Aplikacija za kreiranje dodatnih zabavnih i edukativnih sadržaja

Kokot, Božidar

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologije

BOŽIDAR KOKOT

APLIKACIJA ZA KREIRANJE DODATNIH ZABAVNIH I EDUKATIVNIH SADRŽAJA U KULTURNOM TURIZMU

Diplomski rad

Pula, _____, ___ godine.

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologije

BOŽIDAR KOKOT

APLIKACIJA ZA KREIRANJE DODATNIH ZABAVNIH I EDUKATIVNIH SADRŽAJA U KULTURNOM TURIZMU.

Diplomski rad

JMBAG: 0303038246 redoviti student

Studijski smjer: Informatika

Predmet: Izrada informatičkih projekata

Znanstveno područje: Informatika
Znanstveno polje:
Znanstvena grana:

Mentor: doc.dr.sc. Siniša Sovilj
Sumentor: dipl.ing.rač Nikola Tanković

Pula, _____, ____ godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Božidar Kokot, kandidat za magistra Informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA
o korištenju autorskog djela

Ja, Božidar Kokot, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom Izrada edukativnog alata koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis
_____

SVEUČILIŠTE JURJA DOBRILE U PULI
ODJEL ZA INFORMACIJSKO-KOMUNIKACIJSKE TEHNOLOGIJE

Pula, 1. ožujka 2017.

DIPLOMSKI ZADATAK

Pristupnik:	Božidar Kokot (0303038246)
Studij:	Sveučilišni diplomski studij Informatike
Naslov (hrv.):	Aplikacija za kreiranje dodatnih zabavnih i edukativnih sadržaja u kulturnom turizmu.
Naslov (eng.):	Application for creating additional entertainment and educational content for cultural tourism.
Opis zadatka:	Zadatak je realizirati sustav namjenjen kulturnom turizmu, koji će omogućiti kreiranje i izmjenjivanje sadržaja (npr. fotografija kulturnih eksponata) na web administratorskom panelu, pozivanje vision api-a za obrađivanje slike te spremanje rezultata u bazu podataka. Također je potrebno realizirati edukativno-zabavnu Android aplikaciju zaduženu za prikaz pripadajućeg sadržaja krajnjim korisnicima (npr. posjetiteljima) te im omogućiti fotografiranje i uspoređivanje rezultata iz vision api-a nad uslikanom fotografijom na uređaju sa rezultatima spremljenim u bazi podataka. Potrebno je proučiti način rada Android operacijskog sustava te istražiti i osmisliti arhitekturu sustava za web i mobilnu aplikaciju.

Zadatak uručen pristupniku: 1. ožujka 2017.
Rok za predaju rada: 8. lipnja 2017.

Mentor:

_____
doc.dr.sc Siniša Sovilj

Sadržaj

Uvod 1
1. Organizacijska metodologija korištena za razvoj prototipa 2
1.1 Lean metodologija 2
1.2 Scrum 8
1.3 Sastavljanje i interpretiranje rezultata ankete 9
2. Tehnologije i obrasci korišteni u razvoju aplikacije 15
2.1 Operacijski sustav Android 15
2.2 Android studio 17
2.3 Javascript i MEAN stack arhitektura 24
2.4 Algoritam za prepoznavanje slike VISION API 25
3. Realizacija HistriApp aplikacije 29
3.1 Opis sustava HistriApp platforme 29
3.2 Razvoj Web sučelja 33
3.3 Backend 37
3.4 Izrada Android aplikacije 42
3.5 Budući razvoj 59
Zaključak 61
Literatura 62
Sažetak 65
Abstract 66

Uvod

Naslov ovog diplomskog rada glasi “Aplikacija za kreiranje dodatnih zabavnih i edukativnih sadržaja u kulturnom turizmu”. Cilj samog rada jest izrada platforme koju bi kulturne institucije bile spremne koristiti radi poboljšanja sadržaja aplikacije. Prije svega, bilo bi dobro napomenuti motivaciju za izradu samog diplomskog rada koja proizlazi iz uočenog raskoraka između sadržaja kojeg nude turističke rute kulturnog karaktera i kulturne institucije i zainteresiranosti mlađe populacije za sadržajem kojeg nude te iste ustanove. Aplikacija je napravljena u nadi da pridonese poboljšanju načina na koji turističke agencije i kulturne institucije prilaze svojim krajnjim korisnicima - posjetiteljima. Sam diplomski rad podijeljen je u tri glavne tematske cijeline.

U prvoj tematskoj cjelini samoj temi je pristupljeno kroz opis organizacijske metodologije koja se koristila za izradu prototipa aplikacije. U svakoj cjelini su objašnjeni osnovni pojmovi korištene organizacijske metodologije te prikazan primjer izrade ankete za ispitivanje zainteresiranosti tržišta, koja se koristila kako bi se istražilo jesu li korisnici bili zainteresirani za samu aplikaciju. Zatim su objašnjeni rezultati ankete te je sama aplikacija prilagođena odgovoru tržišta.

U drugoj tematskoj cjelini potanje su objašnjeni detalji o tehnologijama koje su korištene u izradi same aplikacije. Također su opisani sustavi na kojima se izvršava sama aplikacija.

U trećoj tematskoj cjelini opisan je proces izrade same aplikacije. Opisana je arhitektura samog sustava. Prikazani su diagrami korištenja određenih komponenti sustava. Također su predočeni dijelovi koda uz objašnjene pojedinih dijelova sustava.

1. Organizacijska metodologija korištena za razvoj prototipa

Prva cjelina ovog diplomskog rada obuhvaća opis metodologije razvoja nove aplikacije koja je korištena u izradi prototipne aplikacije. Pojašnjena je Lean metodologija te njezine sastavnice. U sklopu Lean metodologije prikazana je i izrada ankete za istraživanje inicijalne ideje aplikacije te njezino eventualno oblikovanje na temelju odgovora tržišta na ideju. Nakon dobivenog rezultata ankete, sama inicijalna ideja je preoblikavana kako bi odgovarala krajnjim korisnicima. Inicijalna ideja aplikacije jest platforma koja posjetiteljima kulturnih institucija ili turističkih ruta kulturnog karaktera pruža mogučnost pregledavanja sadržaja sudjelovanjem u takozvanoj „treasure hunt“ avantura. Koncept „treasure hunt“ avanture predstavlja vid zabave u kojemu korisnici istražuju lokalitet kroz interaktivne zagonetke koje moraju pronaći.

1.1 Lean metodologija

U današnjem svijetu u kojem je naglasak na razvijenom internetu, računarstvu u oblacima ili pak open-source softwaru, cijena izgradnje inovativnog proizvoda je niža no ikada. Statistika temeljena na podatcima nedavnog istraživanja U.S Bureau of Labor Statistics(2015) pokazuje da 79.9% kompanija koje počnu s razvijanjem novog proizvoda prežive prve dvije godine, ono što odvaja uspješne startupe od bezuspješnih ne mora nužno biti to da uspješni startupovi imaju svoj dobro izgrađen plan kojega se drže. U današnjoj ekonomiji startupa inicijalni planovi se mjenjaju dinamičnije no ikada. Upravo je tu svoju ulogu našla Lean metodologija, koja predstavlja sistemski način iteracije od inicijalnog plana do plana koji donosi najviše koristi, sve uz optimalno iskorištavanje svojih resursa.

Klasični pristup pogleda na razvoj proizvoda uključuje korisnika u razvoj svog proizvoda u samoj fazi prikupljanja zahtjeva, dok ih kasnije u samoj fazi izrade proizvoda isključuje te svu korisničku validaciju ostavlja za trenutak kad je sam softver već izdan. Tako da postoji vremenski period u kojemu se startupovi odvoje od svojih korisnika dok izgrađuju i testiraju svoj proizvod. U tom periodu, moguće je da startup izgradi i više nego što su korisnici tražili ili da se u potpunosti odvoji od korisničkih zahtjeva. Upravo zbog toga je u startup žargonu nastao i termin “Customer Development”, koji se koristi za opis paralelnog procesa izgradnje kontinuiranog odgovora kupaca na proizvod kroz cijeli ciklus razvitka proizvoda. “Customer

Development" proces leži u tome da startupovi moraju uključivati korisnike u sam razvitak svog proizvoda. No, iako cijelo poslovanje startup kompanija ovisi o kupcima, može ih se direktno priupitati što oni točno žele, zadatak startupa je da uz pravi kontekst korisnicima da jasno artikulirana riješenja , takozvani problem-solution fit.(Maurya ,2010,str.208)

Lean metodologija u svoje okrilje uzima nekoliko vrsta metodologija, a samu srž lean metodologije Maurya(2010) je podijelio u tri koraka:

- dokumentacija inicijalnog plana
- identifikacija riskantnih dijelova plana
- sistematsko testiranje proizvoda

Kod stvaranja dokumentacije inicijalnog plana, važno je istaknuti da većina startupova krene sa snažnim inicijalnim planom, no ipak ih većina ne uspije. Iako je snažna vizija za kreiranje proizvoda, kod lean metodologije je naglasak na podršci inicijalne ideje sa činjenicama. Dakako, neophodno je naglasiti da je inicijalna ideja izgrađena na uglavnom netestiranim pretpostavkama. Dakle, prva stavka je pisanje inicijalne vizije te njezino dijeljenje s drugim osobama. U tu svrhu se i pišu poslovni planovi, no nedostatak poslovnih planova jest što im nedostaje njihova prava svrha, dijeljenje inicijalne ideje s drugim ljudima.

Drugi važan dio lean metodologije jest identificiranje riskantnih dijelova plana, kako bi ih mogli na vrijeme tretirati. Važno je minimaliziranje rizika kod izgradnje uspješnog proizvoda. Identificiranje riskantnih dijelova plana je posebno bitno kod startupa, budući da su takve vrste kompanija dosta podložne rizicima, kod lean metodologije je zato najbitnije da se prvo mora pozabaviti rizicima u poslovanju. Najveći rizik koji može postojati u startupu jest kreiranje proizvoda koji korisnici ne žele koristiti. Riskantnost proizvoda varira o samoj prirodi proizvoda, pa se tako postavlja i ključno pitanje u svakom startupu a to je – postoji li problem koji vrijedi rješavati. Tako da prva faza u razvoju svakog prozivoda po lean metodologiji bi trebalo biti utvrđivanje jesmo li uspješno identificirali problem koji vrijedi riješavati. Kako navodi Maurya (2010), kod utvrđivanja problema postavljaju se tri glavna pitanja:

- je li naš proizvod nešto što korisnik želi, tu je bitno razaznati je li naš proizvod nešto što korisnik mora imati jer mu dosta olakšava neki njegov problem (takozvani must-

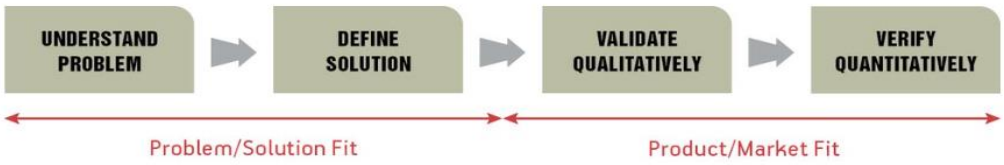
have) ili je to nešto što bi koristilo jer mu je prihvatljivo korištenje no ne rješava u tolikoj mjeri njegov problem (takozvani „nice to have“)

- je li korisnik spreman platiti za prozivod? - pitanje koje je bitno za utvrđivanje eventualne kapitalizacije ideje, te kreiranja raznih modela monetizacije.
- je li problem moguće riješiti? - pitanje koje se postavlja kako bi se utvrdilo je li to neki rješivi problem u skladu s kapacitetima startupova

Tijekom ove početne faze, pokušavaju se pronaći odgovori na gore postavljena pitanja koristeći tehnike intervjuiranja i anketiranja te opservacije potencijalnih korisnika. Na temelju tih anketa, kreće se u realizaciju minimalnih zahtjeva koji odgovaraju nekom problemu koji smo našli. Realizacija takvog proizvoda, koji odgovara minimalnim zahtjevima, naziva se MVP (Minimum viable product). Nakon što je izgrađen MVP, faza koja slijedi jest testiranje kako bi se utvrdilo odgovaraju li naša rješenja nađenom problemu. Faza koja slijedi nakon izgradnje i testiranja MVP-a jest skaliranje te generiranje rasta proizvoda, u toj fazi fokus se usmjerava na rast samog proizvoda, te skaliranje poslovnog plana.

Najvažnije je stvoriti proizvod koji odgovara tržištu te se stoga lean metodologija dijeli na fazu proizvoda i na fazu prije izgradnje proizvoda koji odgovara tržištu, te na fazu nakon izgradnje proizvoda koji odgovara tržištu. Kod faze izgradnje proizvoda koji odgovara tržištu, fokus je na izgradnji pivot proizvoda, dok je fokus u fazi nakon izgradnje proizvoda koji odgovara tržištu na optimizaciji i rastu. Pivot proizvod je termin koji je izmislio kreator Lean metodologije za startupove Eric Ries (2010) kako bi opisao promjene u smjeru startupa dok u isto vrijeme ostaju privrženi učenju navika korisnika. Glavna razlika između pivota i optimizacije proizvoda jest u tome što je cilj pivota nalaženje plana koji funkcionira, dok je cilj optimizacije akceleracija tog plana. Kod pivot faze pokušavaju se validirati dijelovi inicijalne ideje kako bi se pronašao plan koji funkcionira. U optimizaciji se pokušavaju redefinirati dijelovi inicijalne ideje kako bi se poboljšao postojeći plan. U fazi prije izgradnje proizvoda koji odgovara tržištu glavni fokus je na maksimiziranju učenja. Nakon što je incijialni plan dokumentiran te su napravljeni prioriteti početnih rizika, faza koja slijedi jest sustavno testiranje plana. Lean metodologija to ostvaruje izvršavanjem serije testova. Pri testiranju autor Lean metodologije koristi termin validirane petlje za učenje, ili takozvana “Build-Measure-Learn” petlja. Početak razvitka takve petlje nastaje još u fazi izgradnje proizvoda kada

koristimo inicijalni set ideja kako bismo izgradili MVP, te taj MVP nadalje prezentiramo korisnicima te mjerimo njihov odgovor na proizvod.



Slika 1. Obrazac iteracije u Lean metodologiji

izvor: https://www.slideshare.net/sachidananda_bs/lean-methodology-59328170 pristupljeno:10.05.2017

Na slici 1. prikazan je iteracijski obrazac u Lean metodologiji. Prve dvije faze odnose se na dolazak do problema, dok se zadnje dvije faze odnose na dolazak do proizvoda spremnog za tržište. Jedan od glavnih razloga propadanja proizvoda jest neuspjeh u kreiranju jasnog kanala do ciljanih korisnika. Prateći Lean metodologiju ranog anketiranja potencijalnih kupaca, rano se dolazi i do izgradnje kanala prema korisnicima. No, ukoliko proizvod cilja na velik broj korisnika kako bi zaživio, takav put najvjerojatnije neće narasti iznad inicijalne faze, stoga je bitno razmišljati i o kanalima distribucije koji su prilagodljivi rastu. Kao neke od popularnijih načina distribucij (Maurya ,2010,str. 810) navodi:

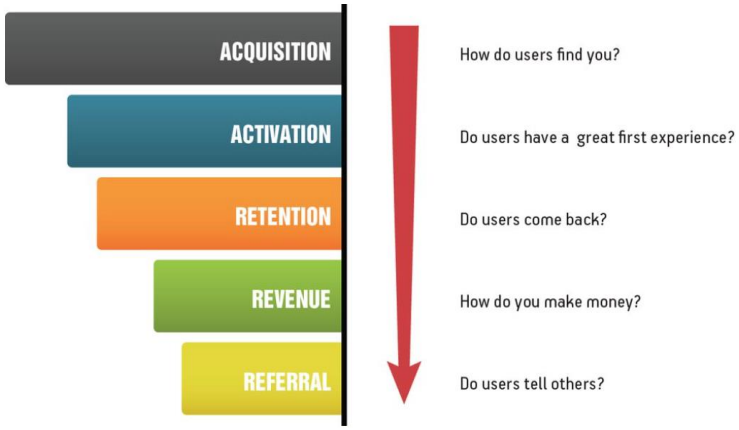
“Free versus paid” - iako sam naziv sugerira da je besplatan, važno je napomenuti kako ne postoje besplatni kanali distribucije, tako primjerice kanali za koje se asocira da su besplatni, kao SEO optimizacija, oglašavanje putem društvenih medija ili blogovi; iza sebe imaju veliki trošak ljudskog kapitala, pa je samim time i teško izračunati njihovu vrijednost.

“Inbound versus outbound” - takozvani inbound kanali koriste poruke koje dopuštaju da se kanali fokusiraju na pronalažanje ciljanih korisnika, dok je takozvani outbound kanal više fokusiran na odašiljanje poruka širokim masama. Ukoliko još nije izgrađen incijalni proizvod, teško je opravdati trošenje resursa na outbound kanale. No, intervju su tipovi outbound kanala koji se mogu smatrati iznimkama jer u većini slučajeva vrijednost od intervjua premašuje vrijednost obavljanja intervjua.

“Direct versus indirect” - ideja iza direktnog kanala jest spajanje s većim kompanijama kako bi se mogli iskoristiti njihovi kanali distribucije i krediblitet. No, problem je u tome

da dok kompanija iza sebe nema izgrađen i dokazan proizvod, neće moći privući pažnju velikih kompanija.

“Retention before referral” - u današnje vrijeme velikog utjecaja društvenih medija na društvo i korisnike, mnoge startup kompanije pokušavaju na sve načine izgraditi viralnost na društvenim mrežama koristeći referral programe.



Slika 2. Pirate Metrics model

Izvor: http://blog.popcornmetrics.com/6-essentials-of-marketing-analytics-a-beginners-guide-for-startup-growth-hackers/, pristupljeno: 10.05.2017

Na slici je prikazan takozvani Pirate Metrics model koji služi za mjerenje uspješnosti razvoja proizvoda.(Maurya ,2010,str.921) navodi detalje svakog od procesa kao:

1.) Skupljanje (Acquisition) - opisuje točku u kojoj nenamjerni posjetitelji postanu zainteresirani za proizvod. Primjerice, na web stranici način na koji natjeramo korisnika da ne napusti web mjesto se može smatrati kao akvizicija.

2.) Aktivacija (Activation) - opisuje točku u kojoj zainteresirani korisnik postane zadovoljan korisničkim iskustvom.

3.) Povratak korisnika (Retention) - opisuje broj korisnika koji su se vratili ili su angažirani s našim proizvodom.

4.) Prihodi (Revenue) - mjeri ključne događaje koji su doveli do porasta prihoda.

5.) Upućivanje (Referral) - mjeri načine na koji su korisnici upućeni na naš proizvod, to obuhvaća širok asortiman od viralnih videa do dijeljenja putem društvenih mreža.

U procesu razvoja softverskog proizvoda Lean metodologijom tri najvažnija dijela su – razvoj, dizajn i marketing. Pod razvojem se misli na snažne sposobnosti razvijanja softvera te iskustva u razvijanju jedno od ključnih stvari. Pod dizajnom se podrazumijeva i estetika samog proizvoda i korisničko iskustvo pri korištenju proizvoda. Pod marketingom se podrazumijeva vanjska percepcija proizvoda koji se razvija, a tu su od najvećeg značaja komunikacijske vještine.

