modelUrl as QLPreviewItem
    }
}
```

Slika 5.34. *Delegatske metode izdvojene u zasebnoj ekstenziji glavne klase*

Prva delegatska metoda *numberOfPreviewItems* kao što samo ime kaže određuje broj podataka koji će se prikazivati u rukovatelju, a kako se za svaku znamenitost može najviše prikazivati jedan 3D model (ukoliko ga sadrži) u toj metodi vraćen je broj jedan. Druga metoda sa slike 5.34. s parametrom *previewItemAt,* određuje koji podatak i kojeg tipa će se prikazivati na određenom indeksu. Unutar metode učitan je resurs s odgovarajućim imenom modela proširene stvarnosti (*arModel*) te je on vraćen kao *QLPreviewItem* čime se omogućuje njegovo prikazivanje unutar rukovatelja. Na kraju je definirana i metoda prethodno kreiranog gumba *arButton* koja prezentira *QuickLook* rukovatelja preko trenutnog zaslona, a time ga i inicijalizira jer se unutar te metode prvi puta referencira *previewController* (Slika 5.35).

```
@objc private func onARButtonTapped() {
    present(previewController, animated: true, completion: nil)
}
```

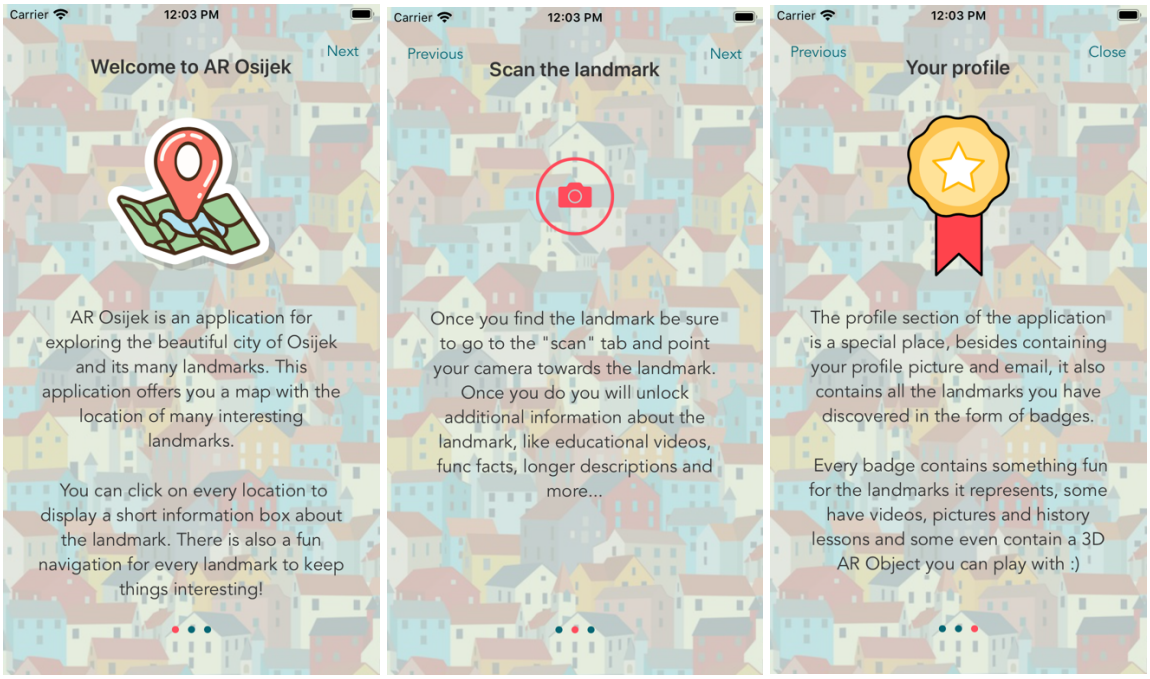
Slika 5.35. *Metoda (akcija) gumba arButton*

Prethodno definiranim metodama i delegatskim pozivima je omogućeno prikazivanje modela proširene stvarnosti putem *QuickLook* preglednika. Korisnik može putem preglednika simulirati virtualni objekt u proširenoj stvarnosti ili ga može pregledavati na uređaju kao običan 3D model, a osim toga *QuickLook* omogućuje korisniku dijeljenje i spremanje modela na uređaj.

6. PRIKAZ I TESTIRANJE RJEŠENJA MOBILNE APLIKACIJE

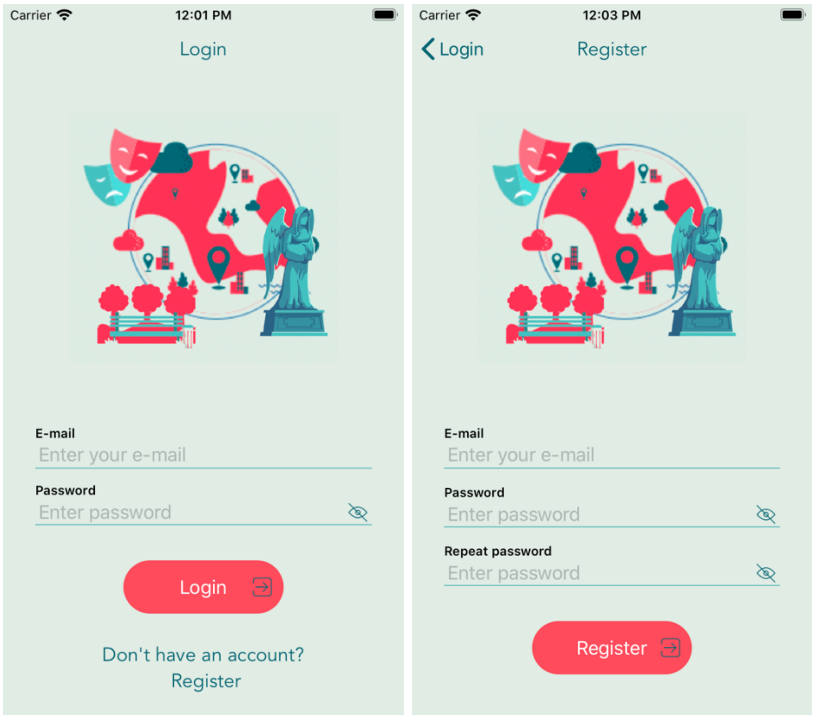
U ovom poglavlju ukratko su opisani svi zasloni aplikacije, prikazani su neki jedinični (engl. *unit*) testovi kojima je provjerena poslovna logika *viewmodel-a*, predstavljeni su nedostatci i problemi trenutne aplikacije te su predložena moguća poboljšanja aplikacije u pogledu korisničkog iskustva i mogućnosti koje aplikacija pruža.

Početni zaslon aplikacije prikazuje kratki uvod u aplikaciju (engl. *onboarding*) kako bi se korisnika obavijestilo o svrsi i mogućnostima koje mu se pružaju. Svaki *onboarding* zaslon sadrži naslov funkcionalnosti, animaciju i kratki tekst koji pobliže opisuje funkcionalnost (Slika 6.1).



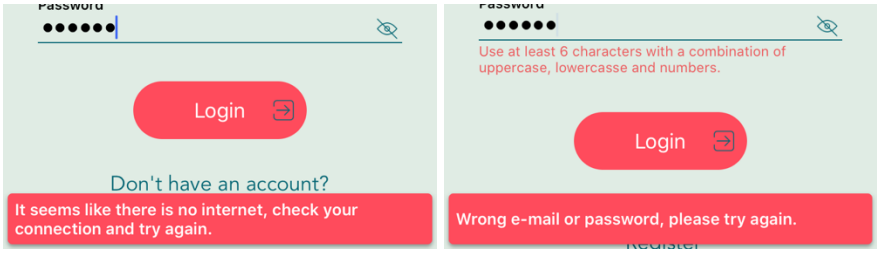
Slika 6.1. *Prikaz onboarding procesa*

Nakon uvoda u aplikaciju korisnik se susreće sa zaslonom za autentificiranje korisnika gdje se može prijaviti sa svojim računom ili može kreirati novi račun ukoliko je to potrebno (Slika 6.2). Ovi zasloni koriste prethodno opisanu uslugu Firebase Auth za rukovanje računima.



