PREGLED PROPUŠTENIH KULTURNIH OBJEKATA NA PRAĆENOJ RUTI

Šulentić, Tomislav

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

PREGLED PROPUŠTENIH KULTURNIH OBJEKATA NA PRAĆENOJ RUTI

Tomislav Šulentić

Zagreb, listopad 2018

Sažetak

Završni rad govori o praćenju propuštenih kulturnih objekata na ruti koju korisnik odabere te detaljni prikaz položaja tih objekata i kratki opis o njima koji omogućuju korisniku mogućnost da u budućnosti ne propusti navedene objekte ako su oni u sferi korisnikovih interesa.

Početni dio rada se odnosi na anketu kojom testiramo varijabilnost aplikacije. Anketiranje će se vrštiti nad ispitanicima za koje smatramo da bi mogli koristiti naše aplikacije.

Glavni dio rada obrađuje aplikacije koje demonstriraju praćenje korisnikovog položaja u zadanom vremenskom periodu, obradu tih položaja te provjeru da li je na unaprijed određenom radijusu korisnik propustio neki kulturni objekt.

Prva mobilna aplikacija prati korisnikove kretnje u zadanom vremenskom periodu te je sa zadanim radijusom unutar kojeg se traže kulturni objekti. Druga je web aplikacija na kojoj korisnik ima detaljan pregled svih ruta te propuštenih objekata na svakoj od njih. Zadnji dio sustava čini poslužiteljska aplikacija preko koje mobilna i web aplikacija komuniciraju.

Cilj rada je utvrditi da li će znanje da na određenim lokacijama postoje kulturni objekti za koje korisnik prije nije znao da su tamo potaknuti korisnika da posjeti navedene objekte ili mu omogućiti da kroz web aplikaciju dobije kratak uvid o kakvom se točno objektu radi te koje su njegove zanimljivosti.

Sadržaj

1. Uvod

U današnjem svijetu, u kojem je tempo života prosječnog čovjeka znatno porastao, sve manje ljudi ima vremena i volje za kulturnim uzdizanjem. Jedan od glavnih problema jest činjenica da se kulturni objekti ne reklamiraju, a u današnjem svijetu u kojem dominiraju internet i televizija to je ekvivalentno njihovom nepostojanju. Jedini način je da samostalno tražite kulturne objekte koji vas interesiraju, a zbog prije navedenih razloga većina ljudi nema vremena niti volje za takvo što.

Kako bismo riješili ovaj problem, biti će izrađene dvije aplikacije koje će biti zadužene za pratnju korisnikove lokacije kroz određeni vremenski period koji sam korisnik bira te uz određeni radijus traženja kulturnih objekata na lokacijama kroz koje korisnik prođe te na koncu ispis propuštenih objekata na zadanoj ruti.

Na kraju rada ćemo izvesti zaključak preko kojeg ćemo pokušati dokazati da znanje postojanja određenih kulturnih objekata na lokacijama za koje korisnik nije znao će ga nagnati da bar istraži što se nudi tamo, a u najboljem slučaju posjeti objekt te obavijesti svoje prijatelje ili obitelj o njegovom postojanju te samim time znatno povećati kulturnu osviještenost svojeg užeg kruga.

2. Hipoteza i cilj

Hipoteza rada glasi: „Poznavanje lokacija kulturnih objekata na nekom području će motivirati ljude da istraže ili posjete navedene objekte“.

Cilj ovog rada je demonstrirati da uz pomoć tehnologije možemo kulturno uzdizati sebe kao i svoje bližnje te time znatno doprinijeti generalnom kulturnom obrazovanju unutar svojeg kruga ljudi, a i šire, sa eksponencijalnim rastom u vidu usmene predaje ili pisanja blogova, komentara na forumima ili društvenim mrežama. Ova tvrdnja će biti konkretnije istražena na različitim demografskim skupinama.

U sklopu rada će biti napravljene dvije aplikacije koje će omogućiti praćenje korisnikove pozicije u zadanom vremenskom periodu te zadanom radijusu i taj dio će obavljati mobilna aplikacija, dok će web aplikacija omogućiti cjeloviti prikaz svih korisnikovih ruta te propuštenih objekata na tim rutama. Obje aplikacije će koristiti zajednički poslužitelj te ćemo time dobiti zaokruženu cjelinu koja će se brinuti za ostvarivanje zacrtanih ciljeva.

3. Istraživanje o kulturnoj osviještenosti

Kako bismo dobili dobar uvid u ciljanu skupinu, za uporabu aplikacije izvršit ćemo istraživanje koje će nam pokazat interes skupine za uporabom tehnologija koje bi im pomogle u kulturnom uzdizanju.

3.1. Anketiranje o uporabi aplikacije za praćenje kulturnih objekata

Sama anketa će se provoditi nad populacijom različitih dobnih skupina i to sa naglaskom na ljude koji su skloniji korištenju različitih oblika tehnologije te su potencijalno i otvoreniji prema novim načinima uporabe tehnologije. Samim time postoji i veća mogućnost da bi taj dio populacije htio koristiti naše aplikacije.