Kao što je navedeno, drugi korak nakon izrade dokumentacije inicijalnog plana je provođenje eksperimenata. Neophodna je brzina u izradi proizvoda, učenje o proizvodu te fokusiranje na najbitnije kako bi se mogli izvršavati optimalni eksperimenti. Izostavljanje samo jednog od parametra može značiti lošiji proizvod, primjerice ukoliko je prisutna brzina u izradi proizvoda te je prisutan fokus no izostavljeno je učenje o proizvodu, može se dogoditi da se troši velika energija uzalud te se tako možemo vrtiti u krug. Ukoliko je prisutno učenje te fokus no proizvod se ne razvija dovoljno brzo može doći do nestanka resursa. Pri formatiranju eksperimenta izuzetno je bitno ostati fokusiran na ključne validacije koje želimo potvrditi tim testom, što može varirati od proizvoda do proizvoda. U eksperimentu je najzahtjevniji poduhvat postavljanje pravilne hipoteze.

Pri razvitku proizvoda Lean metodologijom postoje dvije vrste proizvoda, gore napomenuti MVP (Minimum viable product) koji predstavlja proizvod s minimalnim mogućnostima koji je spreman za tržište kako bi prikupio daljnje podatke, te postoji MMP (Minimum marketable probuct), što predstavlja proizvod koji je spreman za prodaju te predstavlja nešto što su programeri razvili bez provjeravanja odaziva korisnika. Cilj ovog rada je bio izraditi MVP proizvod s minimalnim zahtjevima koji bi bio spreman za daljnju adaptaciju, ovisno o korisničkim preferencijama. U svrhu boljeg razumijevanja MVP koncepta u nastavku teksta su prikazani primjeri MVP proizvoda:

1. Lego - ključ iza uspjeha kompanije Lego, koja uvijek plasira proizvode koji ostvare veliki uspjeh na tržištu, leži u ranom testiranju proizvoda. Pa tako njihovi dizajneri proizvoda nikada ne crtaju inicijalne skice krajnjeg proizvoda nego idu među svoje krajnje korisnike te im predaju male prototipe najnovijeg proizvoda kako bi spoznali njihovu zainteresiranost.

2.) Minecraft - popularna računalna igra Minecraft u svojim početcima 2009 godine nije ni malo ličila svojoj današnjoj verziji. Prva verzija Minecraft igrice je bila MVP. Samo

jedan programer je u šest dana stvorio inicijalni proizvod. Iako u prvoj igrici nije postojalo puno mogućnosti unutar same igrice, odaziv ljudi na inicijalnu ideju je bio dovoljan da programer može utvrditi da je to nešto što bi se ljudima sviđalo.

3.) Spotify - popularna aplikacija za slušanje glazbe je u svojim začetcima izgledala sasvim drugačije od ove današnje verzije. Sam Spotify je u početku također krenuo kao MVP, a inicijalna ideja osnivača je bila da bi ljudi radije uživo prenosili glazbu nego je kupovali te preuzimali na svoj uređaj. Budući da nisu mogli utvrditi je njihova pretpostavka istinita prije izdaje gotovog proizvoda, odlučili su se za izradu prototipa. Njihov prototip se sastojao samo od malog broja pjesama koje su oni posjedovali, no sam prototip se svidio ljudima te izdavačkim kućama koje su bile spremne na suradnju (Grossman, 2017,str. 161)

1.2 Scrum

Kod razvijanja softverskih proizvoda u zadnjih desetak godina sve učestalija postaje Scrum metoda kao metoda organizacije razvojnog tima. Scrum je metoda projektnog upravljanja koja je fokusirana na tim ljudi te njihovo mišljenje, njihovu fleksibilnost, kao i na dostavljanje samo proizvoda visoke kvalitete. Sam Scrum se opisuje kao dio takozvanog “Agile sotfware development”, koja je opisana kao metodologija razvoja softwara u knjizi “The Agile Manifesto” (Robert C. Martin, 2001). Osnovne vrijednosti Scrum metode odgovaraju vrijednostima “Agile software development-a” opisanog u navedenom djelu. Te vrijednosti su:

1.) ljudi i njihove interakcije su važniji od alata koji se koriste i procesa

2.) software je važniji od dokumentacije

3.) kolaboracija je važnija od pregovora

4.) praćenje plana kako bi se reagiralo na promjene

Scrum tim je podijeljen u tri glavne kategorije. Svaka od tih kategorija je ključna za upravljanje samim projektom. Te tri kategorije su:

1.) Razvojni tim - to je tim ljudi koji su odgovorni za razvijanje samog proizvoda. Dizajneri, programeri, pisci te svatko tko je uključen u samo kreiranje proizvoda.

2.) Vlasnik proizvoda - osoba koja je odgovorna za razumijevanje korisnika i razvojnog tima, drugim riječima vlasnik proizvoda predstavlja korisnike te razjašnjava njihove zahtjeve razvojnom timu.

3.) Scrum majstor - predstavlja menadžera odgovoronog za projekt. On je odgovoran za osiguravanje uvjeta razvojnom timu, te se brine da je proces razvoja kontinuiran.

Svaki tim, koji je strukturiran po pravilima Scrum metode, zadužen je za određene aktivnosti i prakse koje povećavaju preformanse i doprinose uspješnosti projekta. Takve aktivnosti zajednički tvore naputke za svaki Scrum projekt. Koraci za uspješno dovršavanje Scrum projekta su:

1. Planiranje projekta - iako nije Scrum aktivnost nego praksa unutar Agile metodologije, planiranje je inicijalni korak u svakom projektu. U toj fazi je kreirana sama vizija cijelog projekta.
2. Izdavanje projekta - u ovoj fazi tim planira nove mogućnosti proizvoda te dogovara nadolazeći datum izlaska. U Agilnim projektima svako novo izdavanje projekta se nanovo planira.
3. Sprint - sprint predstavlja ponavljajuće i regularne cikluse u izgradnji projekta. Sprint se također naziva i iteracija i traje između jednog do četiri tjedna.
4. Planiranje sprinta - svaki sprint započinje sastankom na kojemu tim odlučuje o ciljevima sprinta.
5. Svakodnevni Scrum sastanak - predstavlja svakodnevni sastanak u trajanju od petnaest minuta na kojemu tim diskutira o prijašnjim planovima i aktivnostima, te određuje prioritete za sljedeći tjedan.

6. Scrum izvještaj - na kraju svakog sprinta, vlasnik projekta održava sastanak na kojem razvojni tim prezentira kreirani proizvod.

7. Retrospektiva sprinta - predstavlja sastanak koji se održava na kraju svakog sprinta gdje tim diskutira o sprintu, te raspravlja o eventualnom optimiziranju projekta.

1.3 Sastavljanje i interpretiranje rezultata ankete

Kao što je napomenuto u uvodu, glavna motivacija za ovaj diplomski rad jest uviđen raskorak između sadržaja koji je prezentiran u kulturnim institucijama i

turističkim rutama u Istarskoj županiji te zainteresiranost mlađih populacija za isti taj sadržaj. Važno je za napomenuti kako je ovo subjektivna misao autora te je upravo iz tog razloga napravljena anketa za istraživanje tržišta – kako bi se glavna motivacija za rad potvrdila od strane korisnika.

Anketa za istraživanje tržišta obuhvatila je sveukupno 21 ispitanika. Ispitanici su bili turistički djelatnici te djelatnici u kulturnim institucijama . Sama anketa se sastoji od dva glavna seta pitanja. U prvom setu pitanja ispitanike se ispituje o utjecaju multimedijalnog sadržaja, dok se u drugom setu pitanja ispitanike ispituje o njihovim osobnim podatcima kao što su grad u kojem ispitanik živi, zanimanje te spol. Pri sastavljanju anketa pokušala su se formatirati pitanja na način da se pokuša iz njih izvući opći zaključak, pa tako prvo set pitanja glasi:

- smatra li ispitanik da kulturne institucije mogu svoj sadržaj učiniti pristupačnijim korištenjem mobilnih tehnologija – takvim pitanjem se pokušalo doći do općenitog zaključka o tome je li uopće potrebna izrada ikakvih aplikacija za kulturne institucije, jer ukoliko većina ispitanika odgovori negativno onda nema smisla upuštati se u daljnje faze projekta.

-“Jeste li se ranije susretali s turističkim sadržajima koji su zamišljeni kao "treasure hunt avanture” – cilj ovog pitanja jest profilirati koliki se postotak ispitanika upoznao i susreo s konceptom “treasure hunt” avanture, kako bi se mogao izvesti zaključak o novini i inovativnosti aplikacije na prostoru Istarske županije.

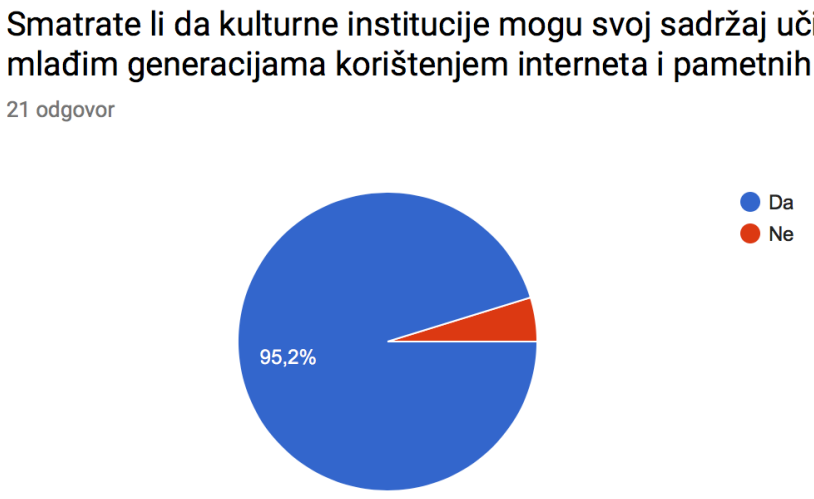
- “Na skali od 1 do 5, koliko ste zainteresirani za aplikaciju u kojoj biste mogli sastavljati interaktivnu rutu koju bi posjetitelji mogli istraživati putem svojih mobilnih uređaja?” – tim pitanjem pokušao se izvući opći zaključak o zainteresiranosti ispitanika za glavnu ideju aplikacije, kako bi se utvrdilo treba li se preoblikovati sama inicijalna ideja.
- “Na skali od 1 do 5, koliko ste zainteresirani za platformu koja bi vam omogućila dinamičko uređivanje multimedijalnog sadržaja vaše institucije?” – ovim pitanjem pokušava se doći do općenitog zaključka o tome koliko su korisnici zainteresirani za administratorsku web platformu unutar koje bi mogli manipulirati svojim sadržajem unutar institucija.

-“Koji bi parametar najviše uzeli u obzir prilikom odlučivanja o kupovini iznad navedenih aplikacija?” - ovim pitanjem pokušava se doći do općenitog zaključka o procjenjivanju vrijednosti same aplikacije te o tome na što se mora najviše obratiti pažnja prilikom realizacije same ideje kako bi ona i zaživjela na tržištu. Ponuđeni odgovori su bili: 1.) cijena platforme (je li ispitanicima bitno da je cijena kompetentna) 2.) dizajn (je li ispitanicima bitno kako aplikacija izgleda te na koji je način prezentirana) 3.) korisnička podrška (je li ispitanicima bitna korisnička podrška) 4.) broj ostalih institucija koje koriste aplikaciju.

-“Kojim kanalima biste preferirali preuzeti aplikaciju?” - ovim pitanjem pokušava se doći do općenitog zaključka o tome na koji način bi ispitanici najradije preuzeli aplikaciju, a ponuđeni odgovori bili su: 1.) Na službenim stranicama razvojnog tima (žele li ispitanici aplikaciju preuzeti direktno s web mjesta) 2.) Nakon demo prezentacije (žele li ispitanici aplikaciju preuzeti nakon što im je prezentirana) 3.) Nakon probe besplatne verzije (žele li ispitanici preuzeti aplikaciju nakon što su isprobali besplatnu verziju).

Pitanja u drugom setu bila su osobnog karaktera, dakle u kojem mjestu ispitanik živi, u kojoj instituciji radi te na kojoj poziciji – kako bi se jasnije profilirao tip ispitanika, odnosno je li zaposlenik kulturne institucije ili nije.

U nastavku su objašnjeni rezultati ankete za istraživanje tržišta kroz grafikone te interpretaciju rezultata.

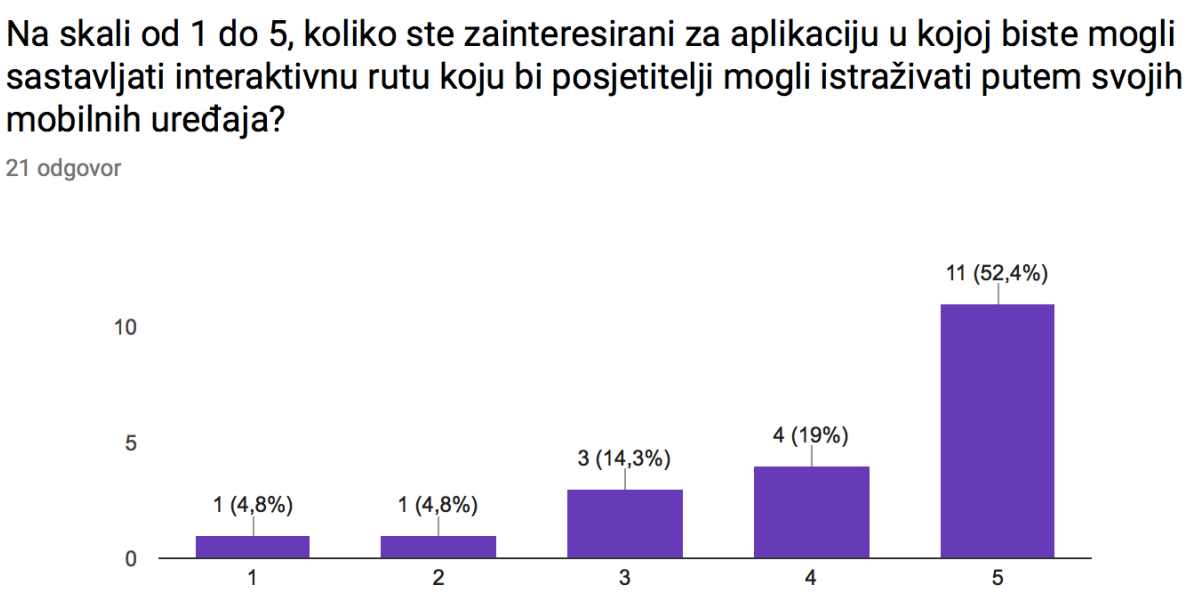


Slika 3 Grafikon rezultata odgovora na prvo pitanje ankete, izvor: izradio autor

Slika prikazuje rezultate ankete za istraživanje tržišta, točnije odgovore na prvo pitanje. Na prvo pitanje odgovorilo je 21 od ukupno 21 ispitanika, te su većinom od 95,2% odgovorili afirmativno na pitanje “Smatrate li da kulturne institucije mogu svoj sadržaj učiniti pristupačnijim mlađim generacijama korištenjem interneta i pametnih telefona?” Iz takvog rezultata dalo bi se zaključiti da među korisnicima postoji zainteresiranost za novim načinima na koji bi se moglo stimulirati mlađe posjetitelje da posjećuju kulturne institucije i koriste turističke rute s naglašenim kulturnim sadržajima.

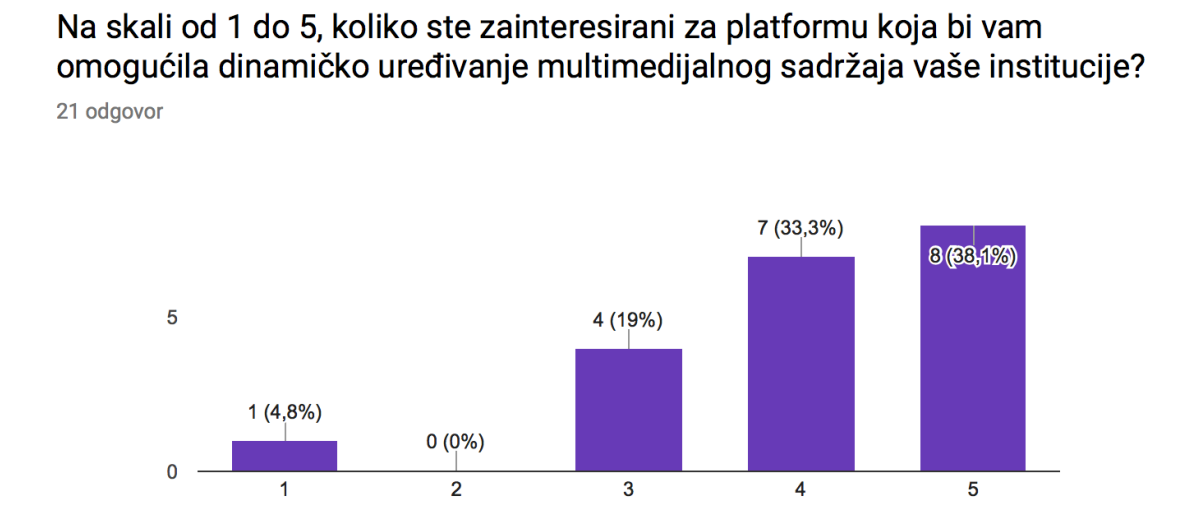
Slika 4. Grafikon rezultata odgovora na drugo pitanje ankete, izvor: izradio autor

Na slici je prikazan odgovor na drugo pitanje koje glasi “Jeste li se ranije susretali s turističkim sadržajima koji su zamišljeni kao “treasure hunt” avanture?”. Kao što je evidentno sa slike, većina ispitanika se nikad nije susrela s takvim i sličnim sadržajima točnije njih 52,4%, dok su se s takvim sadržajem susrelo 47,6 posto ispitanika. Iz ovog odgovora zaključuje se da većina ispitanika takav tip sadržaja smatra inovativnim, te da percipiraju takav sadržaj kao nešto što nedostaje u turističkoj ponudi.



Slika 5. Grafikon rezultata odgovora na treće pitanje ankete, izvor: izradio autor

Na slici je prikazan odgovor na treće pitanje koje ima za cilj provjeriti kolikom postotku ispitanika se svidjela inicijalna ideja aplikacije. Dakako, vidljivo je iz grafikona da su odgovori dosta šaroliki, tako primjerice 4,8% ispitanika ne bi bilo ni malo zainteresirani za platformu, dok bi u isto vrijeme većina od 52,4% ispitanika bila vrlo zainteresirana za takvu vrstu aplikacije. Međutim, ukoliko se uzme u obzir da je također 14,3% posto ispitanika ravnodušno prema toj ideji, te 19% ispitanika zainteresirano, može se zaključiti da je većina ispitanika bila zainteresirana za takvu vrstu aplikacije.