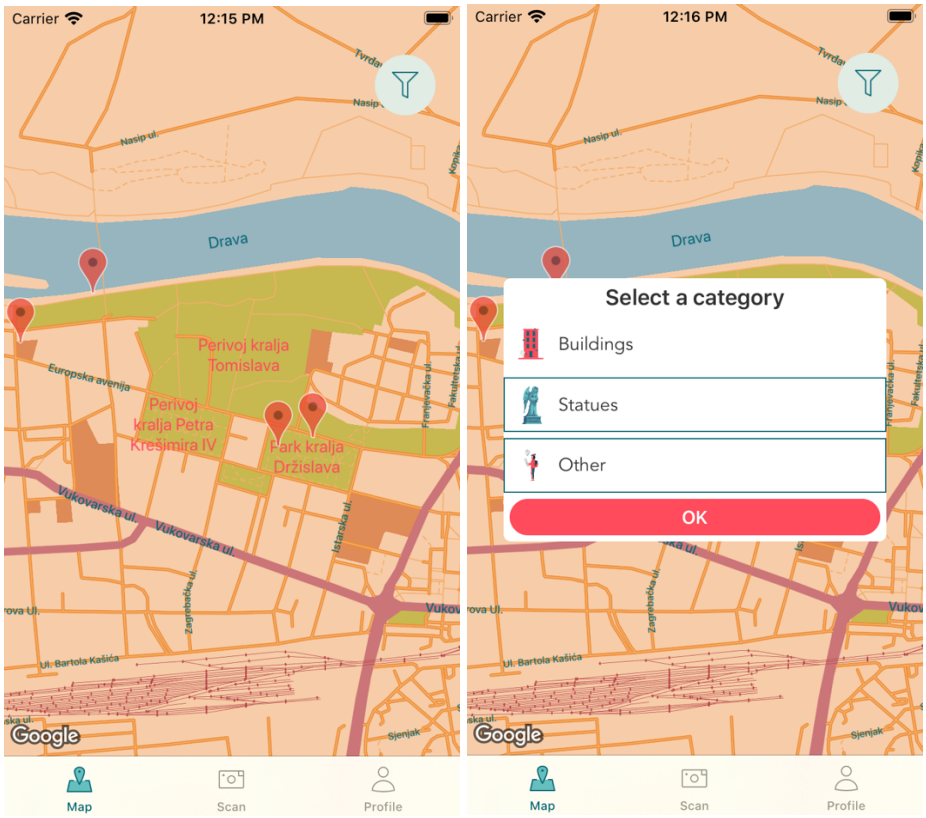
Slika 6.2. *Zaslon za prijavu i kreiranje novog računa*

Ukoliko tijekom prijave ili registracije korisnika dođe do problema korisnik se obavještava o problemu putem kratkih *snackbar* poruka, a primjer obavijesti prikazan je u nastavku (Slika 6.3).



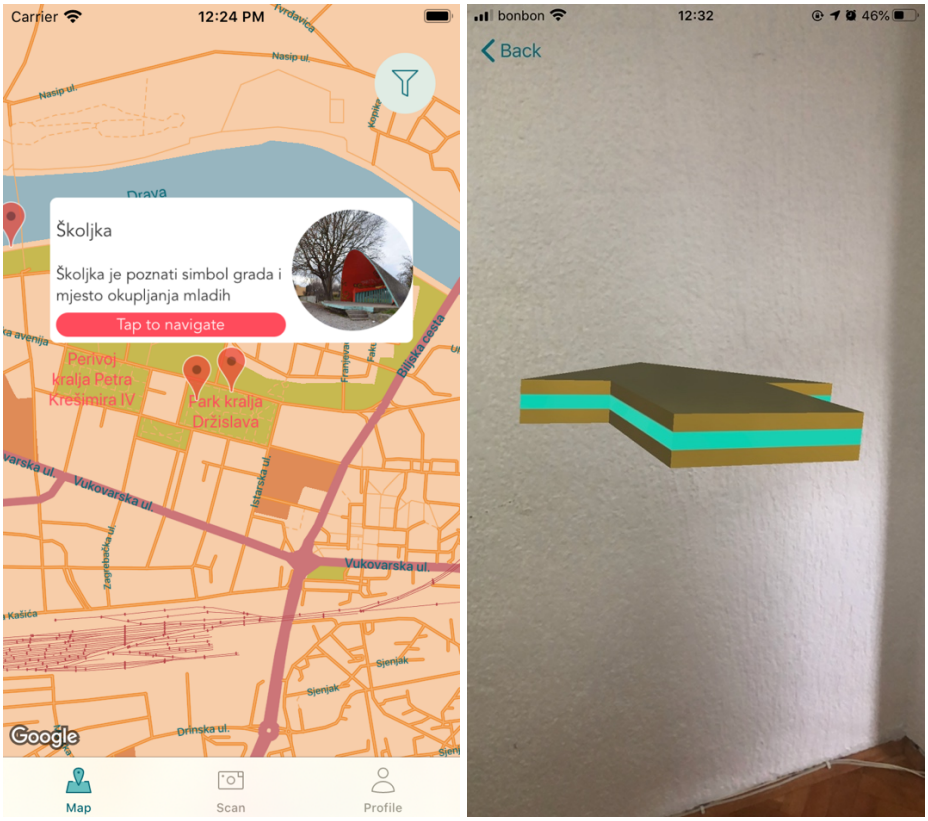
Slika 6.3. *Primjer prikaza pogreške putem snackbar-a*

Kada se korisnik uspješno prijavi u aplikaciju dočekuje ga glavni zaslon na kojemu se nalazi mapa sa svim lokacijama znamenitosti i zanimljivosti. Također, u gornjem desnom kutu mape nalazi se i filter kojim korisnik može filtrirati tip znamenitosti ili zanimljivosti koje ga zanima (Slika 6.4).



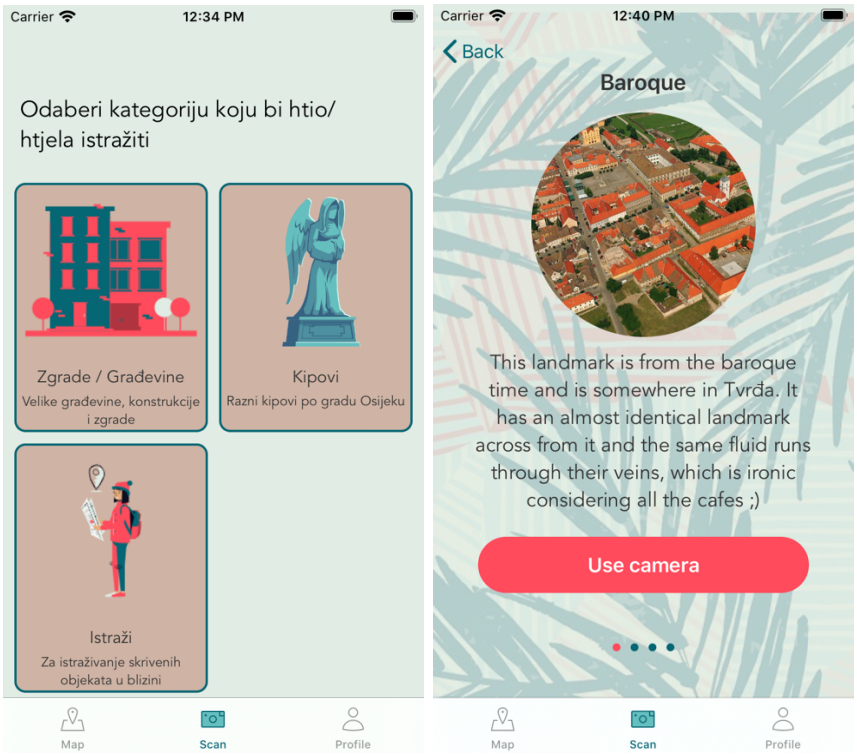
Slika 6.4. *Zaslon s mapom i mogućnosti filtriranja*

Pritiskom na neku od lokacija korisniku se otvara informacijski prozorčić s kratkim opisom lokacije te ukoliko korisnik želi može odabrati interaktivnu navigaciju do željenog mjesta. Upravo zbog takve vrste navigacije na mapi nije omogućeno gledanje vlastitog položaja kako bi se korisnici potakli na istraživanje grada. Primjer informacijskog prozorčića i navigacije prikazan je na slici 6.5.



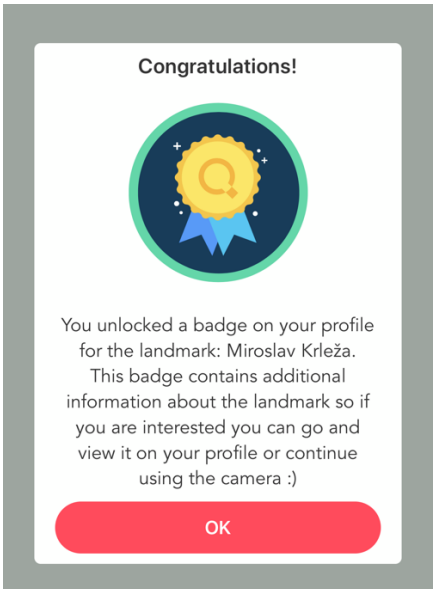
Slika 6.5. *Primjer informacijskog prozora i interaktivne navigacije*

Druga glavna kartica naziva se *scan* i predstavlja zaslon na kojem korisnik može odabrati tip znamenitosti ili zanimljivosti koju misli da je pronašao. Znamenitosti su podijeljene u tri glavne kategorije: zgrade i građevine, kipovi te kategorija istraži koja korisniku pruža nekoliko zagonetnih opisa na temelju kojih mora otkriti traženu znamenitost ili zanimljivost (Slika 6.6).