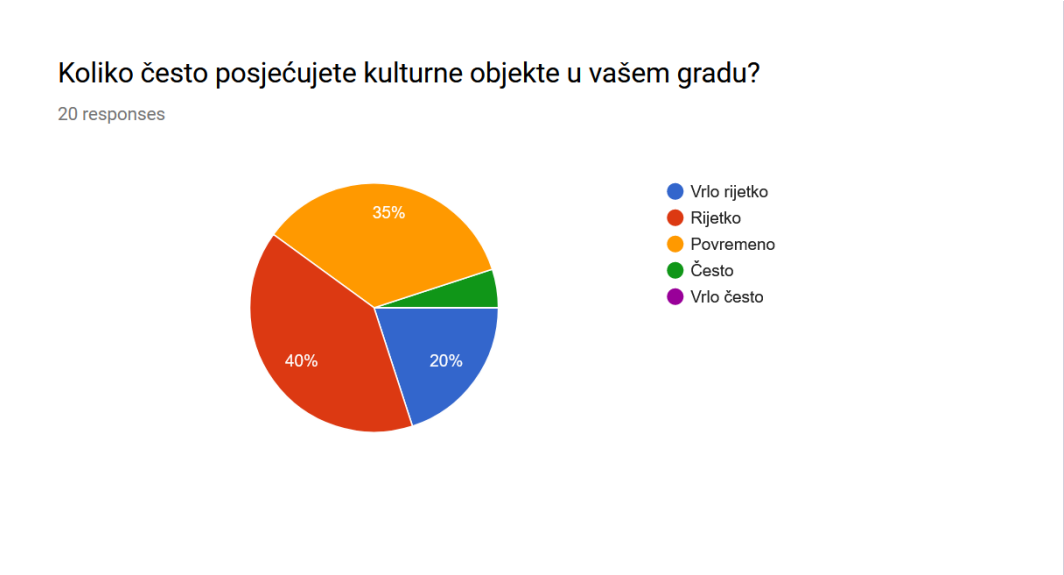
Anketa će biti anonimna jer u ovom slučaju nisu bitni identiteti ispitanika, nego statistički postotak koji bi nam ukazao na potencijal našeg projekta. Kao ciljana skupina izabrani su zaposlenici programske tvrtke Agrivi, a s obzirom da se radi o tehnološkoj tvrtci sa zaposlenicima sklonim tehnologiji te različitih dobi i pozadina, dobili smo izvrsnu podlogu za izradu ankete.

Unutar ankete se nalaze 4 pitanja:

1. Koliko često posjećujete kulturne objekte u vašem gradu?
2. Koliko dobro poznajete lokacije kulturnih objekata u vašem gradu?
3. Da li bi poznavanje lokacija kulturnih objekata povećalo vašu učestalost posjete istih?
4. Da li biste htjeli koristiti tehnologiju za upoznavanje kulturnih objekata?

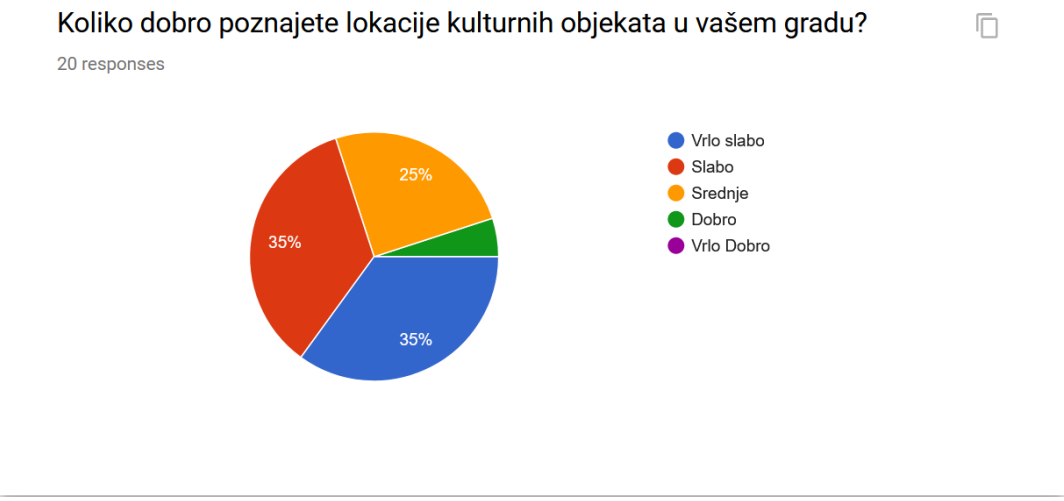
Ideja pitanja jest dobiti uvid u to koliko prosječna osoba iz ciljane skupine trenutno posjećuje kulturne objekte u svom okruženju te kolika je povezanost između posjećenosti te znanja o lokaciji samih objekta te da li se ona može unaprijediti uporabom tehnologije, u ovom slučaju naših aplikacija.