Slika 6 Grafikon rezultata odgovora na četvrto pitanje ankete, izvor: izradio autor

Na slici je prikazan odgovor na četvrto pitanje koje ima za cilj utvrditi koliki je postotak ispitanika zainteresiran i za administratorsku platformu na kojoj bi mogli uređivati sadržaj svoje institucije. Rezultati su također poprilično šaroliki, pa je tako najviše ispitanika na skali od jedan do pet dalo maksimalnu ocjenu pet, te se može izvući zaključak da je većina ispitanika zainteresirana za platformu budući da je većina dala ocjenu četiri i pet.

Slika 7 Grafikon rezultata odgovora na peto pitanje ankete, izvor: izradio autor

Na slici su prikazani odgovori na pitanje koje za cilj ima utvrditi do čega je ispitanicima najviše stalo prilikom odlučivanja o eventualnoj kupovini aplikacije. Odgovori su ravnopravno raspoređeni, a evidentno je da je ispitanicima najviše stalo do kompetitivne cijene aplikacije na što je odgovore dalo 50% ispitanika, zatim do dizajna te načina na koji je sama aplikacije prezentirana (35%), potom do korisničke podrške koja se pruža (10%), dok bi najmanji broj ispitanika gledao broj ostalih institucija koji koriste aplikaciju, odnosno popularnost same platforme.

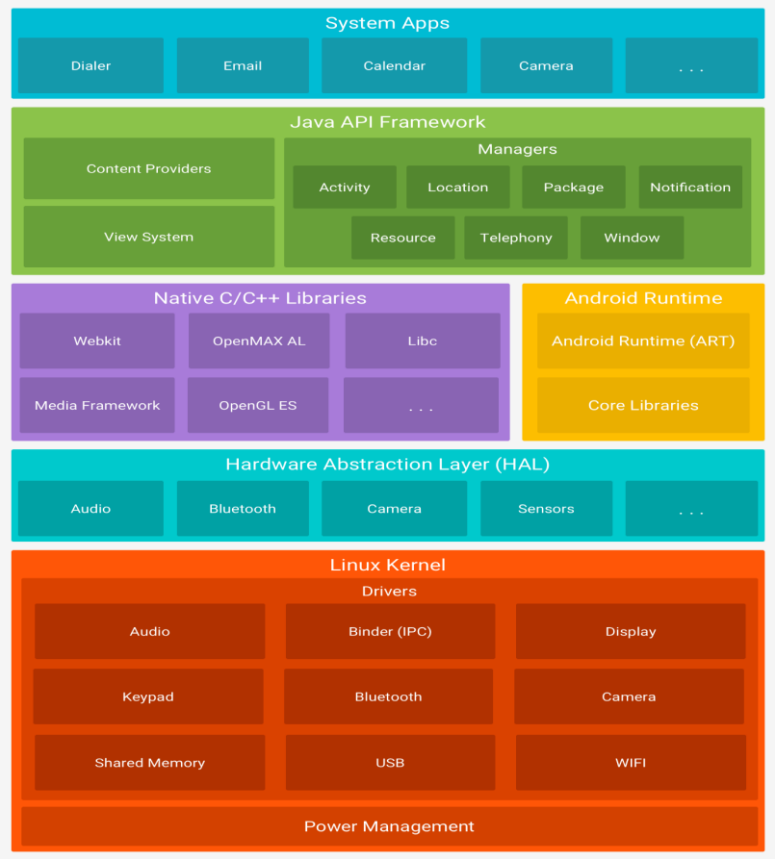
Kratkim istraživanjem tržišta utvrđeno je da postoje potencijali za razvijanje platformi namijenjenih kulturnim institucijama i turističkim agencijama, te da sami korisnici smatraju takvu vrstu aplikacije interesantnom. U nastavku ovog rada opisane su tehnologije koje su korištene u razvoju rješenja, te je opisan sam razvoj kroz prikaz programskog koda.

2. Tehnologije i obrasci korišteni u razvoju aplikacije

Druga cijelina ovog diplomskog rada obuhvaća opis tehnologija korištenih u razvoju programskog riješenja za HistriApp platformu. U nastavku cijeline okvirno su opisane sve glavne tehnologije korištene u razvoju HistriApp platforme. Prvotno je opisano okruženje na kojem se izvodi mobilna aplikacija namijenjena turistima, a zatim i okruženje na kojem se izvodi web aplikacija namijenjena institucijskim partnerima.

2.1 Operacijski sustav Android

Operacijski sustav Android je open-source operacijski sustav čiji je kod dostupan stručnoj zajednici za modificiranje i izradu različitih preinaka. Operacijski je sustav podijeljen u nekoliko slojeva, te se bazira na linux jezgri.



Slika 8. Stog operacijskog sustava Android, izvor: https://developer.android.com/guide/platform/index.html , pristupljeno:10.05.2017

Na slici je prikazan stog operacijskog sustava Android koji se temelji na Linux jezgri, pri čemu je bitno naglasiti da postoji razlika između Linuxa koji je jezgra Android operacijskog sustava i samog Linux operacijskog sustava, budući da Linux operacijski sustav koristi veliki broj programa koje Linux koristi a nisu sadržani u Linux jezgri

Android operacijskog sustava. Google razvojni inženjeri mogli su izmjenjivati jezgru Android operacijskog sustava zbog Linuxovog svojstva otvorenog koda. Dakle, umjesto da pokrene tipičnu Linux aplikaciju Android koristi svoje okruženje za pokretanje aplikacija. U početku se koristio „Dalvik virtual machine“ dok se trenutno koristi „ART“ okruženje za pokretanje mobilnih aplikacija. Android platforma također iskorištava mnoge prednosti koje pruža Linux operacijski sustav, pa tako Android operacijski sustav dodjeljuje jedinstveni korisnički identifikator (UID) svakoj Android aplikaciji te ju pokreće u zasebnom procesu. Ovaj pristup je drugačiji od klasičnog pristupa kojeg ima Linux, gdje više aplikacija koristi istu autorizaciju za korištenjem resursa. Takav način omogućava stvaranje takozvanog Sandboxa na jezgri operacijskog sustava. Budući da je Sandbox u jezgri, ovaj model sigurnosti se nadovezuje na nativni kod i na aplikacije operacijskog sustava. Svi programi iznad jezgre na stogu, kao što su radni okviri za aplikacije, programski paketi za operacijski sustav se pokreću na Android Sandboxu. Budući da se sve aplikacije i resursi nalaze unutar sandboxa na razini operacijskog sustava, problemi s potrošnjom memorije su zadržani samo unutar konteksta aplikacije koja se pokreće, a ne cijelog operacijskog sustava.

Iznad Linuxe jezgre u stogu Android operacijskog sustava nalazi se Hardware abstarction layer (HAL) koji definira standardno sučelje za hardware. HAL dozvoljava implementaciju funkcionalnosti bez modificiranja sistema na nivou iznad operacijskog sistema. (HAL) implementacija je pakirana u modul (.so) datoteku i kada je primjereno učitana u Android operacijski sustav. Moguće je izgraditi nekoliko modula kreiranjem Android.mk datoteke za svaku od HAL implementacija.

U stogu Android operacijskog sustava iznad HAL-a nalazi se Android runtime (ART) koji je u osnovi upravljački sustav koji koriste aplikacije kako bi pristupale servisima na Android-u. ART i njegov predhodnik Dalvik su originalno bili kreirani kao specifični Android projekti. ART je kompatibilni izvođač koda koji se pokreće na Dex bytecode-u, što znači da bi aplikacije koje su razvijane za Dalvik trebale raditi pri pokretanju na ART-u. Međutim, neke tehnike koje rade na Dalvik-u ne rade na ART-u.

Uz Art na istom nivou stoga operacijskog sustava Android nalaze se i nativne C/C++ programske biblioteke koje su zadužene za dostavljanje širokog asortimana funkcija uključujući 2d i 3d grafičke crteže, Secure Sockets Layer (SSL) komunikaciju, SQLite

baze podataka, audio i video biblioteke itd. U praksi Android programer će pristupiti ovim bibliotekama samo putem Android core library-a.

U nivou iznad nativnih C++ programskih biblioteka te ART-a nalaze se radni okviri napisani u programskom jeziku Java. Ti radni okviri pružaju cjeloukupna svojstva Android operacijskog sustava koja su dostupna korisniku putem API-ja napisanih u Java programskom jeziku. Takvi API-i pružaju temelje koji su potrebni programerima za izradu Android aplikacija tako što pojednostavljuju ponovno korištenje koda te nude razne sistemske servise, kao što su View System koji se može koristiti prilikom izrade korisničkog sučelja aplikacije, uključujući listu, gridove te ugrađene Internet preglednike. API servisi također pružaju i upravitelje resursima, notifikacijama i aktivnostima. Upravitelj resursima pruža programerima pristup resursima kao što su lokalizirani znakovi, grafikoni te posebne datoteke korisničkog sučelja; dok upravitelj notifikacijama omogućuje svim aplikacijama da prikažu prilagodljive poruke korisniku.

Na vrhu stoga Android arhitekture nalaze se sistemske aplikacije koje dolaze uz jezgru Android uređaja. Takve aplikacije mogu varirati, ovisno o Android operacijskom sustavu, no uglavnom sadrže razne aplikacije za e-mail, SMS poruke, kalendar, Internet preglednik itd. Sistemske aplikacije imaju dvije funkcije; jedna je da služe kao aplikacija za korisnike kako bi pružili ključne sposobnosti sistema kojima programeri mogu pristupiti putem svojih aplikacija. Primjerice, ako programer razvija aplikaciju za slanje SMS poruka, nije potrebna izrada funkcionalosti slanja, nego je umjesto toga dovoljno pozvati sistemsku aplikaciju za slanje SMS poruka.

2.2 Android studio

Android studio je službeno razvojno okruženje za Android aplikacije koje se temelji na Java integriranom razvojnom okruženju zvanom IntelliJ. Uz IntelliJ-ov urednik programskog koda, Android studio također nudi korisnicima dodatne mogućnosti koje mogu povećati preformanse Android aplikacija, u službenoj dokumentaciji Android operacijskog sistema autori navode:

1. Sustav izgradnje programskog sustava temeljenog na Gradle sustavu

2. Emulator android aplikacija

3. Unificirano razvojno okruženje na kojemu se mogu razvijati aplikacije za sve vrste Android uređaja.

4. Predlošci koda te potpuna integracija s Git sustavom

5. Podrška za integraciju C++ programa.

Svaki projekt u Android sustavu sadrži jedan ili više modula sa svojim programskim kodovima i datotekama resursa. Prikaz strukture projekata je organiziran po modulima kako bi se pružio brzi pristup ključnim datotekama sustava. Sve Gradle skripte za svaki od podmodula dostupne su na vrhu strukture projekta, a svaki modul mora sadržavati sljedeće grupe datoteka:

1. manifests: sadrži AndroidManifest.xml datoteku

2. java: sadrži programski kod napisan u Java programskom jeziku

3. res: sadrži sve resurse potrebne aplikaciji koji nisu vezani uz programski kod, uključujući XML datoteke, znakove za korisničko sučelje te slike

Kao temelj svog sistema za izradu projekta Android studio koristi Gradle. Gradle je napredni upravljački sustav koji se temelji na Groovy-u i Kotlinu. Gradle podržava automatsko preuzimanje i konfiguriranje ostalih biblioteka i radnih okvira koji su dodani u projekt. Gradle skripta se sastoji od jednog ili više projekata. Projekti mogu biti nešto što se treba iznova izgraditi ili nešto što se treba izvršiti. Svaki projekt se sastoji od različitih zadataka. Zadatak predstavlja dio rada koji izvršava izgradnja sustava, primjerice prevođenje izvornog koda pomoću Javadoc-a. Projekt koji koristi gradle opisuje postupak svoje izgradnje unutar „build.gradle“ datoteke. Ova datoteka se nalazi u korijenu datoteke za projekt. Build datoteka za Gradle bazira se na takozvanom DSL-u (Domain Specific Language). Unutar te datoteke mogu se kombinirati deklaracije i imperativne rečenice. Također se može pisati i Groovy ili Kotlin kod, kada god je to potrebno. Također se unutar gradle skripte mogu pisati i zadatci.

```
task hello {
    doLast {
        println 'Hello Gradle'
    }
}
```

Slika 9. Deklaracija Gradle zadataka, izvor: izradio autor

U primjeru koda iznad prikazan je primjer deklaracije Gradle zadatka koji se izvodi upisivanjem komande „gradle hello“ u komandnu liniju.

Svaka aplikacija također mora imati svoju AndroidManifest.xml datoteku unutar root direktorija. Manifest datoteka pruža ključne informacije o aplikaciji Android operacijskom sustavu, koje sustav mora imati prije nego može pokrenuti kod aplikacije. Manifest datoteka obavlja sljedeće zadatke:

1. Imenuje Java pakete potrebne aplikaciji. Ime paketa služi kao jedinstveni identifikator aplikacije.

2. Opisuje komponente aplikacije, što uključuje aktivnosti, servise, primatelje broadcast poruka te „content providere“ koji tvore aplikaciju. Također imenuje klase koje implementiraju svaku od komponenata aplikacije. Takve deklaracije informiraju Android sustav o komponentama i uvjetima unutar kojih se mogu pokrenuti.

3. Određuje procese aplikacijskih komponenti.

4. Deklarira dopuštenja koje aplikacija mora imati kako bi pristupila zaštićenim sustavima API-a te kako bi komunicirala s ostalim aplikacijama.

Sama AndroidManifest datoteka sastoji se od različitih elemenata. No unutar AndroidManifest datoteke samo <manifest> te <application> elementi su obavezni. Svaki od ta dva elementa mora biti prisutan unutar AndroidManifest datoteke te se mora pojavljivati samo jedanput. Većina drugih elemenata se može pojavljivati više puta ili nijednom. Međutim, samo neki od njih moraju biti prisutni kako bi manifest datoteka postala korisna. Ako element nešto sadrži onda sadrži druge elemente. Sve vrijednosti su namještene unutar atributa. Elementi na istom nivou nisu sortirani, primjerice <activity>,<provider>, i <service> elementi mogu biti izmiješani, no postoje dvije iznimke:

-Element <activity-alias> mora slijediti <activity> kojem pripada

-Element<application> mora biti na zadnjem elementu unutar <manifest> elementa.

Jedan od dijelova AndroidManifest datoteke su također i atributi. Općenito govoreći svi atributi su opcionalni. Međutim, postoje određeni atributi koji moraju biti specificirani kako bi element mogao postići svoju svrhu. Izuzev nekih elemenata unutar <manifest> elementa, svi atributi moraju započeti s “android:” prefiksom. Primjerice, android:alwaysRetainTaskState.

Mnogi elementi korespondiraju Java objektima, uključujući elemente same aplikacije (<application> element) te njezinih komponenti: aktivnosti (<activity>), servisa (<service>), broadcast receivera (<receiver>), te content provider-a (<provider>). Ukoliko definiramo podklasu, ona se mora definirati putem atributa „name“, koji mora sadržati kompletan put do navedene klase. Primjerice, klasa Service se može definirati kao:

```
<manifest . . . >
<application . . . >
<service android:name="com.example.project.SecretService" . . . >
. . .
</service>
. . .
</application>
</manifest>
```

Slika 10. Primjer definiranja klase u AndroidManifest-u , izvor: https://developer.android.com/guide/topics/manifest/manifest-intro.html, pristupljeno: 10.05.2017

Međutim ako je prvi karakter unutar stringa znak „.“ , onda je ime paketa aplikacije pridodano nizu znakova, primjerice:

```
<manifest package="com.example.project" . . . >
<application . . . >
<service android:name=".SecretService" . . . >
. . .
</service>
. . .
</application>
</manifest>
```

Slika 11. Primjer alternativnog definiranja klase u AndroidManifest-u izvor: https://developer.android.com/guide/topics/manifest/manifest-intro.html, pristupljeno: 10.05.2017

Pokretanjem komponenti, Android operacijski sustav kreira instancu navedene podklase. Ukoliko podklasa nije specificirana, kreira instancu bazne klase.

Ključne komponente aplikacije, kao što su aktivnosti, servisi i broadcast receiveri, aktivirani su takozvanom namjerom. Namjera je skup informacija koji opisuju određenu akciju, uključujući podatke nad kojima će se akcija izvršavati, kategorija komponente

koja bi se trebala izvršiti, te mnoge druge instrukcije. Android operacijski sustav locira komponentu koja odgovara intentu koji se poziva, pokreće novu instancu komponente ako je ona potrebna te ju proslijedi u objekt namjere. Komponente obavještavaju tip namjene kojem će se odazvati putem filtera namjere. Budući da Android sustav mora naučiti namjere koje komponenta želi obraditi, filteri namjere se deklariraju unutar AndroidManifest datoteke kao <intent-filters> elementi. Komponente mogu imati veliki broj filtera, te svaki može opisivati drugu sposobnost. Namjera koja ekspilicitno imenuje komponentu aktivira imenovanu komponentu, te onda <intent-filter> nije potreban, no ukoliko namjera ne specificira pozivanje komponente imenom, onda se ona aktivira samo pozivanjem putem <intent-filter> elementa.

Permission ili na hrvatskom dopuštenje predstavlja restrikciju koja može limitirati pristup kodu ili datotekama na uređaju. Limitacija se impozira kako bi se zaštitile ključne informacije i kod koji se može iskoristiti kako bi se pogoršalo korisničko iskustvo. Svaki permission je identificiran pomoću svog jedinstvenog identifikatora. Vrlo često ti identifikatori identificiraju akciju koja je zaštićena. Primjer deklariranja permissiona u kodu je prikazan ispod.

- android.permission.CALL_EMERGENCY_NUMBERS
- android.permission.READ_OWNER_DATA
- android.permission.SET_WALLPAPER
- android.permission.DEVICE_POWER

Ukoliko aplikacija treba pristupiti mogućnostima koji su zaštićeni pomoću Permissiona, to se mora specificirati unutar <uses-permission> elementa u manifestu. Kada je aplikacija instalirana na uređaj, sustav odlučuje hoće li dozvoliti pristup komponenti provjeravanjem Permission elementa u AndroidManifest datoteci. Ukoliko je prisutan traženi Permission aplikacija može koristiti mogućnost sustava. Također aplikacija može zaštititi svoje vlastite komponente pomoću permissiona. Može iskoristiti svaku od Permission-a koji su definirani Android sustavom, te koji su izlistani unutar “android.Manifest.permission-a”, ili koji su deklarirani putem drugih aplikacija. Također može definirati svoj vlastiti Permission.

```
<manifest . . . >
    <permission android:name="com.example.project.DEBIT_ACCT" . . . />
        <uses-permission android:name="com.example.project.DEBIT_ACCT" />
```

```
. . .
<application . . .>
<activity android:name="com.example.project.FreneticActivity"
android:permission="com.example.project.DEBIT_ACCT"
. . . >
. . .
</activity>
</application>
</manifest>
```

Slika 12. Primjer deklariranja Permission-a u AndroidManfiest-u izvor: https://developer.android.com/guide/topics/manifest/permission-element.html, pristupljeno: 10.05.2017

U primjeru iznad, DEBIT_ACCT Permission nije deklariran samo <permission> elementom, nego se njegova upotreba također dopušta <uses-permission> elementom. Mora se zatražiti njegovo korištenje kako bi se pokrenula zaštićena aktivnost, iako je zaštita nametnuta samim sustavom.