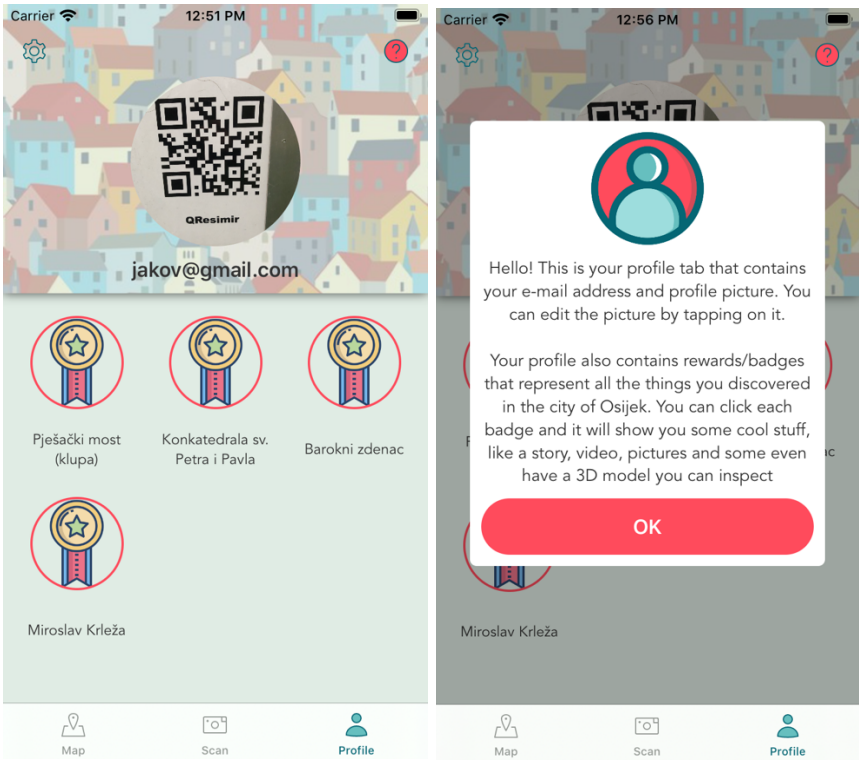
Slika 6.6. *Zaslon s tipovima znamenitosti i primjer zaslona istraži*

Prilikom odabira kategorije znamenitosti otvara se kamera uređaja te korisnik može uslikati znamenitost koju je odlučio istražiti. Kada korisnik potvrdi da želi koristiti sliku pokreće se proces klasifikacije te ukoliko je uslikan ispravni objekt, odnosno ukoliko je objekt uspješno klasificiran prikazuje mu se prozor s kratkom animacijom i potvrdom o uslikanoj znamenitosti (Slika 6.7).



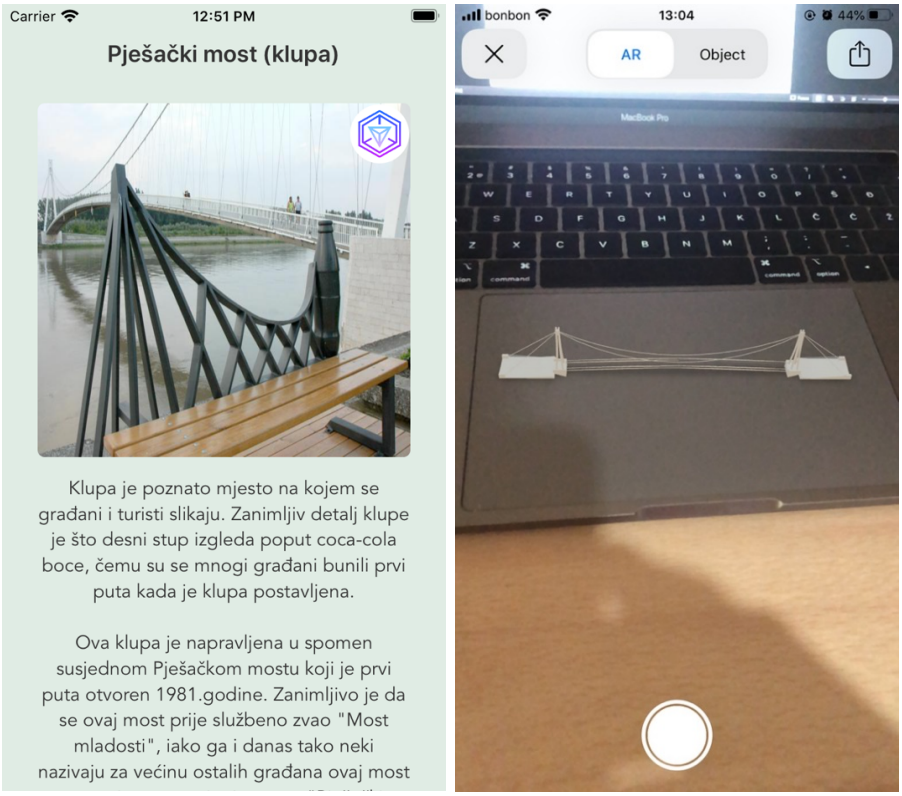
Slika 6.7. *Primjer prozora o uspješnom prepoznavanju objekta*

Zadnja kartica je profil korisnika koja sadrži korisnikovu profilnu sliku, email te sve njegove do sada otključane znamenitosti. Korisnik također ima opciju prikazivanja kratkog informacijskog prozora klikom na gumb u gornjem desnom kutu te dodatne opcije poput odjavljivanja s trenutnog računa i ponovnog prikaza *onboarding* procesa klikom na gumb u lijevom kutu (Slika 6.8).



Slika 6.8. *Primjer profila i informacijskog prozora na profilu*

Najvažniji dio ovog zaslona predstavljaju upravo otključane znamenitosti na profilu koje sadrže dodatne opise znamenitosti ili zanimljivosti, ali i zabavne medije poput slika, videa i 3D modela koji se mogu pregledavati u proširenoj stvarnosti ili brzim pregledom modela kako je prikazano na slici 6.9.



Slika 6.9. *Prikaz dodatnih informacija otključanih znamenitosti i 3D modela*

6.1. Testovi jedinica koda

Aplikacija je rađena prateći MVVM-C arhitekturni obrazac koji omogućuje lako testiranje poslovne logike, ali i elemenata grafičkog sučelja. Za svrhe testiranja koristi se Apple-ov XCTest programski okvir koji omogućuje lako pisanje i integriranje jediničnih testova unutar aplikacije [35].

Obavljeni su testovi *viewmodela* jer oni sadržavaju većinu poslovne logike te kritičnog programskog koda za ispravan rad aplikacije. Kako bi se mogle testirati sve metode potrebno je kreirati makete (engl. *mockups*) određenih podataka i usluga nad kojima će se testirati ispravnost pojedinih metoda. Primjer kreiranja makete usluge za bazu podataka prikazan je na slici 6.10.

```
class DatabaseServiceMock: DatabaseServiceProtocol {

    let shouldFail: Bool
    let userData = UserData(email: "", userID: "", badges: [], profilePicture: UIImage())

    init(shouldFail: Bool) {
        self.shouldFail = shouldFail
    }

    func createUser(email: String, userID: String, completion: @escaping (FirestoreResult<Any?>) -> Void) {
        if shouldFail {
            completion(.failure(.noInternet))
        } else {
            completion(.success("success"))
        }
    }

    func uploadImage(image: UIImage, userID: String, completion: @escaping (FirestoreResult<URL>) -> Void) {
        if shouldFail {
            completion(.failure(.noInternet))
        } else {
            completion(.success(URL(string: "www.youtube.com")!))
        }
    }
```

Slika 6.10. *Prikaz makete usluge za bazu podataka (DatabaseServiceMock)*

Na slici 6.10 je vidljivo da maketa usluge nasljeđuje isti protokol kao i stvarna klasa usluge (DatabaseServiceProtocol), ali je posebnost što ima parametar *shouldFail* koji se postavlja tijekom inicijalizacije usluge. Prethodno spomenutim parametrom se lako može odrediti željeni ishod svih zahtjeva usluge te se tako mogu testirati ponašanja *viewmodel*-a tijekom uspješnih i neuspješnih poziva. Na slici 6.11. prikazana je inicijalizacija *viewmodel-a* sa potrebnim maketama usluga.

```
override func setUp() {
    super.setUp()
    let databaseService = DatabaseServiceMock(shouldFail: false)
    let persistanceService = PersistanceServiceMock()
    viewModel = ProfileViewModel(databaseService: databaseService, persistenceService: persistanceService)
    viewModel.badges = [ClassificationData(rawValue: "rat")!]