3.2. Rezultati anketiranja



Slika 3.1 Odgovori na prvo pitanje iz ankete

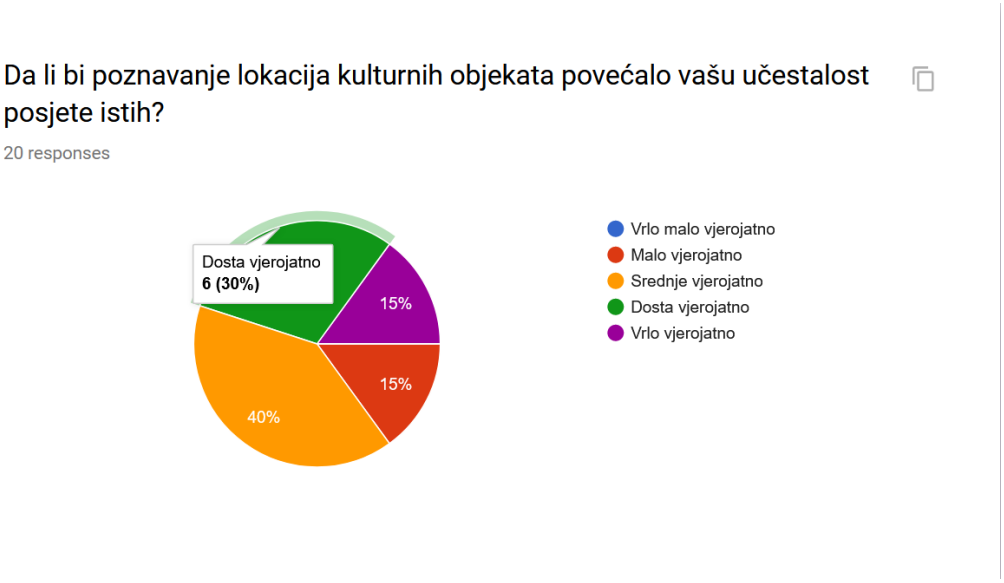
Pogreška! Izvor reference nije pronađen. nam prikazuje odgovore na prvo pitanje iz ankete. Vidimo da većina ispitanika ne posjećuje često kulturne objekte, sa 60% ispitanika koje su odgovorile rijetko ili vrlo rijetko možemo očekivati samo porast uporabom aplikacija.



Slika 3.2 Odgovori na drugo pitanje iz ankete

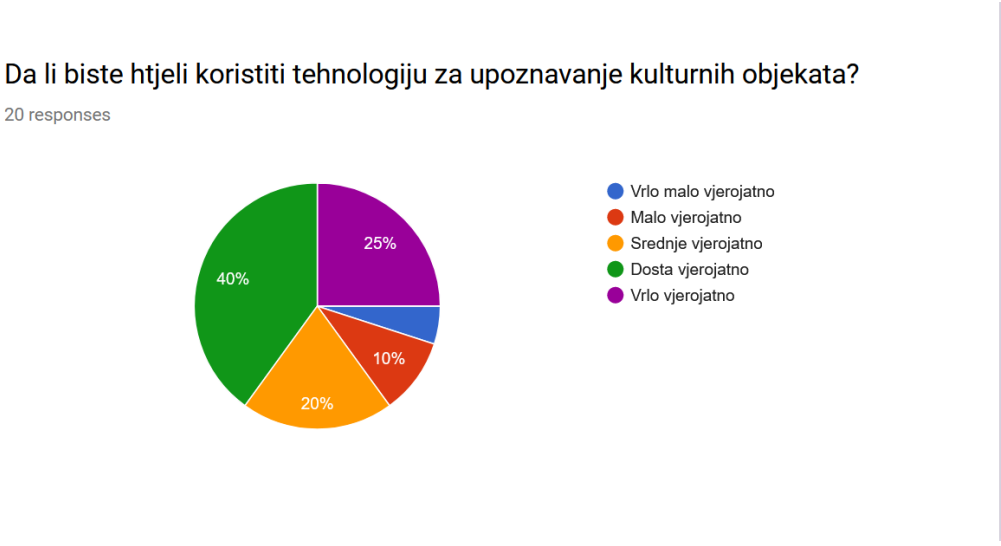
Pogreška! Izvor reference nije pronađen. nam prikazuje odgovore na drugo pitanje iz ankete. Samo 30% ispitanika je donekle upoznato sa svim kulturnim objektima u svom

gradu, dok su većina ostalih upoznati sa bitnim objektima, ali kako je bilo i za očekivati ne sa svima. Pošto se u pitanju navodi okruženje u kojem ispitanici žive, lako je zaključiti kako bi statistika za lokacije na kojima prije nisu bili bila još više usmjerena prema negativnom spektru znanja.



Slika 3.3 Odgovori na treće pitanje iz ankete

Pogreška! Izvor reference nije pronađen. nam prikazuje odgovore na treće pitanje iz ankete. Iz njega vidimo veliki potencijal za našu aplikaciju jer pokazuje da postoji direktna povezanost između znanja o postojanju kulturnog objekta na nekoj lokaciji te posjećenosti tog objekta.



Slika 3.4 Odgovori na četvrto pitanje iz ankete

Pogreška! Izvor reference nije pronađen. nam prikazuje odgovore na četvrto pitanje iz ankete. S obzirom na da je ciljana grupa tehnološki orijentirana, rezlutati su očekivajuće pozitivni.

Na koncu možemo zaključiti kako postoji direktna povezanost između znanja o lokaciji kulturnog objekta te posjećenosti istog, te kako bi aplikacija koja nudi mogućnost upoznavanja lokacija kulturnih objekata bila idealna za ljude koji su zainteresirani za kulturno uzdizanje. Na koncu treba zaključiti kako je anketa provedena na razini lokacije koju ispitanik poznaje, tako da je za očekivati da bi potreba za aplikacijom bila još veća na lokacijama na kojima je korisnik prvi put.