Kao programski jezik za razvijanje aplikacija za Android operacijski sustav najrasprostranjenija je Java. Java predstavlja objektno-orijentirani jezik treće generacije koji je razvijen 1995. godine. Kao objektno orijentirani jezik, Java sadrži i osnovna svojstva objektno orijentiranog programiranja kao što su:

1.) Enkapsulacija – predstavlja proces vezivanja više podatkovnih članova i funkcija u jednu cjelinu.

2.) Apstrakcija – apstrakcija podataka nudi mogućnost smanjivanja redundantnosti koda stvaranjem apstraktnih klasa ili sučelja za objekte koji su slični.

3.) Polimorfizam – nudi mogućnost preopterećivanja metoda, tj. moguće je definirati nekoliko metoda istog imena, a svaka će kao parametre primati objekte različitih tipova.

4.) Nasljeđivanje – predstavlja mogućnost objekata za nasljeđivanje karakteristika nekog drugog objekta.

Jezik za razvijanje korisničkog sučelja u Android studiu jest XML. XML predstavlja jezik koji definira set pravila za enkodiranje dokumenta u formatu koji je čitljiv kako ljudima tako i stroju. Cilj XML-a je jednostavnost, generalnost i iskoristljivost širom interneta. To je tekstualni format podatka sa snažnom podrškom Unicoda za različite ljudske

jezike. U kontekstu razvoja Android aplikacija XML se koristi kao jezik za razvoj korisničkog sučelja. Prednost deklariranja korisničkog sučelja u XML-u jest taj što nam daje veću kontrolu nad samim sučeljem. Općenito, XML vokabular za deklariranje UI elemenata slijedi strukturu i imenovanje klasa i metoda, gdje ime elementa korespondira imenu klase i ime atributa korespondira metodama. Svaka datoteka korisničkog sučelja mora imati jedan root element, koji mora biti ili tipa View ili tipa ViewGroup objekta. Jednom kada su definirani root elementi, mogu se dodati objekti korisničkog sučelja kao dijete root elementa.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

Slika 13. Primjer korištenja vertikalnog LinearLayout elementa, izvor: https://developer.android.com/guide/topics/ui/declaring-layout.html, pristupljeno: 10.05.2017

U primjeru iznad prikazan je primjer koji koristi vertikalni LinearLayout root element koji unutar sebe sadrži jedan element tipa „Button“ i jedan element tipa „TextView“. Nakon što je korisničko sučelje definirano u XML-u sprema se u „res/layout“ direktorij, kako bi se ispravno prevelo.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Slika 14. Učitavanje XML korisničkog sučelja, izvor: https://developer.android.com/guide/topics/ui/declaring-layout.html, pristupljeno: 10.05.2017

Za učitavanje XML korisničkog sučelja unutar aktivnosti poziva se metoda „setContentView()“ kao što je prikazano u primjeru iznad.

2.3 Javascript i MEAN stack arhitektura

JavaScript je programski jezik koji uz HTML i CSS predstavlja glavnu tehnologiju za razvijanje Web stranica. Danas većina web stranica koristi Javascript, te svi moderni Web preglednici ga podržavaju bez korištenja dodatnih priključaka. Javascript se također koristi i kod tehnologija koje se ne baziraju na web strukturi kao primjerice kod PDF dokumenata, također novije i brže JavaScript virtualne strojeve i platforme su također povećale popularnost Javascripta kod serverske strane razvijanja web aplikacija. Javascript svoju upotrebu također ima i kod razvijanja video igara, te kod razvijanja desktop i mobilnih aplikacija. JavaScript podržava većinu standardnih strukturnih programskih sintaksi (if petlje, while petlje, for petlje itd.). JavaScript je skoro u potpunosti objektno orijentiran jezik. Dok većina objektno orijentiranih jezika koristi klase, JavaScript koristi takozvani prototip koji mjenja način nasljeđivanja kod kojega se postojeći objekti ponovno iskorištavaju korištenjem procesa. JavaScript se uglavnom oslanja na okruženje u kojemu se izvodi, kao što su web preglednici kako bi pružili objekte i metode koje mogu komunicirati s okruženjem u kojemu se izvode. Budući da se JavaScript kod može pokrenuti lokalno na korisnikovom web pregledniku (a ne na udaljenom poslužitelju), preglednik brzo može odgovoriti na aktivnosti korisnika. Nadalje, JavaScript kod može otkriti korisničke akcije koje HTML sam ne bi mogao interpretirati, kao što su pojedinačne tipke. Većina logike korisničkog sučelja napisana je u JavaScriptu.

Javascript engine (također poznat kao JavaScript tumač ili JavaScript provedba) je tumač koji interpretira JavaScript kod i izvršava scenarij u skladu s napisanom skriptom. Prvi JavaScript engine za Netscape Navigator web preglednik izradio je Brendan Eich u Netscape-u. Taj engine, pod šifrom SpiderMonkey bio je implementiran u jeziku C. Otada je Javascript engine ažuriran (u JavaScriptu 1.5) u skladu s ECMAScript 3.

MEAN stog predstavlja besplatan programski stog otvorenog koda, namijenjenog za izgradnju dinamičkih web stranica i aplikacija. MEAN stog je svoj nazive dobio po komponentama koje ga koriste, te komponente su:

- MongoDB – besplatna platforma otvorenog koda koja služi za stvaranje objektno orijentiranih baza podataka. MongoDB klasificira se kao NoSQL baza podataka te koristi JSON strukturu za spremanje podataka. MongoDB baza podataka podržava polja, operacije upita nad bazom te pretraživanje regularnih izraza. Indeksiranje u MongoDB dokumentima može se postići s primarnim i sekundarnim indeksima baza podataka. MongoDB također podržava repliciranje podataka.

- ExpressJS – besplatni radni okvir za razvijanje dinamičkih web aplikacija otvorenog koda. Dizajniran je za razvijanje API-ja za web aplikacije te danas predstavlja standardni serverski radni okvir za Node.js. ExpressJS predstavlja backend dio MEAN stog arhitekture.

- AngularJS – predstavlja radni okvir otvorenog koda koji održava Google i programerske zajednice zainteresirane za razvoj takozvanih single-page web aplikacija. U kontektstu MEAN stoga AngularJS predstavlja radni okvir za frontend, odnosno za prikazivanje podataka korisniku. AngularJS funkcionira tako što prvotno pročita HTML stranicu unutar koje su pohranjeni atributi. Angular interpretira takve atribute kao direktive za povezivanje dijelova stranica na model podataka koji je predstavljen standardnim JavaScript varijablama. Vrijednosti tih Javascript varijabli se može ručno namjestiti unutar koda, ili se mogu dohvatiti putem JSON resursa. Prema podatcima GitHuba, AngularJS je šesti najgledaniji projekt u povijesti Github zajednice.

- NodeJS – platforma otvorenog koda napisana u JavaScriptu, namijenjena za razvoj velikog raspona serverskih alata i aplikacija. Iako Node.JS nije Javascript radni okvir, veliki broj osnovnih modula je napisano u JavaScriptu. NodeJS se sastoji od takozvane „event-driven“ arhitekture koja je zadužena za asinkronizirane I/O operacije.

Prednost MEAN stog arhitekture je tome što omogućuje programerima da razviju cijelu web stranicu u jednom programskom jeziku – Javascriptu.

2.4 Algoritam za prepoznavanje slike VISION API

Razvojom digitalne fotografije počinje se javljati potreba za razvojem automatiziranih sustava za prepoznavanje sadržaja slike. Prije generalnog rješenja problema postoje mnoge poteškoće koje se moraju riješiti. Navedene se poteškoće javljaju iz sljedećih razloga:

1.) nije precizno definirano što čini sadržaj slike

2.) stupanj sličnosti među slikama uglavnom ovisi o kontekstu – tip slike i njezin kontekst često odlučuje o karakteristikama sadržaja same slike. Osim toga, tu je i semantički jaz između svojstva slike i njezine jezične interpretacije. Dakle, s obzirom na navedene poteškoće, učinkovit, objektivan i kvalitetan opis ili slikovni sadržaj u svrhu pretraživanja sličnosti između slika je složen zadatak.

Fundamentalni dio sadržaja slike je struktura, koja ovisi o različitim elementima definiranim kvalitetom uzimanja silke (veličina piksela). Stoga, kako se fokus algoritma ne bi promijenio na strukturana različitih veličina, važno je da imaju mogućnost rasta prema odgovarajućem mjerilu. Ako je veličina pojedine strukture mala, jednostavnije je procjena svojstava takve strukture. Kao algoritam za prepoznavanje slike u ovom radu korišten je Google VISION API algoritam, koji omogućava programerima da raspoznaju sadržaj slike inkapsuliranjem modela za strojno učenje u REST API. Taj algoritam klasificira slike u tisuće kategorija te na temelju njih detektira individualne objekte unutar slike. U ovom radu korištena je opcija LABEL_DETECTION koja detektira općenite karakteristike pronađene unutar slike. Komunikacija s API-jem se izvršava isključivo JSON formatom.

```
{
"requests": [
{
"images": {
"content": "/9j/7QBEUGhvdG9zaG9...base64-encoded-image-content...fXNWzvDEeYxxxzj/Coa6Bax//Z"
},
"features": [
{
"type": "LABEL_DETECTION"
}
]
}
]
}
```

Slika 15. Primjer upita poslanog VISION API-ju, izvor: https://cloud.google.com/vision/docs/request, pristupljeno: 11.05.2017

Primjer koda prikazuje upit poslan VISION API-ju u JSON formatu u kojemu je specifirano mjesto na kojemu se nalazi slika, a može biti u obliku url adrese ili u obliku uri-a koji ukazuje na put na serveru. Nadalje, nalazi se i tip detekcije koja se treba izvršiti.


```
{
 "responses": [
  {
   "labelAnnotations": [
    {
     "mid": "/m/01yrx",
     "description": "cat",
     "score": 0.92562944
    },
    {
     "mid": "/m/04rky",
     "description": "mammal",
     "score": 0.90815818
    },
    {
     "mid": "/m/01l7qd",
     "description": "whiskers",
     "score": 0.79939437
    },
    {
     "mid": "/m/07k6w8",
     "description": "small to medium sized cats",
     "score": 0.66373962
    },
    {
     "mid": "/m/0307l",
     "description": "cat like mammal",
     "score": 0.65950978
    }
   ]
  }
```

```
 ]
}
```

Slika 16. Primjer odgovora VISION API-ja, izvor: https://cloud.google.com/vision/docs/request, pristupljeno:10.05.2017

Primjer koda prikazuje odgovor VISION API-ja na ranije postavljen upit, također u JSON formatu. Kao odgovori dobiveni su parametri „mid“ koji predstavljaju jedinstvene identifikatore, „description“ koji predstavlja opis sadržaja slike te parametar „score“ koji predstavlja postotak točnosti ponuđenog opisa slike.

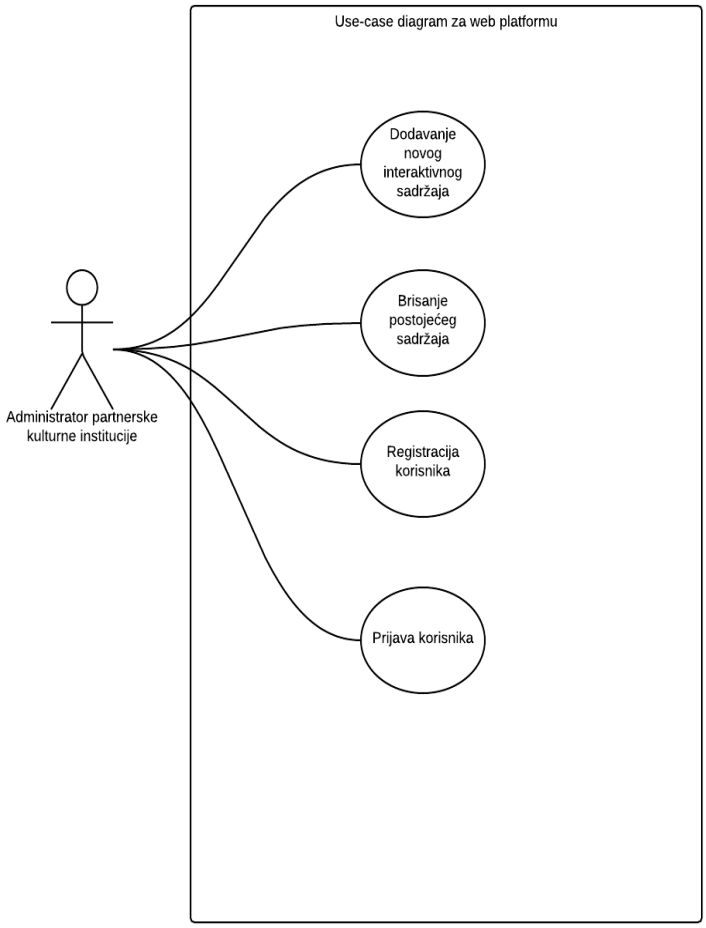
3. Realizacija HistriApp aplikacije

Treća cjelina ovoga rada opisuje sam proces realizacije HistriApp aplikacije. Pri realizaciji HistriApp platforme bilo je potrebno promotriti najoptimalniji pristup izrade arhitekture, kako se korisnički ugođaj ne bi promijenio te kako bi se zadržala temeljna funkcionalnost sustava. Temeljna funkcionalnost sustava podrazumijeva omogućavanje korisniku da za određenu fotografiju dobije odgovor iz Vision API-ja. U sljedećem poglavlju detaljno je razrađen sustav HistriApp platforme te su pojašnjeni pristupi koji su se koristili u razradi pojedinačnih komponenti sustava.

3.1 Opis sustava HistriApp platforme

HistriApp platforme koriste administratori, odnosno zaposlenici partnerskih institucija, koji unutar svoje ingerencije putem web sučelja imaju ovlasti prijave na platformu te dodavanja i brisanja sadržaja, a koriste ga i krajnji korisnici, tj. posjetitelji kulturnih institucija, koji unutar svoje ingerencije imaju mogućnost pregledavanja sadržaja partnerske institucije. Koncept sustava HistriApp platforme je stoga također podijeljen na web sučelje za administratore te na Android nativnu mobilnu aplikaciju za krajnje korisnike – posjetitelje kulturnih institucija.

U ovom podpoglavlju opisane su web i Android komponenta sustava, dakle dio sustava namijeljen zaposlenicima partnerskih institucija i dio sustava namijenjen posjetiteljima navedenih partnerskih institucija. Dakako, prikazani su dijagrami korištenja dva navedena dijela sustava, te je predočen ilustrirani prikaz tehnologija koje su korištene za razvoj svake komponente sustava.



Slika 17 Use-case diagram web platforme, izvor: izradio autor

Slika prikazuje slučaj korištenja web platforme. Web platformu prvenstveno koriste administratori web mjesta. Administratori mogu biti zaposlenici partnerskih institucija koji su u suradnji sa HistriApp sustavom, to su u prvom redu stručnjaci u kulturnim ustanovama te turistički vodiči. Za početno korištenje, dok sustav ne naraste do određene razine korištenja, limitiran je na jednog administratora po partnerskoj instituciji. Administrator institucije može biti zadužen za prijavljivanje u sustav.

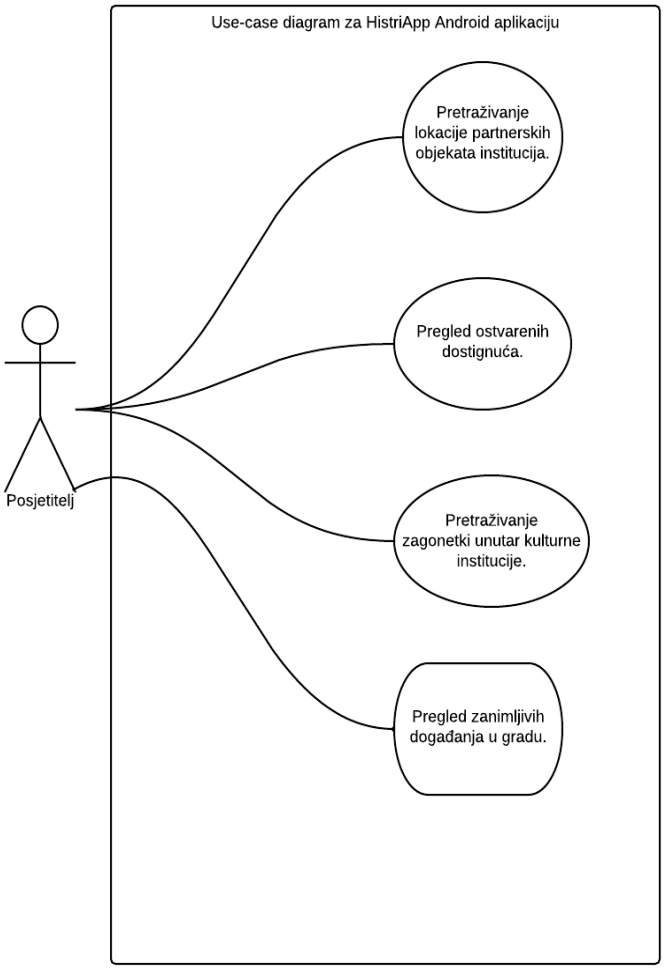
Pri početnoj stranici sustava korisnik može upisati svoju e-mail adresu te lozinku kako bi se prijavio u sustav. Ukoliko korisnik ne postoji bit će adekvatno obaviješten te može nastaviti na registraciju svog računa na HistriApp platformi.