}

func testGetUser() {
    let expect = expectation(description: "got user")
    viewModel.onGotUser = { data in
        expect.fulfill()
    }

    viewModel.getUser()
    wait(for: [expect], timeout: 0.2)
}
```

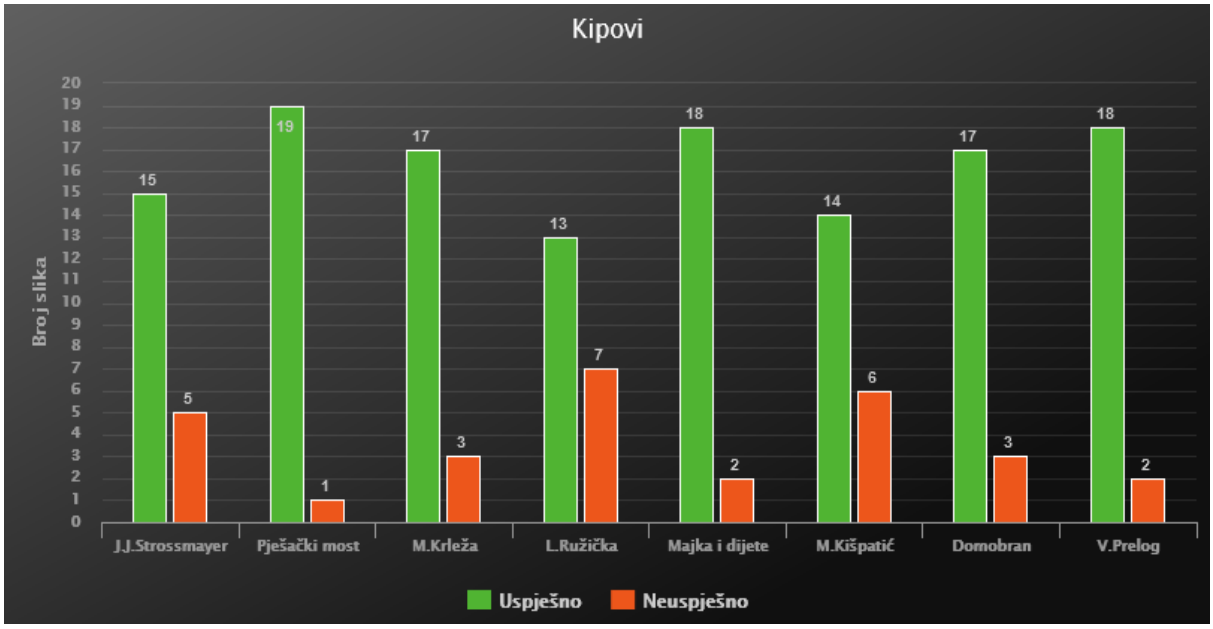
Slika 6.11. *Primjer testiranja viewmodel-a s uspješnim rezultatima zahtjeva*

Slika 6.11 pokazuje inicijalizaciju *ProfileViewModel*-a kojemu se predaju makete usluga *DatabaseService* te *PersistanceService*, a također je vidljivo da se usluzi za bazu podataka zahtjevi postavljaju kao uspješni. Prvi test je *testGetUser*() u kojem se provjerava hoće li metoda *viewmodel*-a *getUser*() koja koristi uslugu *databaseService* uspješno ispuniti očekivanja u

vremenskom roku od 0.2 sekunde. Vremensko očekivanje je postavljeno na malu vrijednost jer se ne radi o stvarnoj usluzi koja šalje pravi zahtjev, već o maketi usluge čiji se asinkroni pozivi odvijaju vrlo brzo. Svi testovi aplikacije su bili uspješni.

6.2. Analiza klasifikacijskih modela

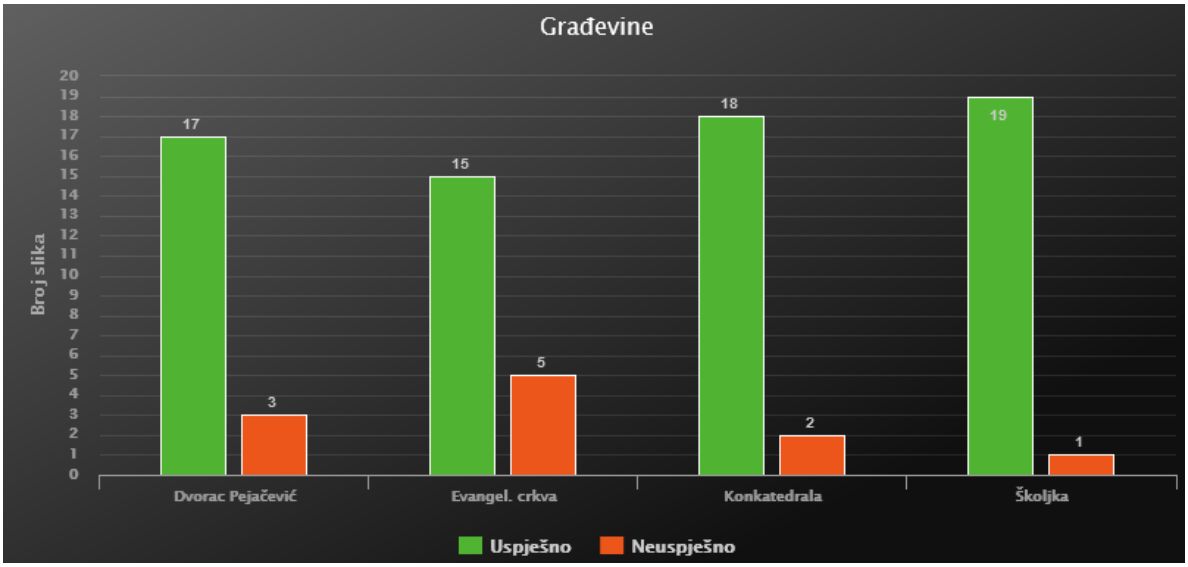
Kako bi se utvrdila ispravnost i uporabljivost klasifikacijskih modela testirane su sve klase iz sva tri klasifikacijska modela (građevine, kipovi i istraži). Izrađeni su grafovi uspješnosti prepoznavanja po klasama koji pokazuju broj uspješnih prepoznavanja uslikane znamenitosti / zanimljivosti. Prilikom testiranja korištene granice uvjerenosti za ispravnu klasifikaciju iznose 85% za kipove te 90% za građevine i kategoriju istraži, a broj testiranih slika za svaku klasu iznosi 20. Na slici 6.12 prikazani su rezultati testiranja klasifikacijskog modela „Kipovi“.



Slika 6.12. *Graf uspješnosti klasificiranja klasa iz kategorije Kipovi*

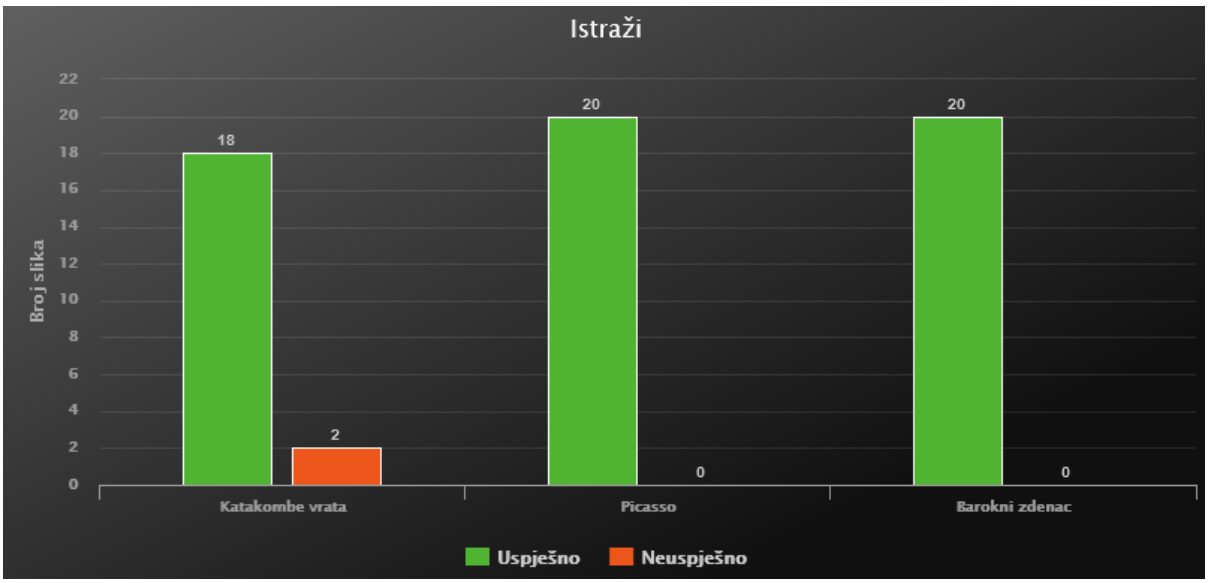
Na slici 6.12 vidljivi su rezultati uspješnih i neuspješnih klasificiranja osam različitih klasa (kipova). Iz rezultata se može zaključiti kako većina klasa ima postotak uspješnog prepoznavanja preko 85%, izuzevši kipove J.J.Strossmayera, Lavoslava Ružičke te Mije Kišpatića koji su smješteni na rondelu velikana u osječkoj Tvrđi. Razlog nižeg postotka uspješnosti je sličnost između kipova / klasa koji se nalaze unutar iste kategorije, odnosno kipovi jedni od drugih ne odstupaju u velikom broju značajki. Kako bi se povećala stopa uspješnosti prepoznavanja moguće je spustiti zadanu granicu uvjerenosti (sigurnosti), čime bi se utjecalo i na prepoznavanje ostalih

klasa unutar modela ili ponovno treniranje modela s više podataka za treniranje. Na isti način izrađen je graf uspješnosti klasifikacijskog modela „Građevine“ prikazan na slici 6.13.