4. Izabrana implementacija rješenja

Kako bismo adekvatno riješili navedene probleme odlučeno je izraditi dvije klijentske aplikacije koje će komunicirati sa jednom poslužiteljskom aplikacijom. Prva je mobilna aplikacija koja se koristi za lociranje korisnika u prostoru i kulturnih objekata na zadanoj ruti te zadanom radijusu. Druga je web aplikacija koja će omogućavati pregled svih korisnikovih ruta te propuštenih objekata na tim rutama. Prilikom faze dizajna aplikacije odlučeno je kako će se mobilna aplikacija razvijati na Android operativnom sustavu, uz primjenu Microsoft Xamarin programskog okvira namijenjenog za razvoj mobilnih aplikacija u .NET okruženju, koji je najzastupljeniji operativni sustav za mobitele u ovome trenutku te nudi sve mogućnosti koje su potrebne kako bismo mogli adekvatno riješiti zadani problem. Web aplikacija je razvijena u Angular 6, to je besplatni programski okvir (engl. *Framework*) koji omogućava izradu klijentskih aplikacija. Također, odlučeno je da će obje klijentske aplikacije imati iste smjernice za korisničko sučelje kako bismo korisnicima olakšali korištenje obje aplikacije i pružili im slično sučelje koje će smanjiti vrijeme prilagodbe.

Prvotno ćemo obraditi funkcionalnost sustava (4.3), nakon toga ćemo obraditi arhitekturu obje aplikacije (4.2) i tehnologije koje su korištene u izradi. Na koncu ćemo obraditi rješenja u poslužiteljskoj aplikaciji (4.3) te korisničkom sučelju (4.3).

4.1. Funkcionalnosti sustava

Uz prethodno spomenutu činjenicu da je rješenje implementirano uz pomoć dvije aplikacije, prvotno ćemo obraditi mobilnu aplikaciju te njene funkcionalnosti.

Funkcionalnosti mobilne aplikacije su:

- Registracija i prijava korisnika
- Pregled dosad obavljenih ruta
- Pregled i uređivanje postavki za praćenje korisnika
- Upute za korištenje aplikacije

Korisnik mora biti registriran da bi mogao koristiti aplikaciju, a registracija se može izvršiti na mobilnoj i na web aplikaciji. Nakon što je registriran, korisnik dobiva pristup na obje aplikacije te može koristiti sve funkcionalnosti unutar obje aplikacije.

Na početnom ekranu aplikacije je pregled ruta koje je korisnik do sada obavio. Prikazuje se ukupan broj ruta, rute koje su završene, a pod završenom rutom se smatra ona u kojoj je korisnik posjetio sve kulturne objekte ili je na web aplikaciji označio kao završenu, te broj propuštenih ruta, a pod njih ulaze sve one u kojima je korisnik propustio barem jedan kulturni objekt na zadanoj ruti te ju nije označio kao završenu na web aplikaciji. Također su prikazani korisnikov avatar, email sa kojim se prijavio te njegova trenutna generalna lokacija (prvo veće mjesto ili grad na temelju korisnikove trenutne lokacije).

Pregled i uređivanje postavki za praćenje korisnika se sastoji od dvije cjeline; prva se odnosi na postavke markera dok se druga odnosi na postavke pri promjeni lokacije.

Prva cjelina unutar postavki za praćenje korisnika se odnose na markere kojima se obilježava korisnikova lokacija. Moguće je namjestiti da se marker prikazuje na početku, na kraju, te na svakoj promjeni lokacije.

Druga cjelina omogućava korisniku da postavi koliko često će se pratiti njegova lokacija u vremenskim jedinicama, postoje četiri moguće opcije:

- 2 sekunde
- 10 sekundi
- 25 sekundi
- 1 minuta.

Nadalje, omogućuje se postavljanje minimalne distance koja mora biti prijeđena da bi se zabilježila promjena lokacija, kod koje postoje 4 opcije:

- 1 metar
- 3 metra
- 5 metara
- 10 metara

Unutar uputa za korištenje aplikacije se obaviještava korisnika na koji način se koristi aplikacija, objašnjava se značenje cjelina u postavkama te kako da započne i završi neku rutu.

Web aplikacija ima sljedeće funkcionalnosti:

- Prijava korisnika,
- Uređivanje korisničkih podataka,
- Cjeloukupan pregled dosadašnjih ruta i

- Detaljni pregled svake od ruta sa informacijama o propuštenim kulturnim objektima na toj ruti.

Prije korištenja web aplikacije od korisnika se traže email i lozinka kako bi se logirao u sustav.

Na početnom ekranu su razne statistike i grafovi koji prikazuju podatke o broju korisnikovih ruta te osnovne podatke o njima. Također postoji stranica sa popisom svih ruta koje je korisnik ostvario, te se klikom na jednu od njih dolazi do detalja za tu rutu.

Na detaljima za svaku rutu su prikazani podaci o kretanjima na toj ruti unutar Google karata, također su označeni kulturni objekti te kratke informacije o samom objektu. Mogu se mijenjati naziv rute, te dodati opis ili završiti rutu što znači da smatramo da smo naučili gdje se nalaze objekti unutar te rute.