Pri registraciji na web mjesto administrator mora unijeti:

- naziv institucije koju predstavljakorisničko ime koje će se vezati za korisnički račun administratora
- lozinku koja će se vezati za korisnički račun administratora
- e-mail adresu administratora na koji će mu kasnije biti poslan potvrdni link

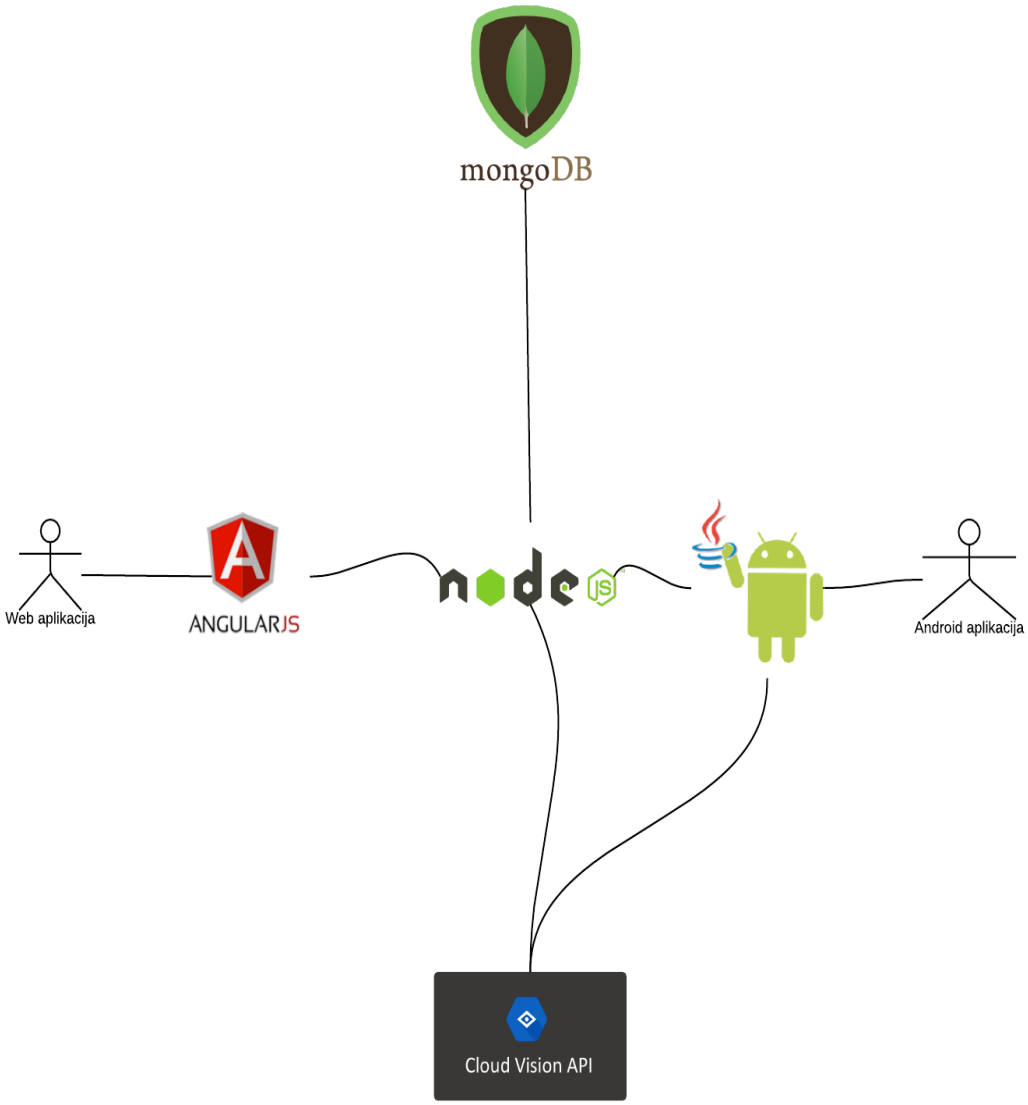
- adresu institucije koju predstavlja koja će se kasnije koristiti kako bi se korisniku prikazale upute do željene lokacije
- web stranicu institucije koja će se također kasnije korisniku prikazati na Android aplikaciji kako bi korisnik mogao pretraživati
- web mjesto partnerske institucije.

Pri uspješnom logiranju, korisnik se preusmjerava na administratorski web panel. Ovdje korisnik može obavljati osnovne CRUD operacije. Dakle, može dodavati nove sadržaje, izmljenjivati postojeće sadržaje te brisati postojeće sadržaje. Odabirom dodavanja novog sadržaja korisnik može dodati novi sadržaj u svoju bazu podataka. Potrebno je dodati ime sadržaja te sliku koja predstavlja naziv sadržaja, a kasnije se, nakon obrađivanja na VISION API-ju, rezultat sprema u bazu podataka. Pritiskom na opciju brisanja korisnik može izbrisati sadržaj iz sustava ili odabirom izmjene može izmijenjivati postojeći sadržaj, tako što će promijeniti naziv postojećeg sadržaja ili popratnu sliku postojećeg sadržaja.



Slika 18 Use-case diagram Android aplikacije, izvor: izradio autor

Slika prikazuje dijagram korištenja nativne Android aplikacije namijenjene posjetiteljima kulturnih institucija. Unutar svoje ingerencije posjetitelji kulturnih institucija mogu pregledavati lokaciju kulturnih institucija, događaje koji se odvijaju na lokalitetu koji posjećuju, također mogu pregledavati dosadašnja postignuća unutar aplikacije te sudjelovati na natjecanju unutar same aplikacije. Odabirom opcije za sudjelovanjem u natjecanju korisnik pokreće sudjelovanje te mu se prikazuju sadržaji povezani s institucijom u kojoj se odvija natjecanje.



Slika 19 Ilustrirani prikaz HistriApp sustava, izvor: izradio autor

Slika donosi ilustrirani prikaz kompozicije arhitekture samog sustava te tehnologije korištene za svaki segment sustava. Dakle, za web sučelje korišteni su primarno CSS

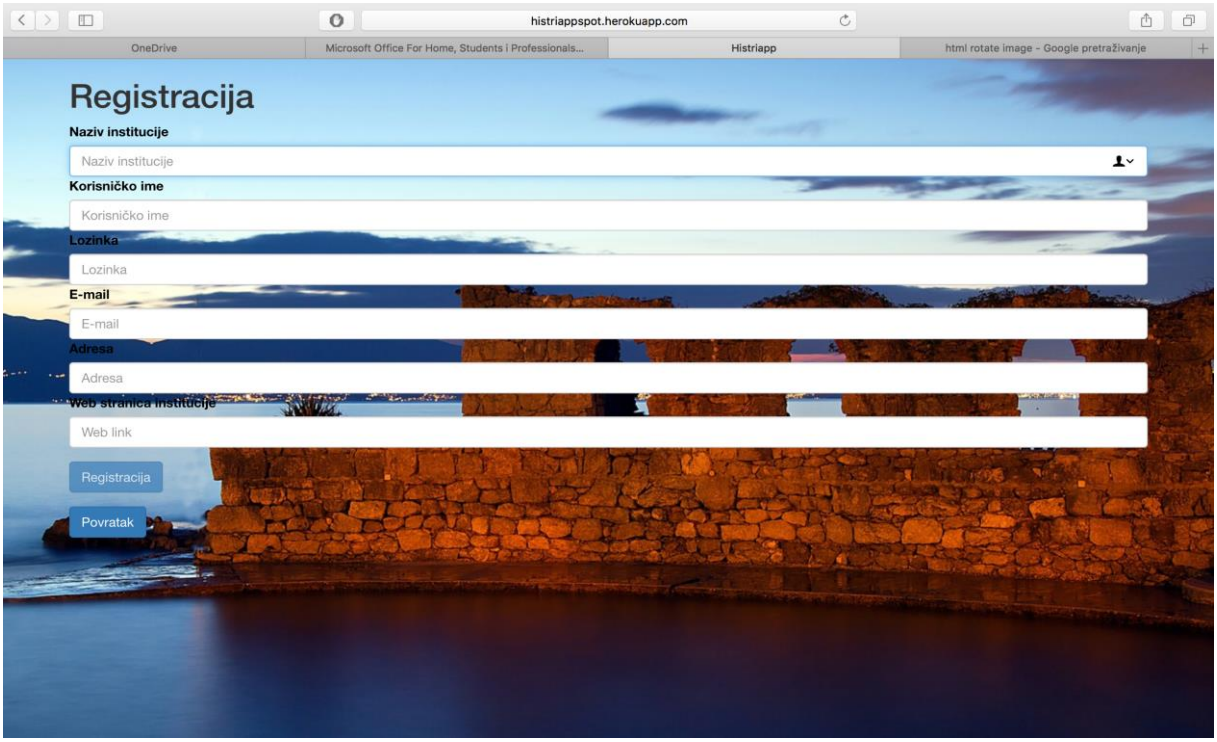
i HTMl za prikaz samih podataka korisniku te AngularJS kao radni okvir unutar Javascripta za manipuliranje samih podataka koji će biti prikazani korisniku. Kao backend, odnosno dio sustava zadužen za manipuliranje podacima, korišten je NodeJs razvojni okvir za razvijanje backend strane samog sustava. NodeJS nudi veliku slobodu pri manipuliranjem backend podatcima i rutama. Kod dijela sustava za korisnike; za prikaz podataka na nativnoj Android aplikaciji korištena je Java i XML te se pozivanje na same podatke koristilo pozivanjem NodeJs skripte. Dakle, za cjeloukupni backend sustava, kako Android tako i web administratorskog sučelja, korišten je NodeJS. Za manipuliranje slikama korišten je Google Cloud VISION API, koji je pozvan putem NodeJS skripte. Također, možemo reći da NodeJS služi i kao svojevrsni frontend, budući da je kod samog obrađivanja slike jedina njegova zadaća slanje parametra Cloud VISION API-ju te primanje odgovora s navedenog algoritma.

Sama NodeJS backend skripta poslužuje se na Heroku servisu. Heroku servis predstavlja jedan od najpopularnijih servisa upravo za mlade startupove, budući da se kod Heroku servisa, naplaćuje po korištenju aplikacije. Pri stavljanju sustava na server procjenjuje se veličina sustava te se sustavu dodjeljuju posebne jedinice koje se zovu dynos, a one se kasnije pretvaraju u novčane jedinice.

3.2 Razvoj Web sučelja

Kao što je prethodno naglašeno, HistriApp sustav je podijeljen na web i na Android aplikaciju. U procesu razvijanja web administratorskog sučelja korišten je takozvani single page application obrazac. Korištenjem takvog obrasca cjelokupna web stranica sadrži samo jednu stranicu dok se kroz aplikaciju mijenjaju podstranice. Dakle, takav obrazac ima početnu indeks stranicu koja je napisana u HTML programskom jeziku, unutar koje se učitavaju sve JavaScript skripte koje su korištene unutar same aplikacije. Pri kasnijem navigiranju kroz stranicu, mijenjanjem navigacije dolazi do promjene podstranica. Svakom novom navigacijom se ne poziva nova HTML datoteka, nego se unutar HTML elementa, kao izvorni kod tog elementa, poziva HTML skripta. Web sučelje također podržava pregled na više internet preglednika te na više uređaja – i na osobnom računalu i na mobilnom uređaju. Kompatibilnost s više uređaja postignuta je korištenjem responzivnog dizajna koji omogućava kreiranje elemenata na jednoj stranici koji se mogu prikazati na više različitih uređaja. Za razvijanje responzivnog dizajna korišten je Bootstrap radni okvir koji omogućava korištenje responzivnih elemenata unutar formatiranja web sadržaja.

Cijelo web sučelje ne sadrži puno grafičkih detalja budući da je cilj da osnovna verzija MVP-a s osnovnim funkcionalnostima bude što jednostavnija za korištenje djelatnicima partnerskih institucija. Dakle sama web stranica, kao što je napomenuto, sastoji se od forme za prijavu postojećeg korisnika, forme za registraciju novog korisnika te tablice s prikazom sadržaja unutar institucije. Svi elementi grafičkog sučelja s kojima korisnik ima mogućnost komunicirati, npr. elementi na stranici kao što su tekstna polja, povezani su s AngularJS skriptom koja zatim ima mogućnost obrađivanja teksta koji je korisnik unio u tekstno polje.



Slika 20. Izgled sučelja web platforme, izvor: izradio autor

Slika prikazuje web sučelje u Safari Web pregledniku na osobnom računalu. Važno je za napomenuti da se elementi stranice automatski prilagođavaju web stranici korištenjem bootstrap radnog okvira. Isti obrazac slaganja elemenata korisničkog sučelja je prisutan tijekom cijele stranice. Kao što je već navedeno, cijelo web sučelje sastoji se na temelju single page aplikacije. Dakle, potrebne skripte za upravljanje web sučeljem učitane su na index stranici. HistriApp web koristi skripte: routes.js", "controller.js", "service.js", "constants.js". Početna skripta koja se poziva zove se „routes.js" te je zadužena za navigiranje unutar web stranice. Pri pozivu stranice,

skripta routes.js prvotno provjerava je li korisnik prijavljen na stranicu. Ako je prijavljen, web sučelje se preusmjerava na unutarnji administratorski panel.

```
$stateProvider
.state('outside',{
   url: '/outside',
   abstract: true,
   templateUrl: '/index.html'
})
.state('outside.login',{
   url: '/login',
   templateUrl: 'subviews/login.html',
   controller: 'LoginCtrl'
})
.state('register',{
   url: '/register',
   templateUrl: 'subviews/register.html',
   controller: 'RegisterCtrl'
})
.state('inside',{
   url: '/inside',
   templateUrl: 'subviews/insert_treasure.html',
   controller: 'InsideCtrl'
});
```

Slika 21. Primjer konfiguriranja ruta u AngularJS-u, izvor: izradio autor

Primjer koda prikazuje konfiguranje ruta za cijelo web sučelje te preusmjeravanje na odgovarajuće HTML datoteke. Sljedeća Javascript skripta zove se “controller.js”, koja je zadužena za izravno komuniciranje s korisničkim sučeljem te prikazom podataka na istom. Elementi unutar HTML-a povezani su s “controller.js” skriptom kako bi mogli direktno komunicirati.

```
<tr ng-repeat="treasure in treasurelist">
        <td style="color:black; font-weight:bold;">{{treasure.riddle}}  </td>
```

Slika 22. Primjer spajanja HTML komponenti s AngularJS skriptom, izvor: izradio autor

Primjer koda prikazuje spajanje HTML komponenta s AngularJS skriptom. Važno je napomenuti da je sva logika odvojena od “controller.js” skripte kako ne bi bila izravno povezana s korisničkim sučeljem. U tom smislu sva logika komuniciranja s backend dijelom web sučelja odvija se na sljedećoj skripti po redu a to je “service.js” u kojoj su definirani glavni servisi koji se odvijaju na administratorskom panelu. Ti servisi mogu biti dodavanje novog sadržaja, brisanje starog sadržaja iz baze podataka, prijava korisnika ili registracija korisnika.

```
    $http.post(API_ENDPOINT.url+'/insertrow', fd, {
                transformRequest:angular.identity,
                 headers: {'Content-Type': undefined,
                 'Process-Data': false
                                                }
                                    })
                .then(function(response){
            if(response.data.msg=='Success'){
            resolve(response.data.msg);
          }else {
            console.log("wadap");
            reject('Failure');
          }


     }, function (response) {
       console.log(response);
       // something went wrong
       reject('Failure');
   });
//  });
```

Slika 23. Primjer korištenja servisa unutar AngularJS skripte, izvor: izradio autor

Primjer koda prikazuje korištenje servisa unutar “service.js” skripte. Specifičnost ove skripte je da je zadužena za dodavanje novog sadržaja u sustav. Ovaj servis poziva “post” metodu, a kao parametar mu dodaje put do skripte koja je spremljena u

konstantoj datoteci te mu dodaje podatke iz forme koji se spremaju na backend strani u sustav. Zadnja skripta frontend dijela web sučelja je “constant.js” – unutar te datoteke spremljene su globalne varijable koje se koriste kroz cijelo web sučelje.

3.3 Backend

Cjeloukupni Backend dio sustava HistriApp platforme počiva na NodeJS Javascript razvojnom okruženju. Kod backend sustava korištena je skripta naziva “app.js” koja je zadužena za obrađivanje različitih zadataka koje prima putem svojih unaprijed definiranih ruta. Skripta je cijelo vrijeme aktivna na Heroku platformi. Skripta sluša dolazeći promet na portu 8080 te na njemu adekvatno odgovara na pozvane funkcije.

```
app.use(function (req, res, next) {
    // Website you wish to allow to connect
    res.setHeader('Access-Control-Allow-Origin', '*');
    // Request methods you wish to allow
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
    // Request headers you wish to allow
   res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');

    // Set to true if you need the website to include cookies in the requests sent
    // to the API (e.g. in case you use sessions)
    res.setHeader('Access-Control-Allow-Credentials', true);
    // Pass to next layer of middleware
    if ( req.method == 'OPTIONS') {
        res.send(200);
    }
    else {
        next();
    }
                                });
```

Slika 24 Konfiguriranje backend skripte, izvor: izradio autor

Slika prikazuje početno definiranje i konfiguriranje backend skripte, napisane u NodeJs Javascriptu. Dakle, prvotno se definira zaglavlje stranice, gdje se specificira s kojih domena je dopušten ulaz na skriptu. Nadalje su također u zaglavlju definirane metode koje su dopuštene za izvršavanje na samoj skripti. Dopuštene metode definirane su kao:

GET - za dohvaćanje podataka sa poslužitelja

POST- za dohvaćanje skripte uz pridodane parametre u samom pozivu metode, ti parametri se kasnije mogu koristiti u samoj metodi

OPTIONS - metoda koja se poziva ukoliko poslužitelj nije u potpunosti siguran da će uspjeti pozvati POST metodu

PUT- metoda za umetanje parametra iz poziva funkcija u poslužitelju

DELETE - metoda za brisanje određenog sadržaja

Iznimno je važno naglasiti da nije dozvoljeno pristupanje određenim funkcijama unutar backend skripte ako se korisnik nije prijavio. To se postiglo korištenjem jedinstvenog tokena izgeneriranog pomoću spremljenog šifrata unutar sustava. Token je spremljen u zaglavlju. Akreditacija je dozvoljena pozivom funkcije zaglavlja "res.setHeader('Access-Control-Allow-Credentials', true);". Dakle korisnikova lozinka se šifrira korištenjem posebnog šifrata, a dešifrira također istim šifratom, ukoliko token nije validan, također neće biti dozvoljen pristup određenim funkcijama unutar backend skripte.

```
app.post('/api/insertrow',upload.any(), passport.authenticate('jwt',{session : false}),function(req, res, next) {

      if(req.body.image==null||req.body.riddle==null){
        return res.status(400).send({success:false, msg: 'Bad request'});
      }
      const signedUrlExpireSeconds = 60 * 5

var url_field = 'https://s3.eu-central-1.amazonaws.com/histriapp-assets/' + config.img_name;
var url_for_db = config.img_name;

const reqImg = new vision1.Request({
  //image: new vision1.Image('https://s3.eu-central-1.amazonaws.com/histriapp-assets/10%3A25%3A28ic_directions_black_2
  image: new vision1.Image({
      url: url_field
    }),
features: [
new vision1.Feature('LABEL_DETECTION', 10),
]
})
```

Slika 25 Primjer unosa novog sadržaja, izvor: izradio autor

Slika prikazuje kod pozivanja skripte za unos novog sadržaja. Metoda kao parametre prima:

1.) URL rutu na koju se odaziva - sama URL ruta pozvana je od strane frontend AngularJS servisa.

2.) Metodu za prijenos slike na server - „any" metoda predstavlja metodu iz „mutils-upload" radnog okvira unutar AngularJS-a, koja je zaslužna za prebacivanje sadržaja na server.

3.) Metoda za provjeru autentifikacije korisnika - „authenticate“ metoda iz „passport“ radnog okvira omogućava izvlačenje osjetljivih informacija, kao što su korisničko ime i lozika iz zaglavlja zahtjeva, te ih ima mogućnost dešifrirati koristeći tajnu riječ. Tako dekriptiran token iz zaglavlja zahtjeva se zatim referencira s podatcima iz baze podataka kako bi se ustvrdilo ima li korisnik pravo pristupa tom dijelu web mjesta.