Slika 6.13. *Graf uspješnosti klasificiranja klasa iz kategorije Građevine*

Slika 6.13. prikazuje graf uspješnosti klasificiranja građevina te je vidljivo kako najveći postotak uspješnosti iznosi 95% dok je najniži postotak 75% za evangelističku crkvu. Može se zaključiti da je klasifikacijski model građevina nešto bolji od modela kipova, a razlog tomu je veća raznolikost i manji broj klasa. Odstupanje klase „evangelistička crkva“ može se pridonijeti sličnosti s konkatedralom i nedovoljnom broju podataka za treniranje. Rezultati uspješnosti posljednje klasifikacijske kategorije „Istraži“ prikazani su na slici 6.14.



Slika 6.14. *Graf uspješnosti klasificiranja klasa iz kategorije Istraži*

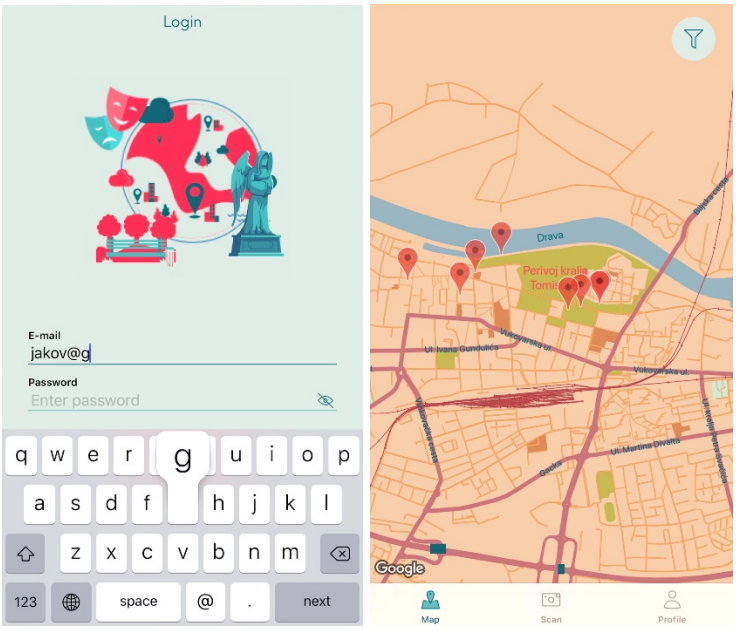
Prema rezultatima sa slike 6.14 vidljivo je da ovaj model ima najveću stopu uspješnosti klasificiranja klasa. Razlog tomu je međusobna različitost klasa, ali i broj kategorija koji se nalaze u modelu. Naime sve tri klase (katakombe, Picasso, barokni zdenac) uvelike se razlikuju jedne od druge izgledom te stoga klasifikacijski model može vrlo lako prepoznati o kojoj znamenitosti tj. zanimljivosti se radi.

Svi podatci ukazuju na zadovoljavajuću istreniranost klasifikacijskih modela, ali postoje određena odstupanja zbog međusobne sličnosti klasa te moguće nedovoljne istreniranosti podataka. Kako bi se podigla uspješnost prepoznavanja moguće je smanjiti predodređenu granicu uvjerenosti (sigurnosti) klasifikatora čime bi se povećao postotak lažno uspješnih klasificiranja ili je moguće povećati broj podataka za treniranje u klasama koje su izgledom slične (npr. kipovi s rondela velikana).

6.3. Ručno testiranje mobilne aplikacije

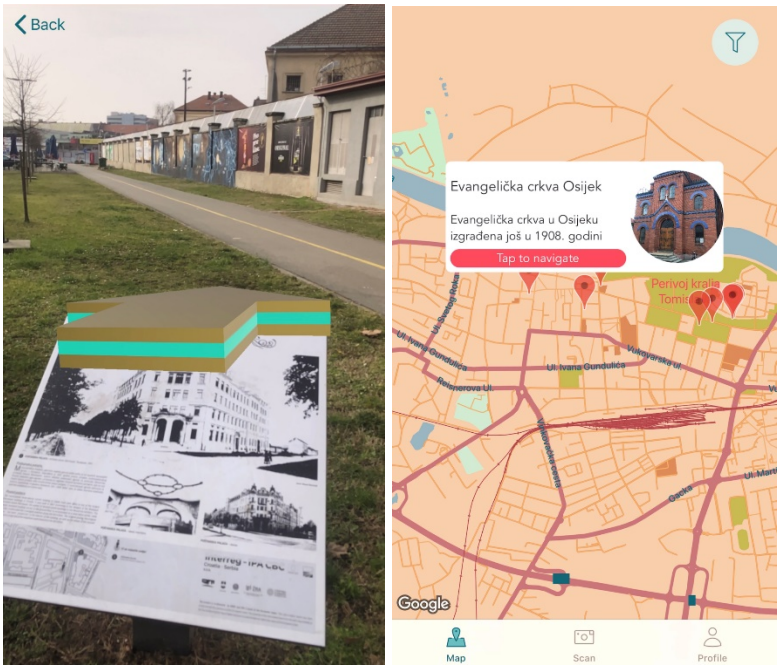
U ovom poglavlju odrađeno je ručno testiranje aplikacije kako bi se utvrdila uporabljivost aplikacije te kako bi se ukazali problemi tijekom korisničkog iskustva. Aplikacija je korištena na proizvoljno odabranoj ruti u kojoj se testira navigacija, prepoznavanje znamenitosti te pregled dodatnih informacija otkrivenih znamenitosti. Cijeli tok testiranja (korištenja) aplikacije opisan je redom te su navedeni mogući problemi tijekom korištenja mobilne aplikacije.

Testiranje je započeto na lokaciji osječke glavne pošte u centru grada, pokretanjem aplikacije prvobitno je obavljeno autentificiranje korisnika nakon čega se otvara karta sa lokacijama zanimljivosti (Slika 6.15).



Slika 6.15. *Autentificiranje korisnika (lijevo), karta s lokacijama zanimljivosti (desno)*

Na karti je odabrana lokacija evangeličke crkve, prikazuje se informacijski prozor s kratkim informacijama i slikom građevine te se pokreće navigacija pritiskom na tipku *Tap to navigate* (Slika 6.16).



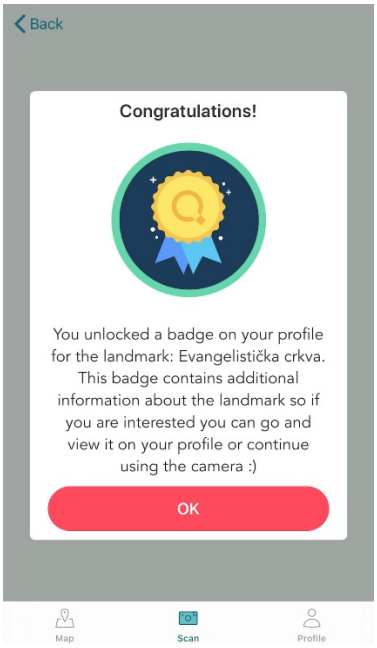
Slika 6.16. *Informacijski prozor s kratkim opisom odabrane lokacije (lijevo) i pokretanje navigacije prema evangeličkoj crkvi (desno)*

Praćenje navigacijske strjelice povremeno je otežano jer ostale građevine blokiraju korisniku pregled pa u trenutcima izgleda kao da strjelica pokazuje u krivu građevinu, ali nastavkom kretanja

postaje jasno gdje je prava lokacija znamenitosti. Takva navigacija možda nije idealna za korisničko iskustvo, ali se time korisnika potiče na istraživanje grada što je jedan od ciljeva aplikacije. Dolaskom na blizinu od 50 metara pojavljuje se obavijest i lokacijska oznaka koja obavještava korisnika o blizini znamenitosti (Slika 6.17).