Uređivanje korisničkih podataka omogućuje korisniku da promjeni svoj email, lozinku te avatar te se promjene tamo učinjene propagiraju i na mobilnu aplikaciju pošto se koristi ista baza podata za prijavu i autentikaciju korisnika. Ako korisnik na web aplikaciji promjeni email i lozinku, pri ponovnom logiranju u mobilnu aplikaciju će morati koristit nove vrijednosti koje je postavio na web aplikaciji. Također promjena avatara se vidi na mobilnoj i web aplikaciji; na web aplikaciji čim se postavi novi avatar, a na mobilnoj pri slijedećoj uspješnoj prijavi.

4.2. Arhitektura sustava

Kako bismo mogli adekvatno razviti bilo koju aplikaciju moramo postaviti primjereni arhitekturalni model, a u slijedećim poglavljima ćemo opisati te objasniti rješenja koja smo izabrali za ovaj sustav.

Prvotno ćemo objasniti što je interoperabilnost informacijskih sustava, njezinu važnost te primjenu na ovom sustavu (4.3).

Zatim ćemo se koncetrirati na pojedine aplikacije te njihove arhitekturalne modele te ih podrobnije objasniti (4.3).

Na koncu ćemo objasniti način autentifikacije te autorizacije korisnika, proces koji osigurava da samo autorizirani korisnici imaju pristup aplikacijama te da svaki korisnik ima pravo pristupa samo svojim podacima (4.3).

4.2.1. Interoperabilnost informacijskih sustava

Kao što smo već naglasili, ovo rješenje se sastoji od dvije klijentske aplikacije koje su implementirane na različitim platformama te jedne poslužiteljske aplikacije. Kako obje klijentske aplikacije komuniciraju sa istom poslužiteljskom aplikacijom te samim time barataju sa istim podacima, moramo omogućiti što jednostavniju komunikaciju prema poslužiteljskoj aplikaciji.

Sam pojam interoperabilnost se inicijalno pojavio kao vojni termin te opisuje mogućnost sustava (jedinica, naoružanja) da surađuje s drugima u cilju zajedničkog djelovanja. Svrha i cilj je povećanje učinkovitosti sustava tako da se optimizira brzina, automatizacija procesa, te omogućavanje djelovanja na velikim udaljenostima [2].

Kako bismo to omogućili, odlučeno je da će se poslužiteljska aplikacija temeljiti na arhitekturalnom modelu zvanom REST (engl. *Representational state transfer*).

REST model omogućava pristup te manipulaciju nad podacima koristeći unificiran te predefiniran set operacija za razliku od npr. WSDL (engl. *Web Services Description Language*) ili SOAP (engl. *Simple Object Access Protocol*) koji imaju vlastite setove operacija koji se mogu razlikovati u svakoj od implementacija potencijalnih rješenja.

Sam termin REST je definirao Roy Fielding u svojoj dokotorskoj disertaciji 2000. godine u kojoj je objasnio REST principe koji su do tada bili znani kao HTTP objektni model (engl. *HTTP object model*) te su bili korišteni u dizajniranju HTTP-a (engl. *Hypertext Transfer Protocol*) [4]. Sam termin REST je bio zamišljen na način da objasni kako bi se trebala ponašati dobro definirana web aplikacija koju je on definirao kao mrežu web resursa, virtualni stroj stanja (engl. *virtual state machine*), gdje korisnik koristeći poveznice (engl. *link*) napreduje kroz aplikaciju.

REST se oslanja na korištenje HTTP kao glavnog protokola te se za manipulaciju nad resursima koristi postojeće HTTP metode kao što su POST, GET, PUT, DELETE koje su ekvivalentne CRUD[1] operacijama u bazama podataka.

[1] CRUD je skraćenica za Create (kreiranje), Read ili Retrieve (Dohvaćanje), Update (Ažuriranje) i Delete (brisanje).

U današnje vrijeme, format podataka koje je standard za slanje i dohvaćanje podataka postao je JSON (engl. *JavaScript Object Notation*), te pošto je kompatibilan sa REST modelom te gotovo sve ozbiljne platforma ga podržavaju ili koriste i mi smo ga odlučili koristiti.

```
PUT /api/Routes
```

Kod 4.1 Primjer PUT zahtjeva iz mobilne aplikacije

Kod 4.1 prikazuje operaciju ažuriranja podataka o ruti koja se poziva iz mobilne aplikacije. Tijelo (engl. *payload*) ovog zahtjeva prima objekt rute u JSON formatu u kojem se nalaze podaci o korisnikovoj ruti.

```
GET /api/Routes
```

Kôd 4.2 Primjer GET zahtjeva iz web aplikacije

Kôd 4.2 prikazuje dohvat svih ruta iz web aplikacije. Podaci dolaze u JSON formatu kao niz objekata (engl. *objects array*) s podacima o svim korisnikovim rutama.

S obzirom na navedene primjere, možemo zaključiti da se radi o interoperabilnom rješenju jer dvije aplikacije izvedene na različitim platformama komuniciraju s istim poslužiteljem. Jedini preduvjet koji aplikacije moraju ispuniti je poznavanje sučelja (engl. *interface*) metoda na poslužitelju s kojim vrše komunikaciju kako bi mogle napraviti valjan zahtjev (engl. *request*), a na samoj aplikaciji je da obradi odgovor (engl. *response*) te čini s podacima koje god procese su predviđeni za tu specifičnu aplikaciju.