Dakle prvo se pregledava validnost poslanog zahtjeva na server te, ako ulazni podatci nisu validni, korisnik se izbacuje. Također, unutar parametra funkcija poziva se metoda “upload.any()”, to je određena metoda unutar Express radnog okvira za NodeJS. Ta metoda prenosi sliku koja je postavljena unutar parametra. Slike se spremaju na Amazon servisu. Amazon nudi robustan i siguran način prijenosa sadržaja na svoje servere putem svoje s3 platforme. Ukoliko je slika prenesena na Amazon server te ako je zahtjev validan, pokreće se pozivanje Google Cloud VIsion Api-a, koji nudi svoj poseban radni okvir za NodeJs. Dakle, za pozivanje Google Cloud VIsion API-ja prvotno se stvara zahtjev, koji kao svoje parametra prima url do spremljene slike na Amazon servisu, te tip detekcije koji se planira izvršiti nad slikom. Taj dio predstavlja postavljanje Vision Api poziva. U nastavku je prikazana obrada rezultata iz Vision Api poziva.

```
vision1.annotate(reqImg).then((res1) => {
 var detectionarray = "";
 if(res1.responses[0].labelAnnotations.length==0){
   return res.status(400).send({success:true, msg: 'Failed'});
 }
   for (var i=0;i<res1.responses[0].labelAnnotations.length;i++){
     var score = parseFloat(res1.responses[0].labelAnnotations[i].score) * 100;
     if(score>60){
       detectionarray+=res1.responses[0].labelAnnotations[i].description;
       if(i!=res1.responses[0].labelAnnotations.length-1){
         detectionarray+=",";
       }
     }

   }

 var token = getToken(req.headers);
 if(token){
   var decoded = jwt.decode(token, config.secret);
 }
 var inst_name = decoded.name;
```

Slika 26 Pozivanje VISION API-a, izvor: izradio autor

Slika prikazuje pozivanje VISION API algoritma. VISION API algoritam se poziva korištenjem metode annotate iz vision1 varijable. Varijabla vision1 predstavlja instancu

AngularJS library-a naziva „google-vision-api“, to je službeni API od Google-a za upravljanje zahtjevima prema VISION API algoritmu.

Pri pozivanju kao parametar se proslijeđuje prethodno definirani zahtjev. Prvotno se provjerava validnost samog zahtjeva. Ukoliko algoritam nije uspio u pronalasku detekcija na slici vraća se povratni status 400. Nadalje se poziva petlja koja se izršava kroz polje detekcija. Takve detekcije se pretvaraju iz polja u jedan skup znakova koji se kasnije spremaju u bazu podataka. Također se osjetiljivi podatci o korisniku izvade iz zaglavlja tokena, dekodiranjem šifrata.

```
  db.histriMongoDb.insert({"riddle":riddleDesc,"detections": detectionarray,"img_url":
   url_for_db,"inst_name": inst_name}, function(err, doc) {

    if(err){
      return res.status(400).send({success:true, msg: 'Failed'});

    }else{

  //  res.json(doc);
  return res.status(200).send({success:true, msg: 'Success'});
}

});
  // handling response
}, (e) => {
  console.log('Error: ', e)
  //res.end('Cloud Vision Error');
//  res.end(e);
  return res.status(400).send({success:false, msg: 'Failure'});
  console.log(e);
})
```

Slika 27 Spremanje novog sadržaja u bazu podataka, izvor: izradio autor

Slika prikazuje spremanje novog sadržaja u mongo objektnu bazu podataka. Podatci se spremaju pozivanjem insert funkcije iz mongoDb librarya za NodeJS. Dakle pri pozivu funkcije „insert“ kao parametri funkcije pozivaju se polje kreirano u JSON formatu te funkcija koja je zadužena za hvatanje grešaka u kodu. Međutim, kao što je napomenuto, podatci se u bazu podataka spremaju u posebnom JSON formatu, a kao parametre uzima se naziv saržaja, detekcije nad slikom povezanog s upisanim nazivom sadržaja, url do uslikane slike (koja je spremljena na s3 amazon serverima), te naziv institucije. Ukoliko je došlo do pogreške u procesiranju zahtjeva, korisniku se vraća povratni status 400 te se korisničko sučelje, ili frontend, obavještava kako

zahtjev nije uspješno procesuiran. No ukoliko je sve prošlo u redu, vraća se povratni status 200, te se obaviještava korisnik kako je zahtjev uspješno procesiran.

Uz backend za web administratorsko sučelje, skripta također opslužuje i Android aplikaciju, pozivanjem na prikladne rute. Tako unutar svoje ingerencije za Android aplikaciju NodeJS skripta vraća sadržaj povezan s partnerskom institucijom. Primjer ospluživanja Android aplikacije korištenjem NodeJS skripte je prikazano u nastavku. Ta skripta se poziva kada je pozvan put prema ruti definiranoj u prvom parametru funkcije, a poziva se s Android aplikacije.

```
app.post('/api/androidbackend',function (req, res) {

  console.log('I received a GET request');

  var institution_name = req.body.user_name;
  console.log(req.body);
  db.histriMongoDb.find({inst_name: institution_name},function (err, docs) {
    if(err){
    //  console.log(err);
      res.json(err);
      return res.status(400).send({success:false, msg: 'Bad request'});

    }else{
    console.log(docs);
     res.json(docs);
    // return res.status(200).send({success:true, msg: 'Success'});
    }
    //res.json(docs);
  });
});
```

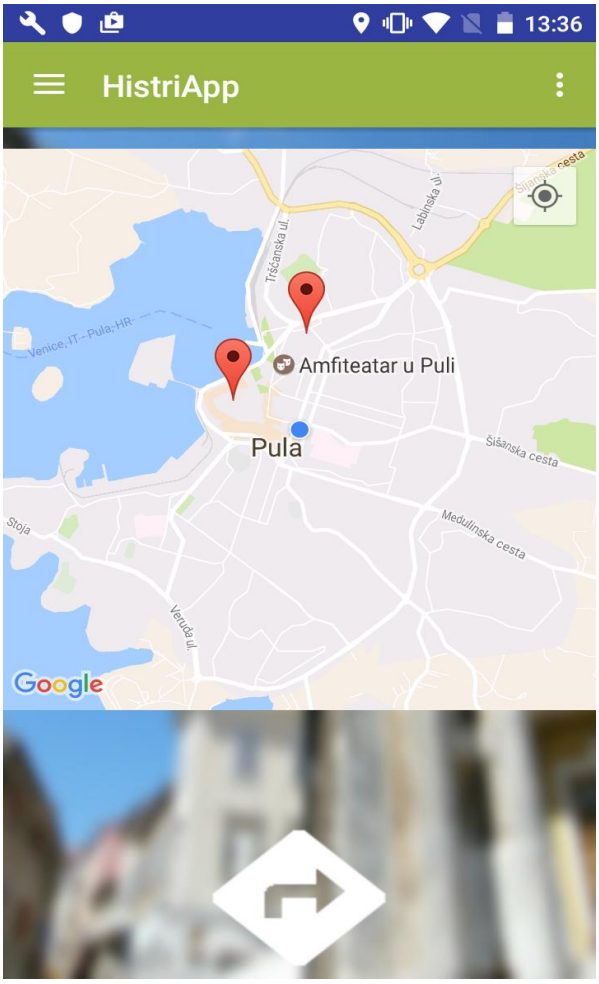
Slika 28 Pozivanje backend skripte za Android aplikaciju, izvor: izradio autor

Slika prikazuje pozivanje backend skripte za Android aplikaciju. Post metoda kao parametre uzima rutu na kojoj se backend skripta odaziva, ta je ruta pozvana na Android kada se prilikom sinkronizacije podataka kao URL pošalje ruta iz parametra. Sama skripta prvotno uzima naziv sadržaja iz zahtjeva, te uspoređuje s bazom podataka kako bi pronašla sav sadržaj referenciran s tom bazom podataka, te kao povratnu informaciju vraća JSON format sa svim sadržajima koji su povezani s promatranom institucijom. Podatci se uspoređuju s bazom podataka koristeći funkcije „find“ iz MongoDb instance. Funkcija „find“ može se promatrati kao SQL upit na bazu, no kod MongoDB baze je slučaj drugačiji budući da nije relacijska baza podataka. MongoDB baza traži zapis u JSON polju koji odgovaraju pruženom nazivu institucije.

Dakle funkcija „find“ kao parametre uzima JSON polje koje mora pronaći u pruženom entitetu, u ovom slučaju je to histriMongoDB. U ovom konkretnom slučaju kao JSON polje pruža se naziv institucije koje se traži. Pronalaskom traženog naziva institucije, korisniku se vraćaju svi parametri koji odgovaraju pruženoj instituciji.

3.4 Izrada Android aplikacije

Pri izradi Android aplikacije koristio se Android studio razvojno okruženje. Programski kod izrađen je u Java programskom jeziku, dok je sučelje aplikacije izrađeno u XML jeziku. Početna aktivnost Android aplikacije započinje početnim ekranom unutar kojega se sinkroniziraju sve dostupne institucije na promatranom lokalitetu. Sinkroniziranjem podataka korisnik se preusmjerava na početnu aktivnost aplikacije.



*Slika 29 **Početna** aktivnost Android aplikacije, izvor: izradio autor*

Slika prikazuje glavnu aktivnost Android aplikacije. Unutar te aktivnosti korisniku je prikazana mapa lokaliteta s označenim partnerskim institucijama unutar kojih korisnik može istraživati dodane sadržaje. Mapa lokaliteta je prikazana koristeći Google Maps API za obrađivanje geografskih karata. Lokaliteti markera partnerskih institucija dobiveni su putem web platforme. Pri registraciji korisnika na web platformu korisnik mora unijeti adresu partnerske institucije koja se zatim pretvara u geografsku širinu i duljinu korištenjem Javascrpita te tako pretvorena adresa se zatim sprema u MongoDB bazu podataka. Prilikom pokretanja aplikacije podatci iz baze za institucije s sinkroniziraju s lokalnom SQLite bazom podataka. Odabirom jedne od partnerskih institucija korisnik može odabrati navigacijsku rutu do partnerske institucije ili može direktno pristupiti istraživanju sadržaja. Sama početna aktivnost u sebi sadrži Broadcastlistener. BroadcastListener instanca služi za osluškivanje promjene korisnikove pozicije putem Google Locations API-a. Google Locations API služi za otkrivanje korisničke lokacije korištenjem GPS servisa. Kako bi se mogle koristiti Google-ovi API-ji potrebno je na Google razvojnim stranicama zatražiti API ključ koji aplikaciji daje potrebnu autentifikaciju za korištenjem API-ja. Android aplikacija dodatno koristi i FusedLocations, kao dio Google Locations API-ja kako bi se što efikasnije pronašla korisnička lokacija. Dakle, FusedLocations koristi sve resurse od korisničkog mobilnog uređaja, kao što su internet mreža, SIM kartica te sam GPS lokator kako bi što efikasnije pronašli korisničku lokaciju. Kako bi FusedLocations što kvalitetnije pronašao lokaciju prima tri parametra:

1.) Razmak - promatra na kojoj razdaljini se korisnička lokacija treba tražiti - to može biti na razini ulice, na razini grada te na razini države.

2.) Vremenski interval - promatra u kojem vremenskom intervalu treba pronaći korisničku lokaciju

3) Potrošnja baterije - promatra koliki postotak mobilnog uređaja treba iskoristiti kako bi se pružila što kvalitetnija usluga, tako primjerice može koristiti nisku potrošnju baterije – gdje se uzima u obzir samo GPS lokator, srednju potrošnju bateriju – gdje se uzima u obzir GPS lokator te SIM kartice ili maksimalna potrošnja baterije – gdje se iskorištavaju svi dostupni lokatori na uređaju.

```
@Override
protected void onStop() {
    super.onStop();
    locationService.stopLocationUpdates();
    // unregister observer
    LocalBroadcastManager.getInstance(MainSelectionAct.this).unregisterReceiver(mLocationUpdated);
    mLocationUpdated.clearAbortBroadcast();

}

@Override
protected void onResume() {
    super.onResume();
    LocalBroadcastManager.getInstance(MainSelectionAct.this).registerReceiver(mLocationUpdated,
            new IntentFilter(INTENT_FILTER_LOCATION_UPDATE));
    locationService = new LocationService(this);
    locationService.startLocationUpdates();

}
```

Slika 30 Stvaranje instance BroadcastReceiver-a, izvor: izradio autor

Slika prikazuje stvaranje instance BroadsastRecievera koji se stvara pri vraćanju aktivnosti na vrh memorijskog stoga i micanje s vrha memorijskog stoga unutar onStop metode. Dakle, prilikom prestanka aktivnosti poziva se metoda Android-ove klase Activity naziva „onStop“. Unutar te metode se poziva metoda iz klase LocationService koja zaustavlja ažuriranje korisničke lokacije.

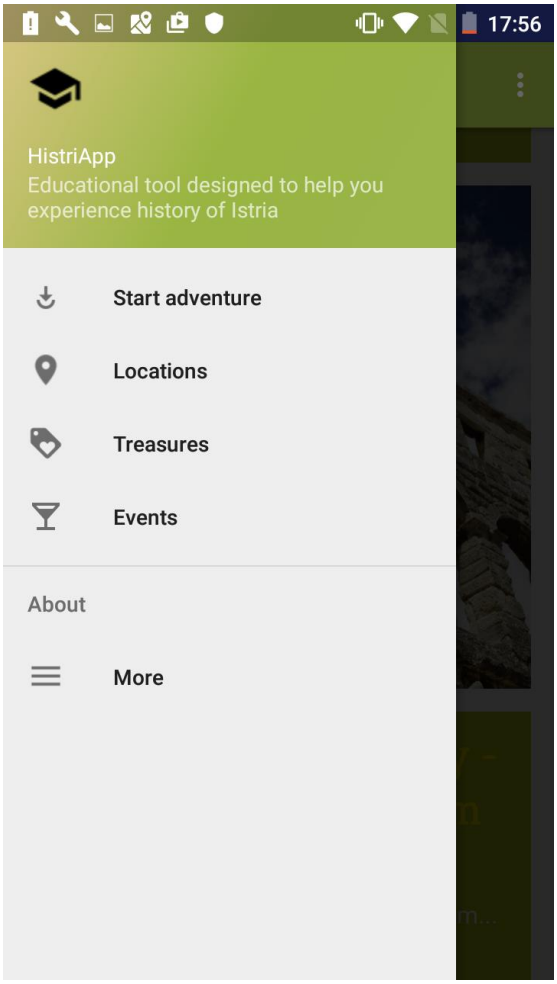
```
private BroadcastReceiver mLocationUpdated = (context, intent) → {
        Location location = intent.getParcelableExtra(LBM_EVENT_LOCATION_UPDATE);
        locationForZoom = location;
        locationForRoute = location;

            updateCameraBearing(map,location.getBearing());
        progressDialog.dismiss();
            if (map != null) {
                if(dirsOn){
                    CameraPosition cameraPosition = new CameraPosition.Builder()
                            .target(new LatLng(location.getLatitude(),
                                    location.getLongitude())).zoom(zoomFactor).build();
                    CameraUpdate cameraUpdate = CameraUpdateFactory
                            .newCameraPosition(cameraPosition);
                    map.moveCamera(cameraUpdate);
                }else {
                    CameraPosition cameraPosition = new CameraPosition.Builder()
                            .target(new LatLng(location.getLatitude(),
                                    location.getLongitude())).zoom(13).build();
                    CameraUpdate cameraUpdate = CameraUpdateFactory
                            .newCameraPosition(cameraPosition);
                    map.moveCamera(cameraUpdate);
                }
            }


};
```

Slika 31 Primjer upravljanja lokacijskim podatcima, izvor: izradio autor

Slika prikazuje upravljanje primljenim lokacijskim podatcima iz BroadcastReceivera. Prvotno se podatci o samoj lokaciji dobivaju instanciranjem klase Location. Nadalje se

poziva metoda updateCameraBearing koja je odgovorna za ažuriranje prikaza karte, ovisno o trenutnoj korisnikovoj lokaciji. Također se, ukoliko je uključena opcija navigacije, korisniku zumira karta na njegovu lokaciju.

Glavna navigacija kroz aplikaciju odvija se putem NavigationDrawer nativne Android komponente. NavigationDrawer sadrži jednu klasu koja je nazvana BaseActivity, unutar koje se upravlja odabirom drugih opcija. Sve aktivnosti unutar aplikacije nasljeđuju gore navedenu BaseActivity klasu.



Slika 32 Izgled NavigationDrawer komponente, izvor: izradio autor

Slika prikazuje glavnu aktivnost s uključenim NavigationDrawer-om. Korisnik može odabrati NavigationDrawer prijelazom s lijeve strane ekrana ili odabirom ikonice s gornje lijeve strane. Glavne aktivnosti koje korisnik može odabrati su:

- “Start adventure” - korisnik se odabirom opcije preusmjerava na glavni ekran aplikacije, unutar koje može birati institucije čiji sadržaj želi razgledavati i sudjelovati u natjecanju.
- “Locations” - korisnik se odabirom opcije preusmjerava na ekran s lokacijama partnerskih institucija i drugih znamenitosti u gradu Puli.
- “Treasures” - korisnik se odabirom opcije preusmjerava na ekran unutar kojega može pregledati sve sadržaje koje je već riješio te može podijeliti svoja dostignuća sa svojim prijateljima putem društvenih mreža.
- “Events” - korisnik se odabirom opcije preusmjerava na ekran unutar kojega može pregledavati događaje i zanimljivosti koja se odvijaju u gradu. Trenutno se korisnik preusmjerava na stranice Turističke zajednice grada Pule.
- “More” - korisnik se odabirom opcije preusmjerava na ekran unutar kojega može saznati više o povijesti grada.