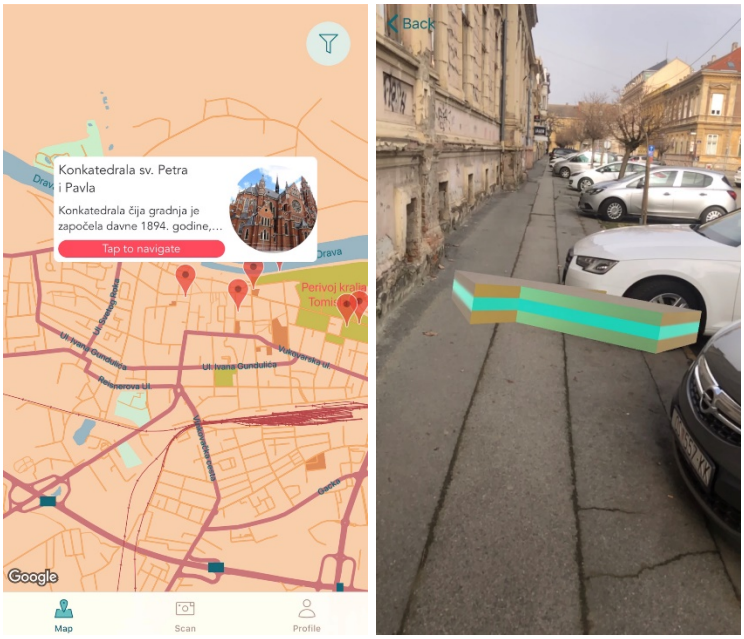
Slika 6.17. *Prikaz obavijesti i lokacijske oznake*

Uočeno je kako se lokacijska oznaka pojavljuje uranjeno u postupku navigacije što zbunjuje korisnika, ali udaljavanjem od znamenitosti ponovno se pojavljuje navigacijska strjelica čime se korisnika vraća na ispravni put. Za poboljšavanje korisničkog iskustva potrebno je svakako smanjiti radijus u kojem se pokazuje lokacijska oznaka i obavijest sa 50 metara na 20 ili 10 metara. Nakon pronalaženja evangeličke crkve izlazi se iz navigacije te se odabire opcija *scan* za slikanje i prepoznavanje znamenitosti. Znamenitost je uspješno prepoznata te se korisniku prikazuje prozor s informacijom o uspješnom prepoznavanju evangeličke crkve (Slika 6.18).



Slika 6.18. *Informacijski prozor o uspješnom prepoznavanju evangeličke crkve*

Od lokacije evangeličke crkve ponovno je otvorena karta sa znamenitostima, odabrana je lokacija osječke konkatedrale te je ponovno pokrenuta navigacija (Slika 6.19).



Slika 6.19. *Odabir konkatedrale na karti (lijevo) i pokretanje navigacije (desno)*

Približavanjem konkatedrali ponovno se pojavljuje lokacijska oznaka, ali u ovom slučaju je puno jasnije gdje se nalazi građevina, jer je prostor otvoreniji što je prikazano na slici 6.20.

Slika 6.20. *Prikaz lokacijske oznake za konkatedralu*

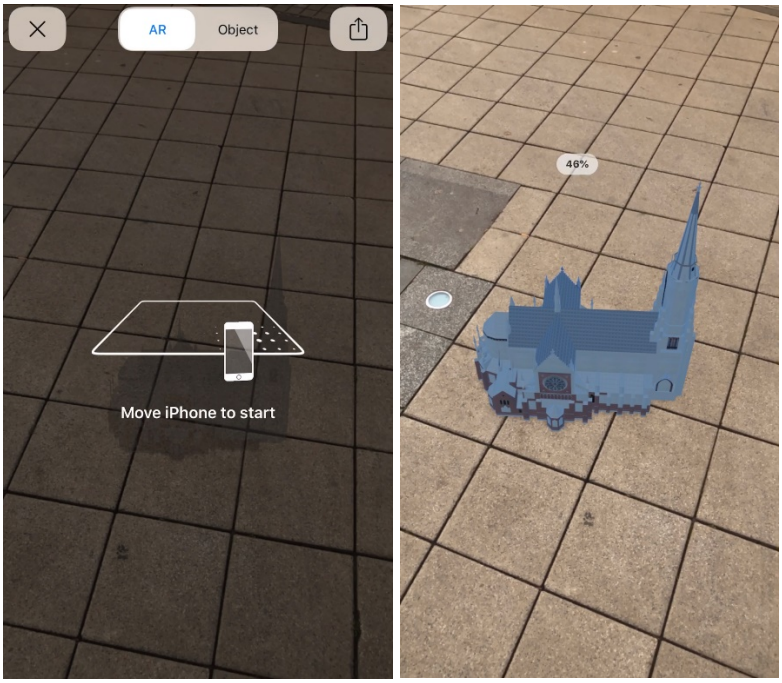
Nakon lociranja konkatedrale odrađeno je prepoznavanje građevine istim postupkom opisanim za evangeličku crkvu. Prepoznavanjem građevina korisnik svaki puta otključava nagradu na profilu u obliku značke kako je prikazano na slici 6.21.

Slika 6.21. *Otključane nagrade na korisnikovom profilu*

Pritiskom na otključanu značku otvara se prozor s detaljnim opisom znamenitosti i dodatnim medijskim sadržajem, a prikaz detalja značke za evangeličku crkvu i konkatedralu prikazan je na slici 6.22.

Slika 6.22. *Detalji značke za evangeličku crkvu (lijevo) i konkatedralu (desno)*

Konkatedrala na detaljima značke sadrži dodatnu tipku za pregled 3D modela koja otvara kameru uređaja i navodi korisnika na pronalaženje horizontalne podloge, a zatim postavlja model u stvarni svijet na pronađenu podlogu (Slika 6.23).



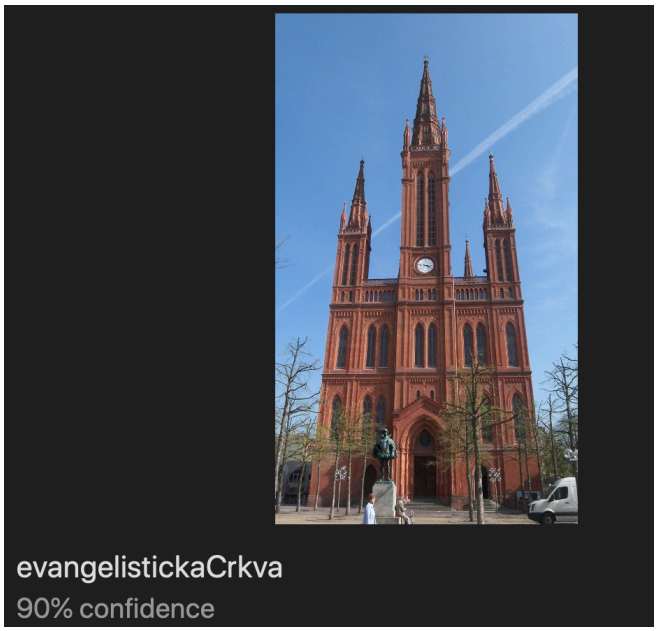
Slika 6.23. *Navođenje korisnika na horizontalnu podlogu (lijevo) i smještanje 3D modela u stvarni svijet (desno)*

Ručnim testiranjem potvrđeno je da aplikacija obavlja sve definirane funkcionalnosti iz idejnog rješenja, ali u određenim postupcima poput navigacije ima dodatnog prostora za poboljšavanje korisničkog iskustva.

6.4. Nedostatci i problemi mobilne aplikacije

Iako je aplikacija u potpunosti funkcionalna u njoj su uočeni određeni nedostatci i problemi koji su najčešće vezani uz kameru uređaja i vanjske uvjete u kojima se kamera koristi. Naime, najočitiji problem je što se aplikacija za ispravno funkcioniranje proširene stvarnosti i prepoznavanja slika mora koristiti u dovoljnim svjetlosnim uvjetima. Apple za pravilno prepoznavanje 3D objekata preporučuje minimalno osvjetljenje od 250 do 400 luksa (lm/m^2) te izbjegavanje bilo kakvih obojenih svjetlosnih izvora [36].

Modeli za prepoznavanje slika kreirani uporabom alata CreateML pokazali su veliku preciznost kada je riječ o očekivanom objektu, ali također pokazalo se da ponekad dolazi do lažno pozitivnih rezultata. Većina tih rezultata filtrirana je postavljanjem granice sigurnosti koja je otprilike 85% za većinu istreniranih modela, no kod kompleksnijih građevina poput osječke evangelističke crkve i dalje dolazi do pogrešne klasifikacije kako je prikazano na slici 6.24.



Slika 6.24. *Primjer pogrešne klasifikacije*

Gornji primjer prikazuje pogrešnu klasifikaciju iako model govori da ima pouzdanost od 90%, a razlog je kombinacija kompleksne strukture te manjka kvalitetnih slika za treniranje modela. Kod

manje kompleksnih struktura poput kipova ne dolazi do ovakvih pogrešaka, već su klasifikacije većinski točne i nema lažno pozitivnih rezultata.

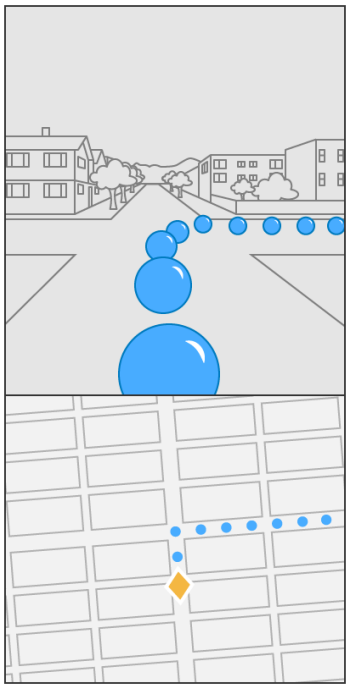
6.5. Moguća poboljšanja mobilne aplikacije

Svakodnevnim razvijanjem tehnologije i novih programskih okvira, određena rješenja unutar aplikacije se mogu poboljšati radi ugodnijeg korisničkog iskustva i bolje općenite funkcionalnosti aplikacije. U ovom poglavlju predloženo je nekoliko poboljšanja koja bi olakšala ili pružila zadovoljnije iskustvo korisnicima.