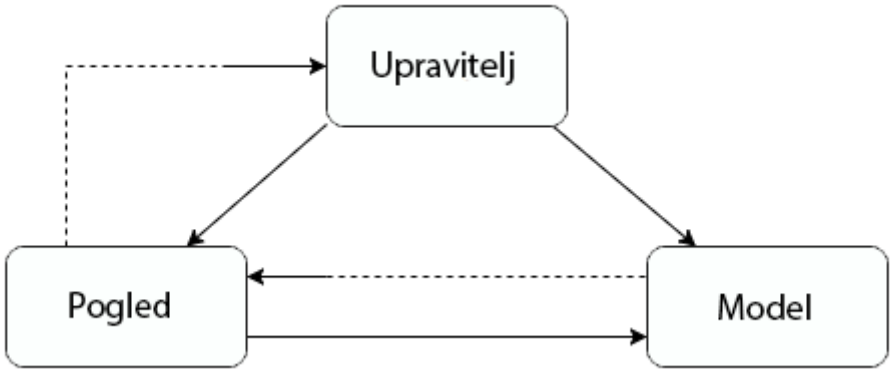
4.2.2. Arhitekturalni modeli sustava

Opisavši u prethodnom poglavlju što definira interoperabilno rješenje te REST model koji omogućava komunikaciju te razmjenu podataka između sustava, u slijedećim poglavljima ćemo opisati arhitekturalni model samih aplikacija, tj. strukturu programskog koda. Prvotno ćemo opisati što je Model-Pogled-Upravitelj (engl. *Model–View–Controller* ili skraćeno MVC), a nakon toga Model-Pogled-Prikazivač (engl. *Model–View–Presenter* ili skraćeno MVP).

MVC je obrazac softverske arhitekture koji ima široku primjenu, ali se tradicionalno te najčešće koristi za razvijanje korisničkih sučelja. Razdvaja aplikaciju u tri povezana dijela u kojem svaki dio ima svoje odgovornosti. Razlog zbog kojeg se to čini jest da se razdvoji interna prezentacija podataka od načina na koji su podaci prikazani i prihvaćeni od korisnika

aplikacije. Većina popularnih programskih jezika u današnje vrijeme ima programske okvire koji podržavaju MVC.

MVC je povijesno gledano jedan od plodonosnijih arhitekturalnih obrazaca te je bio jedan od prvih koji je pristupio razvoju te opisu softvera na način da se sami kod razdvoji po odgovornosti.



Slika 4.1 Model-Pogled-Upravitelj model

Slika 4.1 prikazuje MVC arhitekturalni obrazac. Strelice prisutne na slici nam pokazuju tip veze između slojeva aplikacije; puna crta predstavlja direktnu vezu dok isprekidana crta predstavlja indirektnu vezu. Upravitelj je glavni dio ovog arhitekturalnog modela jer on upravlja korisničkim zahjevima. Upravitelj zaprima korisničke naputke te izgrađuje model, a na koncu šalje taj model u pogled koji se prikazuje korisniku. Upravitelj možemo promatrati kao segment koji konvertira korisničke naputke u naloge za pogled ili model.

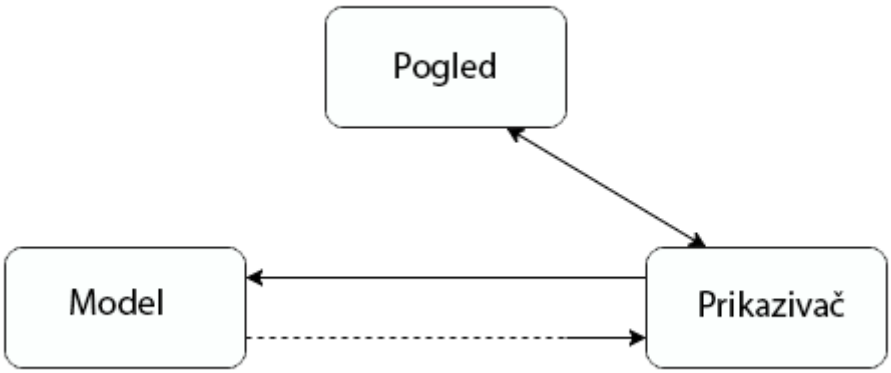
Model predstavlja ponašanje aplikacije u sferi domene problema, neovisno o grafičkom sučelju aplikacije. Model se može sastojati od podataka, poslovnih pravila, logike te funkcija ugrađenih u poslovnu logiku (engl. *business logic*). Model dojavljuje sebi pridruženim pogledima i upraviteljima kada je došlo do promjene u njegovom stanju, odnosno u stanju podataka unutar njega. Ove dojave omogućuju pogledu da prikaže obnovljeno stanje modela, a upravitelju promjenu dostupnog skupa naredbi [3].

Pogled je dio MVC – a koji je zadužen za prikaz informacija poslanih iz modela korisniku, zato i postoji direktna veza između modela i pogleda, model obrađuje podatke koje pogled prikazuje. Indirektna veza između pogleda i upravitelja postoji iz razloga što pogled može od upravitelja zatražiti obradu podataka kako bi mogao prikazati novo stanje tih obrađenih podataka. Osvježavanje podatakta prikazanih u pogledu se na taj način može vršiti asinkrono ili osvježavanjem cijelog pogleda.