U nastavku će biti detaljno objašnjena svaka od glavnih aktivnosti pojašnjavanjem ključnih dijelova koda vezanih za aktivnosti. Odabirom jedne od partnerskih institucija te nastavljanjem na sudjelovanje u razgledavanju sadržaja poziva se servis za preuzimanje sadržaja odabranih institucija, naziva “syncAdventureAndProceed”, koja kao parametre prima kontekst aplikacije te korisničko ime institucije za koju se sadržaj treba preuzeti.

```
public static void syncAdventureAndProceed(final Context context,String user_name){
    final DatabaseHelper db = new DatabaseHelper(context);
    final Intent intent = new Intent(context,QuizActivity.class);
    JSONObject jsonBodyObj = new JSONObject();
    try{
       jsonBodyObj.put("user_name", user_name);
    }catch (JSONException e){
     //  Intent intent1 = new Intent(context, MainSelectionAct.class);
      //  CustomAlertDialog.showErrorAlert(context,intent1,"There was an error,check your internet connection and try again later");
       e.printStackTrace();
    }
```

```
final String requestBody = jsonBodyObj.toString();
JsonArrayRequest JOPR = new JsonArrayRequest(Request.Method.POST,
    BuildConfig.URL_ENDPOINT_ALL_CLUES, null, new
Response.Listener<JSONArray>(){
    @Override
    public void onResponse(JSONArray response) {
      try {
    String dropCategoriesSQL = "DROP TABLE IF EXISTS "+
     DatabaseHelper.databaseTable.ADVENTURES.table;
        db.getWritableDatabase().execSQL(dropCategoriesSQL);
        String createAdventureTable = "CREATE TABLE "+
    DatabaseHelper.databaseTable.ADVENTURES.table+
          "("+ DatabaseHelper.adventureColumn.ID.column+" INTEGER PRIMARY
KEY AUTOINCREMENT, "+
          DatabaseHelper.adventureColumn.RIDDLE.column+" VARCHAR, "+
          DatabaseHelper.adventureColumn.DESC.column+" VARCHAR);";
           db.getWritableDatabase().execSQL(createAdventureTable)
```

Slika 33. Primjer kreiranja "syncAdventureAndProceed" metode, izvor: izradio autor

U ovom dijelu koda prikazano je kreiranje iznad objašnjenog servisa syncAdventureAndProceed. U prvom koraku se kreira instanca klase DatabaseHelper, koja predstavlja klasu koja upravlja SQLite bazom podataka. Zatim je kreirana instanca tipa JSONObjectBody, koja je odgovorna za kreiranje zahtjeva u obliku JSON-a koji se kasnije obrađuje u NodeJS skripti. Nadalje se u JSONObjectBody stavlja varijabla tipa String “user_name” koja predstavlja korisničko ime institucije koje služi za upite nad MongoDB bazom unutar NodeJS skripte. Potom se kreira instanca tipa JsonArrayRequest koja kreira mrežni zahtjev u obliku JSON notacije. Kao parametri pri kreiranju zahtjeva uzima se tip metode, u ovom slučaju je to post metoda, url koji se poziva, u ovom slučaju je to url na NodeJS rutu te odgovor na zahtjev koji se dobiva u obliku JSON polja. Dakako, pri dobivenom odgovoru prvotno se izbriše stara verzija zapisa u SQLite tablici, te se kreira nova SQLite tablica koja lokalno sprema dobivene sadržaje iz partnerske institucije.

```
if(response.length()==0){
    Intent intent1 = new Intent(context,MainSelectionAct.class);
```

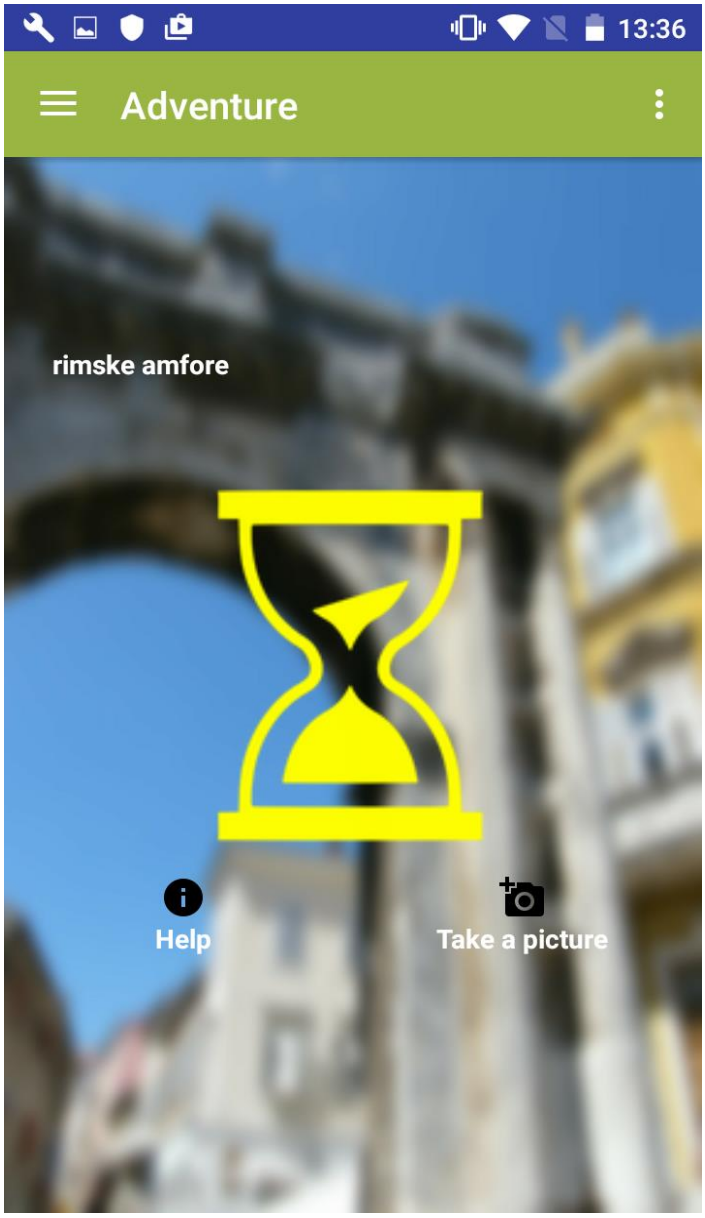
```
                    CustomAlertDialog.showErrorAlert(context,intent1,"Unfornatenantly there are
no clues at this                    location");
                }else {
                    for (int i = 0; i < response.length(); i++) {
                        JSONObject jsonObject1 = response.getJSONObject(i);
                        riddle = jsonObject1.getString("riddle");
                        detections = jsonObject1.getString("detections");
                        db.addAdventure(riddle, detections);
                        if (i + 1 == response.length()) {
                            context.startActivity(intent);
                            ((Activity) context).finish();
                        }
                    }
                }
            } catch (JSONException e) {
                Intent intent1 = new Intent(context,MainSelectionAct.class);
                CustomAlertDialog.showErrorAlert(context,intent1,"There was an error, check
your internet connectivity and try again");
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            VolleyLog.e("Error: ", error.getMessage());
            Intent intent1 = new Intent(context,MainSelectionAct.class);
            CustomAlertDialog.showErrorAlert(context,intent1,"There was an error, check your
internet connectivity and try again");
        }
    }){
        @Override
        public Map<String, String> getHeaders() throws AuthFailureError {
            HashMap<String, String> headers = new HashMap<String, String>();
            headers.put("Content-Type", "application/json");
```

```
    return headers;
        }
        @Override
        public byte[] getBody() {
            try {
                return requestBody == null ? null : requestBody.getBytes("utf-8");
            } catch (UnsupportedEncodingException uee) {
                    requestBody, "utf-8");
                return null;
            }
        }
    };
    RequestApp requestApp = new RequestApp();
    requestApp.setContext(context);
    requestApp.getInstance();
    requestApp.addToReqQueue(JOPR);
}
```

Slika 34. Primjer upravljanja odgovorom na JSON zahtjev, izvor: izradio autor

Ovaj dio koda prikazuje upravljanje odgovorom na JSON zahtjev; za obavljanje REST mrežnih poziva korišten je library Volley. Dakle prvotno se provjerava je li dobiven validan odgovor, a ako nije korisnika se obavještava da pokuša ponovno ili se u suprotnom pokreće for petlja koja prolazi kroz polje tipa JSON. Unutar te petlje izvlači se objekt tipa JSON na svakom prolasku polja te se poziva funkcija klase DatabaseHelper “addAdventure”. Ta funkcija kao svoje parametre prima naziv sadržaja koji korisnik treba pronaći na svom mobitelu, te rezultate nad slikom koju korisnik treba pronaći. Rezultati nad samom slikom su obrađeni na web administratorskom sučelju kao što je već navedeno te su spremljeni u MongoDB bazu podataka. Nadalje, kada je petlja na kraju poziva se nova aktivnost u kojoj korisnik može sudjelovati u “treasure huntu”. Metoda također ima implementirano nekoliko upravljanja greškama koje su se mogle dogoditi. Prvotno ima “listener” na odgovor od zahtjeva ukoliko nešto pođe po krivu pri samom procesiranju zahtjeva na backend strani; također su pozvane metode “getHeaders” koje postavljaju zaglavlje JSON upita. U to zaglavlje upita se proslijeđuje tip zahtjeva, a u ovom slučaju je to JSON zahtjev. Treća metoda koja je pozvana iz Volley librarya je “getBody” metoda.

Unutar te metode se postavlja tijelo JSON zahtjeva, koje je prethodno kreirano iz instance tipa JSONBodyObject. Na kraju se kreira instanca tipa RequestApp, koja predstavlja globalnu klasu unutar koje se Volley zahtjevi izvršavaju na pozadinskoj dretvi izvan korisničkog sučelja. Zahtjev se na kraju izvršava pozivanjem metode “addToReqQueue” iz klase RequestApp. RequestApp klasa predstavlja globalnu klasu unutar koje se postavlja kontekst trenutne aktivnosti, koji se zatim prosljeđuje u Volley library zadužen za mrežne pozive.



Slika 35 Aktivnost za pretraživanje sadržaja partnerske institucije, izvor: izradio autor

Slika prikazuje aktivnost za pretraživanje sadržaja kulturne institucije. Korisniku je prikazan naziv sadržaja koji mora pronaći. U doljnjem dijelu ekrana prikazana su dva gumba: za pomoć i za otvaranje kamere. Odabirom gumba za pomoć korisnik može odabrati da preskoči trenutni sadržaj te krene tražiti sljedeći sadržaj; a s druge strane odabirom gumba za otvaranje kamere otvara se kamera mobitela te korisnik može snimiti fotografiju za koju misli da odgovara nazivu sadržaja. Nakon snimanja fotografije vraća se u aktivnost te se poziva VISION API za procesuiranje snimljene fotografije. Ako snimljena fotografija odgovara traženom sadržaju, korisnik može nastaviti tražiti sljedeće sadržaje sve dok nije prešao sve sadržaje koji pripadaju partnerskoj instituciji. Nakon što je pregledao sav sadržaj partnerske institucije, korisnik se preusmjerava na aktivnost unutar koje može pregledati svoj rezultat te podijeliti ostvareni rezultat sa svojim prijateljima na društvenim mrežama. U prilogu koda ispod prikazana je Java metoda “callCloudVision”.

```
private void callCloudVision(final Bitmap bitmap, final TextView mImageDetails, final String comparison) throws IOException {
    // Switch text to loading
mImageDetails.setText(getResources().getString(R.string.process_text));
    // Do the real work in an async task, because we need to use the network anyway
    new AsyncTask<Object, Void, String>() {
      @Override
      protected String doInBackground(Object... params) {
        try {
          HttpTransport httpTransport = AndroidHttp.newCompatibleTransport();
          JsonFactory jsonFactory = GsonFactory.getDefaultInstance();
          Vision.Builder builder = new Vision.Builder(httpTransport, jsonFactory, null);
          builder.setVisionRequestInitializer(new
          VisionRequestInitializer(BuildConfig.VISION_API_KEY));
          Vision vision = builder.build();
          BatchAnnotateImagesRequest batchAnnotateImagesRequest =
              new BatchAnnotateImagesRequest();
          batchAnnotateImagesRequest.setRequests(new
ArrayList<AnnotateImageRequest>() {{
            AnnotateImageRequest annotateImageRequest= new AnnotateImageRequest();
```

```
                // Add the image
                Image base64EncodedImage = new Image();
                // Convert the bitmap to a JPEG
                // Just in case it's a format that Android understands but Cloud Vision
                ByteArrayOutputStreambyteArrayOutputStream=new
ByteArrayOutputStream();
                bitmap.compress(Bitmap.CompressFormat.JPEG,                    90,
byteArrayOutputStream);
                byte[] imageBytes = byteArrayOutputStream.toByteArray();
                // Base64 encode the JPEG
                base64EncodedImage.encodeContent(imageBytes);
                annotateImageRequest.setImage(base64EncodedImage);
                // add the features we want
                annotateImageRequest.setFeatures(new ArrayList<Feature>() {{
                    Feature labelDetection = new Feature();
                    labelDetection.setType("LABEL_DETECTION");
                    labelDetection.setMaxResults(10);
                    add(labelDetection);
                }});
                // Add the list of one thing to the request
                add(annotateImageRequest);
            }});
            Vision.Images.Annotate annotateRequest =
                    vision.images().annotate(batchAnnotateImagesRequest);
            // Due to a bug: requests to Vision API containing large images fail when GZipped.
            annotateRequest.setDisableGZipContent(true);
            BatchAnnotateImagesResponse response = annotateRequest.execute();
            return convertResponseToString(response);
        } catch (GoogleJsonResponseException e) {
            Log.d(TAG, "failed to make API request because " + e.getContent());
        } catch (IOException e) {
            Log.d(TAG, "failed to make API request because of other IOException " +
                    e.getMessage());
        }
```

```
        return "failed";
    }
```

Slika 36. Primjer deklaracije "callCloudVision" funkcije, izvor: izradio autor

Ovaj dio koda prikazuje deklaraciju void funkcije “callCloudVision” koja kao parametre prima sliku koju će obraditi u obliku tipa objekta Bitmap naziva “bitmap”, zatim prima objekt naziva “mImageDetails” tipa TextView koji prikazuje korisniku tekst zagonetke koja se traži, a naposljetku prima varijablu tipa String naziva “comparison”. Funkcija obavlja mrežne poziva pa stoga ima implementirano hvatanje I/O grešaka, dakle budući da se u funkciji obavljaju mrežni pozivi sama funkcija se izvodi asinkrono u pozadinskoj dretvi izvan korisničkog sučelja. U Javi se to također zove AsyncTask koji poziva dvije metode, prva je “doInBackground” koja se izršava izvan dretve korisničkog sučelja, a pri završetku metode poziva se metoda “onPostExecute” unutar koje se mogu obavljati izmjene na korisničkom sučelju. Dakle unutar “doInBackground” metode prvotno se tvori instanca klase VisionBuilder koja je zaslužna za kreiranje zahtjeva za VISION API u JSON notaciji. Sam VisionBuilder pri samom inicijaliziranju kao parametre u konstruktoru prima instance klase tipa HttpTransport i JsonFactory. VisionBuilder varijabla je zaslužna za kreiranje zahtjeva za Vision API a to ostvaruje pozivanjem „setVisionRequestInitializor“ metode. Nadalje se kreiraju dvije instance, jedna klase Vision i druga klase BatchAnnotateRequest. Varijabla vision se inicijalizira pozivanjem “build” metode iz VisionBuilder klase. Instanca klase BatchAnnotateRequest je zaslužna za kreiranje samog zahtjeva prema VISION API-ju pozivanjem “setRequest” metode. Zatim se kreira instanca tipa AnnotateImageRequest te instanca tipa Image te se tvori polje bitova koje sadrži sliku koja će se obraditi u algoritmu u vidu bitova. Nadalje se to polje dekodira u Image klasu te se takva instanca predaje kao parametar u metodi “setImage”, unutar AnnotateImageRequest klase. Nakon što su dodane slike u AnnotateImageRequest poziva se funkcija istoimene klase naziva “setFeatures” unutar koje se definiraju paramteri za VISION API koji bi trebao pretraživati. U ovom primjeru kreira se instanca klase tipa Feature, koja postavlja tip koji će se pretraživati, te je ovdje naveden “LABEL_DETECTION” što označava opće karakteristike slike. Uz “LABEL_DETECTION” postoje još i “FACE_DETECTION” za detekciju lica, “LANDMARK” za detekciju znamenitosti itd. Također, uz tip detekcije pridodan je i maksimalan broj rezultata koje algoritam mora pronaći na slici. Na kraju metode se kreira i instanca klase BatchAnnotateImagesResponse koja služi kao odgovor na gore

kreirani zahtjev iz VISION API-ja, koji se tvori pozivanjem metode “execute()” od instance tipa Vision.Images.Annotate. Na kraju “doInBackground metoda” vraća vrijednost odgovora u tipu String varijable.

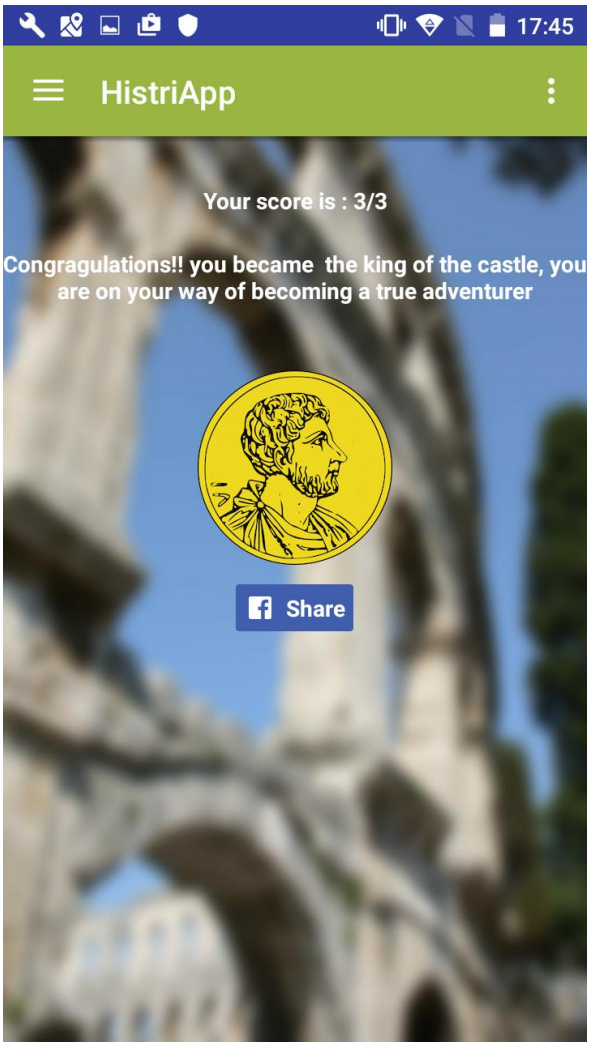
```
protected void onPostExecute(String result) {
    int counter = 0;
    progressBar.setVisibility(View.GONE);
    String[] resultArray = result.split(",");
    for(int i=0;i<resultArray.length;i++){
        Log.d("detect",resultArray[i]);
        if(!resultArray [i].equals("")) {
            String tempComp = resultArray[i];
            for (int j = 0; j < comparisonArray.length; j++) {
                if (comparisonArray[j].contains(tempComp)) {
                    counter++;
                }
            }
        }
    }
    double percentage=0
    percentage = Math.round((counter * 100.0) / resultsArray.length);
    if(result.equals("failed")){
        failure_sound.start();
        buildCompletionDialog(false,context,"Processing failed, please check your internet connection");
        mImageDetails.setText(arrayList.get(picture_inc).getRiddleDesc());
    }else
    if(percentage>=50) {
        success_sound.start();
        buildCompletionDialog(true,context,"Congratulations, you found the  clue");
    }else if(percentage>=10&&percentage<=30){
        failure_sound.start();
        buildCompletionDialog(false,context,"Close, maybe try with different angle?");
                mImageDetails.setText(arrayList.get(picture_inc).getRiddleDesc());
```

```
            }else {
                failure_sound.start();
                buildCompletionDialog(false,context,"Sorry wrong picture, you can proceed to next clue by clicking on info button");
                mImageDetails.setText(arrayList.get(picture_inc).getRiddleDesc());
            }
        }
    }.execute();
}
```

Slika 37. Primjer koda odgovornog za manipuliranjem odgovora na VISION API upit, izvor: izradio autor

Primjeru koda prikazuje dobiveni odgovor iz VISION API-a koji se dobiva kao rezultat funkcije “convertResponseToString”. Dakle, na početku metode se prvotno inicijalizira brojač tipa integer, koji služi za obrađivanje broja točno nađenih rezultata u slici. Nadalje se kreira polje Stringa koje nastaje tako što se dobiveni rezultat (koji je čisti String) prepolovi funkcijom “split” po znaku “,”. Nadalje se kroz to polje kreira for petlja. Unutar for petlje se kreira privremena varijabla tipa String koja preuzima vrijednost polja na brojaču. Zatim se kreira druga petlja koja prolazi kroz polje Stringa naziva “comparisonArray”. To predstavlja nađene rezultate za sliku koja se traži. Ti rezultati su prethodno obrađeni na web sučelju, te su sinkronizirani s Android aplikacijom. Rezultati s web sučelja su sinkronizirani lokalno na Android uređaju tako što su spremljeni u SQLite bazu podataka. Polje je “comparisonArray” rezultat upita nad SQLite bazom podataka koje vraća rezultate za sliku koja se trenutno traži. Dakle u drugoj petlji kroz “comparisonArray” se uspoređuje vrijednost nađenog rezultata na slici (koje je spremljeno u privremenoj varijabli) s vrijednošću rezultata nad slikom koja se traži. Ukoliko rezultat slike, spremljen u privremenoj varijabli, odgovara rezultatu spremljenom u polju “comparisonArray” brojač se povećava za jedan. Nakon petlje se kreira varijabla tipa double koja predstavlja postotak točnih pronalazaka. Postotak točnih pronalazaka se dobiva tako što se prethodno kreirani brojač točnih pronalazaka nad snimljenom fotografijom se pomnoži sa sto, te se podijeli s ukupnim brojem rezultata nad slikom. U nastavku se if petljama provjeravaju mogući scenariji, dakle ukoliko VISION API nije uspio pronaći nijedan rezultat vraća “failed” te se korisnik obavještava da ponovi postupak. Ako je postotak točnih pronalazaka jednak ili veći od pedeset posto poziva se metoda “buildCompletionDialog”. “BuildCompletionDialog”

metoda preusmjerava korisnika na sljedeću zagonetku ako je pronašao sliku ili ga zadržava na trenutnoj zagonetci ukoliko nije uspio pronaći sliku. “BuildCompletionDialog” metoda kao parametre prima varijablu tipa boolean koja ukoliko je vrijednost true preusmjerava korisnika na sljedeću zagonetku, a ako je vrijednost false zadržava korisnika na trenutnoj zagonetci. Nadalje ukoliko je postotak točno pronalaženih slika između deset i trideset i pet posto korisnika se zadržava na trenutnoj zagonetci te ga se obavijesti da pokuša fotografirati iz drugog kuta, budući da je VISION API našao određenu podudarnost. Naposljetku, ako je postotak ispod deset posto, korisnika se obavještava da je pronašao krivu fotografiju.