ARKit navigacija

Izlaskom sustava iOS 14.0 krajem 2020. godine Apple je najavio dolazak novih mobilnih uređaja s boljim sklopovljem, a time je također omogućeno korištenje nekih do sada nedostupnih mogućnosti u postojećim programskim okvirima. Tako je u programskom okviru ARKit uvedena nova vrsta konfiguracije AR scene, a naziva se *ARGeoTrackingConfiguration* te putem GPS-a, kamere i kompasa omogućuje ograničavanje iskustava proširene stvarnosti na željena geografska područja. Tako se korištenjem ARKit programskog okvira mogu zadati nekoliko geografskih koordinata koje se generiraju u obliku lokacijskih sidra (engl. *location anchors*) na koje se aplikacija može pozvati kada je u neposrednoj blizini [37].

Na prethodno opisan način bi se mogla izraditi zanimljivija, interaktivnija te intuitivnija navigacija za korisnika. Umjesto 3D strjelice koja se rotira na mjestu mogli bi se postaviti razni virtualni objekti diljem puta kojim se korisnik kreće (Slika 6.25).



Slika 6.25. *Primjer navigacije putem ARGeoTrackingConfiguration* [37]

Također, noviji iPhone uređaji imaju LiDAR kamere koje kreiraju puno detaljniju mrežu referentnih točaka prostora i objekata, čak i pri slabijim svjetlosnim uvjetima. Korištenjem nekih od tih uređaja mogli bi se stvoriti AR referentni objekti za kompleksnije i veće strukture poput kipova te bi se na taj način omogućilo korištenje tih struktura u proširenoj stvarnosti. Osim generiranja vlastitih referentnih točaka, LiDAR uređaji puno brže i preciznije smještaju virtualne objekte u stvarni svijet.

CoreML modeli

U prethodnom poglavlju su već spomenuti problemi s prepoznavanjem kompleksnijih struktura pa bi se za bolje korisničko iskustvo mogao napraviti model s više podataka za treniranje. Osim toga model bi se mogao izraditi koristeći neki drugi alat umjesto CreateML-a kako bi se mogle koristiti drugačije tehnike i algoritmi treniranja.

Također, postoji mogućnost implementiranja usluge u kojoj bi korisnici mogli slati svoje slike koje su uzrokovale krive klasifikacijske rezultate. Na taj način bi se model mogao konstantno širiti i poboljšavati i nadopunjavati sa što više različitih podataka od strane korisnika. Drugi način za poboljšavanje uspješnosti klasifikacijskog modela je podjela znamenitosti u manje klasifikacijske grupe sa što raznovrsnijim objektima kako objekti ne bi sadržavali vizualno slične interesne / referentne točke.

Baza podataka

Trenutna baza podataka je sasvim dovoljna za svrhe ove aplikacije jer se većinski spremaju kratki s*tring*-ovi, ali je problem spremanje korisnikovih slika. Umjesto korištenja kombinacije Firebase Storage i Firestore usluge, baza podataka bi se trebala ujediniti korištenjem jedne usluge koja je sposobna za spremanje podataka većih od 1 MB. Osim promjene usluge za bazu podataka, alternativno rješenje bilo bi smanjivanje kvalitete slika koje korisnici mogu spremati te zatim spremanje tih podataka u Firestore bazu. Time bi se smanjila vremenska trajanja zahtjeva koja bi poboljšala iskustvo korisnicima s lošijim brzinama preuzimanja i slanja podataka.

Također, u bazu podataka bi se mogli učitati svi dostupni modeli proširene stvarnosti kako ne bi zauzimali memorijski prostor na mobilnom uređaju korisnika, a nakon što korisnik otkrije određenu zanimljivost automatski bi otključao poveznicu na odredište pripadajućeg modela proširene stvarnosti. Jedini nedostatak ovakvog pristupa je nemogućnost pregledavanja modela proširene stvarnosti prilikom manjka internetske veze.

7. ZAKLJUČAK

Tehnologija proširene stvarnosti u zadnjih nekoliko godina eksponencijalno je napredovala, pa su danas moguće razne simulacije proširene stvarnosti koje su u prošlosti zahtijevale iznimno složene strojeve i sustave. Danas se te simulacije mogu obavljati putem običnih pametnih telefona. U ovom diplomskom radu izrađena je i ispitana mobilna iOS aplikacija koja koristi tehnologiju proširene stvarnosti za poticanje ljudi na istraživanje znamenitosti i zanimljivosti grada Osijeka. Aplikacija je ostvarena uporabom programskog jezika Swift i razvojnog okruženja XCode, baza podataka i autentifikacijske usluge ostvarene su platformom Firebase, a za potrebe simuliranja proširene stvarnosti korišten je programski okvir ARKit. Osim proširene stvarnosti aplikacija koristi i strojno učenje za olakšano prepoznavanje znamenitosti, a modeli strojnog učenja izrađeni su pomoću alata CreateML. Programsko rješenje aplikacije, njena arhitektura kao i potrebni alati detaljno su opisani, a funkcionalnost aplikacije testirana je pomoću jediničnih testova i ručnog testiranja. Osim funkcionalnosti aplikacije testirani su i izrađeni modeli strojnog učenja.

Analiziranjem rezultata testiranja zaključeno je da je aplikacija funkcionalna, ali postoji puno prostora za poboljšanjem mobilne aplikacije, ali i korisničkog iskustva. Pokazano je da iako su klasifikacijski modeli precizni moguće je poboljšanje njihove točnosti putem dodatnog treniranja modela kako bi se smanjila količina lažno pozitivnih rezultata. Korištenjem aplikacije također je pokazano da se navigacijski sustav može poboljšati korištenjem novijih tehnologija i programskih okvira proširene stvarnosti u svrhe poboljšavanja korisničkog iskustva. Osim nedostataka, može se zaključiti da još uvijek postoji opći nedostatak tehnologija proširene stvarnosti, jer unatoč tome što simuliranje proširene stvarnosti radi u idealnim uvjetima, pokazalo se kako simulacije nisu uvijek precizne i stabilne. Korištenjem novije tehnologije s naprednijim sklopovljem te uporabom posljednjih programskih alata i okvira, aplikacija AROsijek može se dodatno usavršiti i optimirati.

LITERATURA

[1] Popsci.com, *„Microsoft's Augmented Reality HoloLens Will Be Available In Early 2016“,* https://www.popsci.com/microsoft-hololens-augmented-reality-coming-early-2016-3000-dollars/ , pristup ostvaren 19.06.2020.

[2] C. Language, N. Bandekar, A. Bello and T. Coron, *„ARKit by Tutorials, building Augmented Reality Apps in Swift“, Razeware LLC., 2019.*

[3] Microsoft Dynamics, „*Dynamics 365 Guides“*, https://dynamics.microsoft.com/en-gb/mixed-reality/guides/, pristup ostvaren 19.06.2020.

[4] N. Statt, „*Google opens its latest Google Glass AR headset for direct purchase“*, The Verge, https://www.theverge.com/2020/2/4/21121472/google-glass-2-enterprise-edition-for-sale-directly-online , pristup ostvaren 19.06.2020.

[5] Glass, Google, https://www.google.com/glass/tech-specs/, pristup ostvaren 19.06.2020.

[6] S. Khillar, *„Difference between Google Glass and Microsoft HoloLens“,* http://www.differencebetween.net/technology/difference-between-google-glass-and-microsoft-hololens/, pristup ostvaren 19.06.2020.

[7] IQUII, „*From Augmented Reality to Mobile Reality: it's the smartphone that drives augmented reality“,* https://medium.com/iquii/from-augmented-reality-to-mobile-reality-its-the-smartphone-that-drives-augmented-reality-31ae65b44958, pristup ostvaren 21.06.2020.

[8] DJ, appypie.com, *„5 Great Reasons Why Augmented Reality is the Future of Mobile Apps“*, https://www.appypie.com/augmented-reality-future-of-mobile-apps, pristup ostvaren 21.06.2020.

[9] Apple Developers, „*Occluding Virtual Content with People“*, https://developer.apple.com/documentation/arkit/occluding_virtual_content_with_people, pristup ostvaren 21.06.2020.

[10] Google Developers, „*ARCore overview“,* https://developers.google.com/ar/discover, pristup ostvaren 21.06.2020.

[11] Fyresite, dev.to, „*OS And Android AR Development: Which OS Wins In 2020?“,* https://dev.to/fyresite/os-and-android-ar-development-which-os-wins-in-2020-1ee8, pristup ostvaren 21.06.2020.

[12] Google AR & VR, Google, https://arvr.google.com/arcore/, pristup ostvaren 21.06.2020.