Iz prethodno definiranih dijelova MVC arihtekture možemo vidjeti zašto je toliko raširen i popularan; pruža jasnu podjelu aplikacije u segmente te time olakšava brzinu razvoja, održavanje te prototipiranje novih modula. U današnje vrijeme MVC je jedan od najzastupljenijih arhitekturalnih obrazaca te kao što je već prije spomenuto, većina popularnih programskih jezika te njihovih programskih okvira podržava MVC.

MVC svoju široku raširenost ima najviše zbog svoje jednostavnosti implementacije. Osnovna veza u kojoj Upravitelj kontaktira Model za podatke i zatim ih ispisuje na Pogled razumljiva je i lako se primjenjuje. Treba napomenuti da se u upotrebi slojeva može jednostavno pogriješiti pa obradu podataka odraditi u sloju Pogleda ili Modela što u manjoj mjeri nije preveliki problem, ali isto tako može model učiniti nevaljalim. Time on gubi svoju ideju, a to je da bolje organizira komponente u aplikaciji.

Kako smo ranije napomenuli, Web aplikacija koristi Angular 6 programski okvir koji podržava arhitekturu ekvivalentnu MVC-u, te je rješenje tako implementirano o čemu ćemo više govoriti u nastavku rada (4.3).



Slika 4.2 Model-Pogled-Prikazivač obrazac

Slika 4.2 prikazuje MVP arhitekturalni obrazac, pošto vuče svoje korijene iz MVC – a i on je tradicionalno korišten za izgradnju korisničkih sučelja, ali ostavlja manje prostora za druge strukture programkog koda i samim time namjene nego što to čini MVC. Na slici imamo prikaz direktnih i indirektnih veza kao što je bio slučaj sa MVC obrazcem. Pošto je MVP obrazac više usmjeren na izradu korisničkih sučelja možemo po slici primjetiti da se u njegovu centru nalazi Pogled za razliku od MVC – a kojem je glavni dio bio Upravitelj.

Prednosti koje MVP obrazac omogućuje su automatizirani testovi pojedinih modula te poboljšava separaciju odgovornosti u sloju prezentacijske logike.

Pogled u MVP obrascu je pasivno sučelje koje prikazuje podatke i usmjerava naputke zadane od strane korisnika Prikazivaču koji odrađuje potrebne radnje nad tim podacima. Pri inicijalizaciji pogleda bitno je naglasiti da on mora predati svoju referencu Prikazivaču jel upravo je Prikazivač taj koji kako smo već naglasili vrši potrebne radnje nad podacima.

Model u MVP obrazcu je sučelje koje definira podatke koji će biti prikazani ili na kojima će se vršiti određena poslovna logika. Indirektna veza između Modela i Prikazivača služi zato da bi Model mogao obavijestiti Prikazivač u slučaju promjene podataka, nakon toga Prikazivač odrađuje potrebne radnje nad podacima te proslijeđuje obrađene podatke Pogledu koji zatim osvježuje svoj prikaz korisniku.

Prikazivač u MVP obrazcu ima ulogu čovjeka u sredini (engl. *middle-man*) jer se sva prezentacijska logika obavlja upravo u njemu. To je ujedno najveća razlika od MVC obrazca jer, za razliku od MVC – a, Pogled i Model ne komuniciraju direktno.

S obzirom da MVP vuče svoje korijene iz MVC – a, njegova popularnost je znatnim udjelom direktno vezana uz tu činjenicu. Neke od poznatijih platformi te programskih okvira koje koriste MVP su Microsoft Web Forms i Win Forms, prvotna namijenjena za razvoj web aplikacija dok je druga usmjerena na razvoj Windows aplikacija. Nama je posebno zanimljiv jer se naširoko koristi u Android platformi [5].

Kako smo prije napomenuli, mobilna aplikacija je razvijena na Android platformi koja podržava MVP te je rješenje implementirano sa MVP – om, o čemu ćemo više govoriti u nastavku rada (5.3.2).

4.2.3. Autentikacija i autorizacija sustava

Kako bismo omogućili da samo prijavljeni korisnici imaju pristup aplikacijama te da mogu pristupiti samo svojim podacima, moramo riješiti mehanizme za autentifikaciju te autorizaciju korisnika. U nastavku ćemo detaljnije opisati što podrazumijevamo pod tim pojmovima te način na koje smo navedene probleme riješili unutar našeg sustava.

Autentikacija (grč. *authentikos*) je čin potvrđivanja istine nekog atributa ili informacije za koje neki entitet tvrdi da je istinit. Može uključivati potvrđivanje identiteta osobe na način da se provjere njihovi osobni dokumenti, verificira autentitet web stranice sa digitalnim certifikatom ili utvrđivanje identiteta korisnika neke aplikacije što je u ovom slučaju naša

potreba. Uobičajeno je da se identitet korisnika potvđuje uporabom korisničkog imena i pripadajuće lozinke, ali u današnje vrijeme postoje mnogi načini na koji se identitet može potvrditi, neki od njih su preko uporabe bioloških atributa korisnika koji su jedinstveni za svaku osobu na svijetu kao što su na primjer zjenica oka, otisak prsta, boja glasa ili čak prepoznavanje strukture lica korisnika. U našem sustavu koristimo mehanizme za autentikaciju na obe klijentske aplikacije, a koji su to mehanizmi korišteni će biti detaljnije obrađeno u nastavku.