Slika 38 Aktivnost za pregled ostvarenog rezultata, izvor: izradio autor

Slika prikazuje aktivnost unutar koje korisnici mogu pregledati ostvareni rezultat unutar partnerske institucije. Također odabirom gumba “Share” korisnici mogu podijeliti svoje rezultate putem društvenih mreža. Funkcija dijeljenja sadržaja putem društvenih mreža

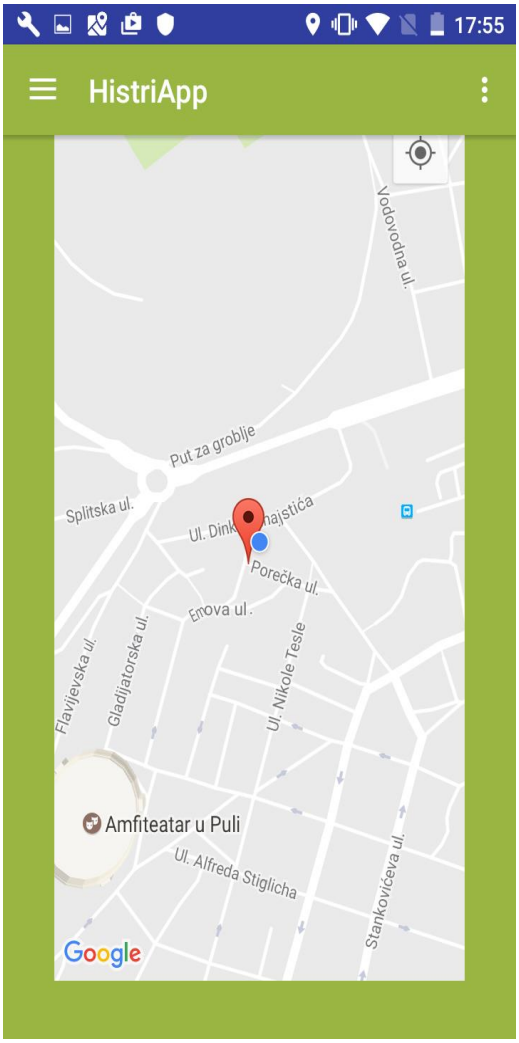
izvršava se pozivanjem Facebook API-ja za dijeljenje sadržaja. Kao mogućnost dodatnog proširenja funkcionalnosti aplikacije, ukoliko MVP bude prihvaćen od strane tržišta, razmatra se mogućnost uvođenja popusta lokalnim poslovnim subjektima, jer bi tako aktivnost izgledala znatno drukčije zato što bi pokazivala podatke o poslovnom subjektu unutar kojega se mogu iskoristiti popusti te upute na karti do poslovnog subjekta. Samo korištenje popusta moglo bi se ostvariti korištenjem QR koda koji bi se poslao posjetitelju koji je uspješno pregledao sve sadržaje unutar partnerske institucije. Svakako, razmatra se uvođenje sustava rangiranja korisnika tako da bi, ovisno o uspješnosti kojom je pregledao sadržaj unutar institucije, ostvarivao i pravo na popust.

Slika 39 Aktivnost za prikaz događaja u gradu, izvor: izradio autor

Slika prikazuje aktivnost za prikazivanje događaja koji se odvijaju na lokalitetu gdje se aplikacija izvodi, u ovom slučaju na širem području grada Pule. Cijela aktivnost je prikazana unutar WebView komponente koja omogućava prikaz web sadržaja unutar nativne aplikacije. WebView komponenta omogućava i učitavanje samog URL-a ili

kreiranje kustomizirane stranice koristeći web tehnologije kao što su css, Javascript ili HTML.

Trenutno za MVP verziju proizvoda, stranica koja se učitava jest stranice Turističke zajednice grada Pule, no za sljedeću verziju jedna od mogućnosti je razvijanje same web stranice unutar koje bi korisnici mogli pregledavati događaje ili potvrditi dolazak kako bi se napravila i određena komponenta socijalne mreže za dijeljenje među posjetiteljima, te kako bi određeni posjetitelji znali kakav tip posjetitelja posjećuje određene događaje. Također, ovisno o korištenosti aplikacije, moguće je prebacivanje fokusa same aplikacije s pretraživanja sadržaja unutar institucije na pretraživanje sadržaja unutar partnerske institucije na pretraživanje sadržaja lokaliteta.



Slika 40 Aktivnost za prikaz lokacija partnerskih institucija, izvor: izradio autor

Slika prikazuje korištenje Google karti za prikaz lokacija partnerskih institucija i prikaz korisničke lokacije. Karta je prikazana korištenjem Google Maps API-ja, dok je lokacija korisnika prikazana korištenjem Google locations API-ja. Lokacija partnerskih institucija dobivena je prilikom registracije novog korisnika na web sučelju. Korisnik pri registraciji upisuje adresu svoje institucije, a upisana adresa se zatim u AngularJS kodu pretvara u geografsku širinu i duljinu te se sprema u bazu podataka unutar NodeJS skripte. Prilikom pokretanja aplikacije podatci s baze podataka se sinkroniziraju na lokalnu bazu podataka na mobitelu, pa se s njima spreme i podatci o partnerskoj instituciji kao što su: lokacija, naziv institucije i url do mrežnog mjesta institucije. Korisnička lokacija se prati korištenjem instance BroadcastListener instance koja osluškuje korisničku lokaciju u intervalu od tri sekunde. U kontekstu budućeg razvoja, razmatra se uvođenje pregledavanja lokaliteta poslovnih subjekata te povijesnih znamenitosti.

3.5 Budući razvoj

U ovom diplomskom radu prikazan je razvoj verzije aplikacije s osnovnim funkcionalnostima, takozvani MVP (Minimum Viable product). Osnovna verzija aplikacije dakle ima mogućnost pretraživanja lokacija partnerskih institucija, posjećivanja partnerskih web mjesta, kao i mogućnost pregledavanja sadržaja koji je pregledao te njegovo dijeljenje na društvene mreže. Također, korisnik ima mogućnost pregledavanja osnovne funkcionalnosti a to je pregledavanje sadržaja na partnerskim institucijama.

Koraci daljnjeg razvoja ovise o prihvaćenosti inicijalne ideje na tržištu. Primjerice, ukoliko korisnici budu više koristili mogućnost lokacijskih usluga, kao što je npr. nalaženje događaja u blizini te navigiranje do partnerskih institucija, tada bi i fokus budućih verzija bio na lokacijskim uslugama a ne na samom pregledavanju sadržaja partnerskih institucija, samim time bi se promijenio i vizualni identitet same aplikacije.

U slučaju da inicijalna ideja (MVP) bude prepoznat od strane tržišta kao dobra ideja za aplikaciju, postoje tri glavna smjera daljnjeg razvoja:

1.) Proširenje aspekta korištenja aplikacije - pod tim se podrazumijeva proširenje korištenja funkcionalnosti istraživanja sadržaja na interaktivan način, na područje grada Pule umjesto, kao što je trenutni slučaj fokusiranje na partnerske institucije. U tom scenariju postojao bi jedan administrator, primjerice zaposlenik Turističke

zajednice grada Pule, koji bi imao ovlasti prijave na administratorske stranice, te dinamički dodavati ili brisati sadržaj koji će moći pregledavati svi posjetitelji grada Pule.
2.) Proširenje društvenog aspekta aplikacije - pod društvenim aspektom aplikacije prvenstveno se misli na proširivanje aspekta društvene mreže. Primjerice, u tom scenariju bi se svaki zainteresirani korisnik mogao registrirati na službenim stranicama, te tamo dodavati sadržaj te uređivati svoju vlastitu rutu unutar aplikacije. Pregledavanju takvog sadržaja mogli bi pristupiti samo korisnikovi prijatelji, na taj način postoji potencijal stvaranja jedne vrste internog “treasure hunta” , gdje bi određena skupina ljudi mogla sama kreirati svoje sadržaje i zabavljati se s njima.
3.) Proširenje poslovnog aspekta aplikacije - pod poslovnim aspektom aplikacije, smatra se spajanje lokalnih poslovnih subjekata koji bi bili voljni ostvariti suradnju s partnerskim institucija na način da nude popuste za svoje proizvode posjetiteljima institucija. U takvom scenariju pravo na popust imali bi samo posjetitelji koji su pregledali sve sadržaje partnerskih institucija, a pravo na korištenje popusta dobili bi putem QR koda koji bi naknadno bio poslan posjetiteljima na e-mail adresu.

Zaključak

Od pojave pametnih uređaja na tržištu do danas bilježi se konstantan rast u korištenju mobilnih aplikacija. Turisti sve češće koriste svoj pametni telefon kao svojevrstnog turističkog vodiča kako bi se lakše kretali kroz turističku destinaciju koju posjećuju. Taj personalizirani turistički vodič savjetuje korisnike koja mjesta posjetiti pa čak i u kojim će objektima najviše uživati. Posebice mlađe generacije sve više traže dodatni sadržaj putem svojih pametnih uređaja. Većina kulturnih institucija i turističkih institucija razumije tu potrebu te kontinurano traže nove načine na koji mogu približiti svoj sadržaj mlađim populacijama.

Ovaj je diplomski rad nastao kao motivacija za pronalaženjem novog načina korištenja modernih tehnologija kako bi se dodatno približio sadržaj kulturnih institucija njihovim posjetiteljima. Kroz tri tematske cjeline prikazan je proces izrade aplikacije; od prve tematske cjeline gdje je objašnjena početna ideja te istraživanje tržišta za zainteresiranost za aplikaciju, preko druge tematske cjeline gdje su opisane tehnologije korištene u izradi aplikacije do treće tematske cjeline gdje je opisan proces razvoja aplikacije.

Osobni su dojmovi autora da je izrada aplikacije bila način na koji se može dodatno poboljšati ponuda sadržaja kulturnih institucija te modernizirati sveukupni turistički sadržaj u Istarskoj županiji.

Literatura

Knjige

[1] Eric Ries - The Lean Startup,Mate d.o.o.,2011

[2] Ash Maurya -Running Lean: Iterate from Plan A to a Plan That Works,O'Reilly,2010

[3] Scott Grossman - Minimum Viable Product: Master Early Learning and Develop an MVP with Scrum,Kindle edition,2017

Web izvori

[1] U.S Bureau of statistics- https://www.bls.gov/bdm/us_age_naics_00_table7.txt [Pristupljeno: 10.05.2017]

[2] Gradle - https://gradle.org/docs [Pristupljeno: 10.05.2017]

[3] Android developers - https://developer.android.com/index.html [Pristupljeno: 10.05.2017]

[4] JavaScript - http://media.wiley.com/product_data/excerpt/88/07645790/0764579088.pdf [Pristupljeno: 10.05.2017]

[5] VISION API- https://cloud.google.com/vision/?utm_source=google&utm_medium=cpc&utm_campaign=2017-q1-cloud-emea-gcp-skws-freetrial&gclid=CIOPoriR8NMCFdXJsgodIMsLHg [Pristupljeno: 11.05.2017]

[6] UML DIAGRAMS- http://www.uml-diagrams.org/use-case-diagrams.html [Pristpljeno: 11.05.2017]

Popis slika:

Slika 1. Obrazac iteracije u Lean metodologiji 5
Slika 2. Pirate Metrics model 6
Slika 3 Grafikon rezultata odgovora na prvo pitanje anket,, izvor: izradio autor 11
Slika 4. Grafikon rezultata odgovora na drugo pitanje ankete, izvor: izradio autor.....12
Slika 5. Grafikon rezultata odgovora na treće pitanje ankete, izvor: izradio autor 13
Slika 6 Grafikon rezultata odgovora na četvrto pitanje ankete, izvor: izradio autor 13
Slika 7 Grafikon rezultata odgovora na peto pitanje ankete, izvor: izradio autor 14
Slika 8. Stog operacijskog sustava Android, izvor: https://developer.android.com/guide/platform/index.html , pristupljeno:10.05.2017 ...15
Slika 9. Deklaracija Gradle zadataka, izvor: izradio autor 18
Slika 10. Primjer definiranja klase u AndroidManifest-u , izvor: https://developer.android.com/guide/topics/manifest/manifest-intro.html, pristupljeno: 10.05.2017.....20
Slika 11. Primjer alternativnog definiranja klase u AndroidManifest-u izvor: https://developer.android.com/guide/topics/manifest/manifest-intro.html, pristupljeno: 10.05.2017.....20
Slika 12. Primjer deklariranja Permission-a u AndroidManfiest-u izvor: https://developer.android.com/guide/topics/manifest/permission-element.html, pristupljeno: 10.05.2017.....22
Slika 13. Primjer korištenja vertikalnog LinearLayout elementa, izvor: https://developer.android.com/guide/topics/ui/declaring-layout.html, pristupljeno: 10.05.2017.....23
Slika 14. Učitavanje XML korisničkog sučelja, izvor: https://developer.android.com/guide/topics/ui/declaring-layout.html, pristupljeno: 10.05.2017.....23
Slika 15. Primjer upita poslanog VISION API-ju, izvor: https://cloud.google.com/vision/docs/request, pristupljeno: 11.05.2017.....27
Slika 16. Primjer odgovora VISION API-ja, izvor: https://cloud.google.com/vision/docs/request, pristupljeno:10.05.2017 28
Slika 17 Use-case diagram web platforme, izvor: izradio autor 30
Slika 18 Use-case diagram Android aplikacije, izvor: izradio autor 31
Slika 19 Karikaturni prikaz HistriApp sustava, izvor: izradio autor.....32
Slika 20. Izgled sučelja web platforme, izvor: izradio autor.....34
Slika 21. Primjer konfiguriranja ruta u AngularJS-u, izvor: izradio autor.....35
Slika 22. Primjer spajanja HTML komponenti s AngularJS skriptom, izvor: izradio autor 36
Slika 23. Primjer korištenja servisa unutar AngularJS skripte, izvor: izradio autor 36
Slika 24 Konfiguriranje backend skripte, izvor: izradio autor.....37
Slika 25 Primjer unosa novog sadržaja, izvor: izradio autor.....38
Slika 26 Pozivanje VISION API-a, izvor: izradio autor 39
Slika 27 Spremanje novog sadržaja u bazu podataka, izvor: izradio autor 40
Slika 28 Pozivanje backend skripte za Android aplikaciju, izvor: izradio autor 41
Slika 29 Početna aktivnost Android aplikacije, izvor: izradio autor.....42
Slika 30 Stvaranje instance BroadcastReceiver-a, izvor: izradio autor.....44
Slika 31 Primjer upravljanja lokacijskim podatcima, izvor: izradio autor 44
Slika 32 Izgled NavigationDrawer komponente, izvor: izradio autor 45

Slika 33. Primjer kreiranja "syncAdventureAndProceed" metode, izvor: izradio autor 47
Slika 34. Primjer upravljanja odgovorom na JSON zahtjev, izvor: izradio autor 49
Slika 35 Aktivnost za pretraživanje sadržaja partnerske institucije, izvor: izradio autor 50
Slika 36. Primjer deklaracije "callCloudVision" funkcije, izvor: izradio autor.....53
Slika 37. Primjer koda odgovornog za manipuliranjem odgovora na VISION API upit, izvor: izradio autor 55
Slika 38 Aktivnost za pregled ostvarenog rezultata, izvor: izradio autor.....56
Slika 39 Aktivnost za prikaz događaja u gradu, izvor: izradio autor 57
Slika 40 Aktivnost za prikaz lokacija partnerskih institucija, izvor: izradio autor.....58

Sažetak

Zadatak ovog diplomskog rada bio je realizirati sustav namjenjen kulturnom turizmu, koji će omogućiti kreiranje i izmjenjivanje sadržaja (npr. fotografija kulturnih eksponata) na web administratorskom panelu, pozivanje VISION API-ja za obrađivanje slike te spremanje rezultata u bazu podataka. Također je bilo potrebno realizirati edukativno-zabavnu Android aplikaciju zaduženu za prikaz pripadajućeg sadržaja krajnjim korisnicima (npr. posjetiteljima) te im omogućiti fotografiranje i uspoređivanje rezultata iz vision api-a nad uslikanom fotografijom na uređaju sa rezultatima spremljenim u bazi podataka. Kako bi se ostvarili navedeni zadatci bilo je potrebno proučiti način rada Android operacijskog sustava te istražiti i osmisliti arhitekturu sustava za web i mobilnu aplikaciju.

Abstract

The main assignment behind this final thesis was to develop a platform that is intendent for cultural tourism. That platform has to allow users to create and edit content on their web panel, and to call VISION API alghorithm for recognizing the content of an image, and saving the results in a database. It was also necessary to develop an Android native application that is in charge of showing appropriate content to the end users and which will allow them to take pictures and to compare the results from their pictures with the results that are saved on a database. To accomplish these main tasks, it was necessary to study the Android operating system and to develop an architecture for the web and mobile application.