[13] T. Tall, *„Augmented Reality vs. Virtual Reality vs. Mixed Reality – An Introductory Guide“,* https://www.toptal.com/designers/ui/augmented-reality-vs-virtual-reality-vs-mixed-reality, pristup ostvaren 21.06.2020.

[14] HistoryOfInformation.com, „*Louis Rosenberg Develops Virtual Fixtures, the First Fully Immersive Augmented Reality System“, https://www.historyofinformation.com/detail.php?id=4231* , pristup ostvaren 21.06.2020.

[15] K. Dupzyk „I saw the future through Microsoft's HoloLens“, http://www.popularmechanics.com/technology/a22384/hololens-ar-breakthrough-awards/ , pristup ostvaren 21.06.2020.

[16] Assemblr Team, *„3 Different Types of Augmented Reality“,* https://blog.assemblrworld.com/3-different-types-of-marker/, pristup ostvaren 21.06.2020.

[17] J. Tißler, „*Augmented Reality in Marketing: 8 Current Examples*“, https://dmexco.com/stories/augmented-reality-in-marketing-8-current-examples-2/, pristup ostvaren 21.06.2020.

[18] technologymagazines.net, „*Augmented Reality vs Virtual Reality Which Is Better“,* https://technologymagazines.net/augmented-reality-vs-virtual-reality-which-is-better/, pristup ostvaren 21.06.2020.

[19] M. Sttiles, „Reality - virtuality continuum“, https://medium.com/desn325-emergentdesign/reality-virtuality-continuum-868cb8121680 , pristup ostvaren 21.06.2020.

[20] X. Wei, D.Weng, Y. Liu, Y. Wang, „*A Tour Guiding System of Historical Relics Based on Augmented Reality*“, Beijing Engineering Research Center of Mixed Reality and Advanced Display, Beijing Institute of Technology, https://ieeexplore.ieee.org/document/7504776 , pristup ostvaren 25.10.2020.

[21] H. Jacobs, *„I traveled the world for 6 months, and here's the single best app I couldn't live without“,* Business Insider, https://www.businessinsider.com/best-app-for-traveling-the-world-google-translate-2018-10 , prisutp ostvaren 25.10.2020.

[22] J. Hong, „*Augmented Reality in Medicine*“, Department of Robotics Engineering, Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu, Južna Koreja, https://www.researchgate.net/publication/311467975_Augmented_Reality_in_Medicine , pristup ostvaren 20.11.2020.

[23] K. M. Kanno, E. A. Lamounier, A. Cardoso, E. J. Lopes, G. F. Mendes de Lima, *„Augmented Reality System for Aiding Mild Alzheimer Patients and Caregivers“,* Federal

University of Uberlândia, https://ieeexplore.ieee.org/document/8446143 , pristup ostvaren 20.11.2020.

[24] MIT News, *„What 126 studies say about education technology“,* https://news.mit.edu/2019/mit-jpal-what-126-studies-tell-us-about-education-technology-impact-0226 , pristup ostvaren 20.11.2020.

[25] C. S. C. Dalim, T. Piumsomboon, A. Dey, M. Billinghurst, S. Sunar, *„TeachAR: An Interactive Augmented Reality Tool for Teaching Basic English to Non-Native Children“,* Empathic Computing Lab, Faculty of Computer Science and Information Technology University of South Australia, University of Technology Malaysia, Tun Hussein Onn University of Malaysia, https://ieeexplore.ieee.org/document/7836534 , pristup ostvaren 21.11.2020.

[26] Apple, „*Welcome to XCode“,* https://help.apple.com/xcode/mac/current/#/devc8c2a6be1 , pristup ostvaren 21.11.2020.

[27] Swift.org, „*About Swift“,* https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html , pristup ostvaren 21.11.2020.

[28] Apple Developer, „*Create ML*“, https://developer.apple.com/documentation/createml , pristup ostvaren 21.11.2020.

[29] Apple Developer, *„Machine Learning“*, https://developer.apple.com/machine-learning/create-ml/ , pristup ostvaren 21.11.2020.

[30] Core ML Tools, https://coremltools.readme.io/docs/other-converters , pristup ostvaren 21.11.2020.

[31] M. Santa, *“MVVM-C with Swift“,* https://marcosantadev.com/mvvmc-with-swift/ , pristup ostvaren 21.11.2020.

[32] T. Kerpelman, „*Cloud Firestore vs the Realtime Database: Which one do I use?“,* Firebase, https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html , pristup ostvaren 23.11.2020.

[33] Apple Developer, *„Core Location“,* https://developer.apple.com/documentation/corelocation, pristup ostvaren 23.11.2020.

[34] Apple Developer, *„Core ML“,* https://developer.apple.com/documentation/coreml , pristup ostvaren 23.11.2020.

[35] Apple Developer, *„XCTest“*, https://developer.apple.com/documentation/xctest , pristup ostvaren 23.11.2020.

[36] Apple Developer, „*Scanning and detecting 3D objects*“, https://developer.apple.com/documentation/arkit/scanning_and_detecting_3d_objects , pristup ostvaren 23.11.2020.

[37] Apple Developer, *„Tracking geographic locations in AR*“, https://developer.apple.com/documentation/arkit/tracking_geographic_locations_in_ar , pristup ostvaren 23.11.2020.

[38] Apple Developer, „*Creating an Image Classifier Model*”, https://developer.apple.com/documentation/createml/creating_an_image_classifier_model, pristup ostvaren 28.01.2021.

[39] Apple Developer, *„Vision*“, https://developer.apple.com/documentation/vision, pristup ostvaren 28.01.2021.

[40] N. Gupton, *„What's the Difference Between AR, VR, and MR?“,* https://www.fi.edu/difference-between-ar-vr-and-mr, pristup ostvaren 28.01.2021.

SAŽETAK

MOBILNA iOS APLIKACIJA ZA ISTRAŽIVANJE ZNAMENITOSTI GRADA PODRŽANO PROŠIRENOM STVARNOŠĆU I POSTUPCIMA STROJNOG UČENJA

Cilj ovog diplomskog rada je izrada zabavne iOS aplikacije za istraživanje znamenitosti i zanimljivosti grada Osijeka putem tehnologije proširene stvarnosti, te korištenjem postupaka strojnog učenja. Aplikacija je ostvarena korištenjem programskog jezika Swift i razvojnog okruženja XCode, a za potrebe pozadinskog sloja i njegovih usluga korištena je platforma Firebase te njeni programski okviri koji omogućuju potrebne funkcionalnosti aplikacije poput autentificiranja i spremanja korisničkih podataka. Tehnologija proširene stvarnosti korištena je za prikazivanje virtualnih 3D modela te za interaktivnu navigaciju, dok je strojno učenje korišteno u svrhe prepoznavanja raznih objekata poput zgrada, građevina i kipova metodom klasifikacije slika. Aplikacija je napisana prema obrascu MVVM-C što omogućuje lako testiranje svih dijelova koda putem jediničnih testova i olakšanu kontrolu toka aplikacije. Testiranjem i uporabom aplikacije pokazano je da je aplikacija funkcionalna, ali postoji još puno prostora za poboljšanje zbog stalnog razvoja tehnologija, programskih alata i okvira.

Ključne riječi: iOS aplikacija, MVVM-C, navigacija, proširena stvarnost, strojno učenje, znamenitosti

ABSTRACT

MOBILE iOS APPLICATION FOR EXPLORING THE CITY AND SIGHTSEEING WITH THE HELP OF AUGMENTED REALITY AND MACHINE LEARNING

The goal of this masters thesis is to create a fun iOS application for exploring the sights and curiosities of the city of Osijek using augmented reality (AR) and machine learning procedures. The application was accomplished using the Swift programming language and the XCode development environment, while the backend requirements and its services have been achieved using the Firebase platform and its libraries, which enable required features like authentication and storing user information in the database. The augmented reality technology was used for presenting virtual 3D objects and for an interactive navigation system, while machine learning was used for recognizing objects such as buildings, structures and statues using image classification. The application was written following the MVVM-C architectural pattern which allows for easy testing of every part of the code and simpler application flow control. Testing and the usage of the app showed that it is functional, but there is a lot of room for improvement because of the continuous developments in technology, programming tools and frameworks.

Keywords: iOS application, MVVM-C, navigation, augmented reality, machine learning, sights

ŽIVOTOPIS

Jakov Jurić rođen je 12. svibnja 1996. godine u Osijeku. Nakon završene III. Gimnazije u Osijeku, 2015. godine ostvaruje direktan upis na preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija, tadašnji Elektrotehnički fakultet u Osijeku. 2018. godine stječe akademski naziv sveučilišni prvostupnik (lat. *baccalaureus*) inženjer računarstva. Iste godine upisuje diplomski sveučilišni studij programsko inženjerstvo.

Potpis:

_____

PRILOZI

- Prilog 1. – Diplomski rad u .pdf formatu
- Prilog 2. – Programski kod aplikacije
- Prilog 3. – Video snimak uporabe aplikacije