Najkorišteniji tipovi autentikacije korišteni u suvremenim aplikacijama su:

- HTTP Basic autentikacija
- Kolačići (engl. *cookies*)
- Tokeni (engl. *tokens*)
- Jednokratne lozinke (engl. *one-time passwords*)

Najjednostavniji oblik autentikacije jest HTTP basic autentikacija. Koristi se na način da se u svakom zaglavlju HTTP zahtjeva koje klijent šalje ujedno šalju korisničko ime i lozinka korisnika koje su enkodirani pomoću Base64 algoritma[2]. Zbog činjenice da sam HTTP nije siguran protokol rijetko se koristi u današnje vrijeme.

Kolačići su često korišten pristup za autentikaciju zbog jednostavnosti korištenja. Postoje razne implementacije i načini korištenja kolačića, no najčešće se koristi na način da nakon što se korisnik autorizira poslužitelj postavi određene vrijednosti koje se zatim spremaju u kolačić i šalju zajedno s odgovorom klijentu. Na taj način svaki slijedeći zahtjev na poslužitelj ujedno ima i kolačić što omogućava automatsko provjeravanje identiteta korisnika.

Tokeni se koriste na način da nakon što se korisnik autorizira prima određenu vrijednost preko koje se automatski provjerava njegov identitet. Tokeni imaju određeni period u kojem su validni, a nakon što isteknu klijent mora zatražiti novi token kako bi mogao nastaviti koristiti resurse sa poslužitelja. Po uporabi su slični HTTP basic autentikaciji jel se u svakom zahtjevu šalje token za razliku od korisničkog imena i lozinke u slučaju HTTP basic autentikacije te samim time pružaju puno veću razinu sigurnosti jel se tokeni generiraju na zahtjev korisnika te kao što je već napomenuto imaju određeni period trajanja [6].

[2] Dvosmjeran tip enkripcije što znači da je moguće enkriptirati i dekriptirati poruku pomoću istog algoritma.

Jedonkratne lozinke su jedan od najsigurnijih pristupa za autentikaciju korisnika jer omogućuju samo jedan pristup zatraženim resursima na poslužitelju. Uobičajeno je da se koriste u situacijama u kojima je sigurnost od velike važnosti, na primjer u bankarskim sustavima, sustavima za plaćanje ili drugim sustavima u kojima je sigurnost veliki aspekt samog djelovanja sustava.

Autorizacija specificira da li korisnik ima privilegije, odnosno prava za pristup nekim resursima unutar sustava ili određenim radnjama za koje su točno određena prava potrebna da bi se navedena radnja mogla izvršiti. Najbolji primjer zašto je autorizacija toliko važna može se dati izvući iz poslovnih praksi većine tvrtki na svijetu, samo ljudi zaduženi za baratanje informacijama o plaćama zaposlenika neke tvrke bi trebali imati pristup resursima i funkcijama aplikacije koja se bavi tim segmentom dok ostali iz lako zaključivih i logičkih razloga ne smiju moći pristupiti.

Za autentikaciju i autorizaciju korisnika u našem sustavu koristimo Identity Server 4 koji podržava sve prije navedene načine autentikacije i omogućava autorizaciju korisnika.

Mobilna klijentska aplikacija za autentikaciju koristi tokene. Nakon što se korisnik uspješno autorizira koristeći svoje korisničko ime i lozinku prima token koji traje jedan sat, što znači da može pristupati resursima i funkcijama poslužitelja u tom periodu. Nakon isteka jednog sata mora obnoviti token kako bi mogao dalje koristit aplikaciju. Pri primitku tokena se i prima vrijeme mogućeg korištenja, a kako klijent često komunicira sa poslužiteljem, pri svakom zahtjevu se provjerava koliko dugo još token vrijedi. Ako je ostalo 5 minuta ili manje, klijent automatski traži obnovu čime se ne zahtijeva ponovni unos korisničkog imena i lozinke od strane korisnika. Sam token se sprema u lokalnu bazu kako bi bio lako dostupan u svim dijelovima aplikacije te kako je već spomenuto, šalje se zajedno sa svakim zahtjevom na poslužitelj kako bi klijent mogao pristupiti zatraženim resursima.

4.3. Tehnologije i programski okviri sustava

Poslužiteljska aplikacija koristi programski okvir ASP.NET Core 2 koji nam omogućava korištenje REST modela te laganu integraciju sa Identity Serverom za autentikaciju i autorizaciju korisničkih zahtjeva.

Mobilna klijentska aplikacija koristi Android platformu, a web aplikacija koristi Angular 6 uz jQuery javascript biblioteku te Bootstrap 4 CSS biblioteku za oblikovanje grafičkog sučelja.

4.3.1. ASP.NET Core 2 programski okvir

ASP.NET Core 2 je besplatan, otvorenog izvornog koda (engl. *open-source*) programski okvir izrađen od strane Microsofta i zajednice. Predstavlja novu generaciju ASP.NET okvira te kao najveću prednost nad prethodnim verzijama nudi mogućnost izvedbe na različitim platformama, npr. Windows i Linux. Predstavlja kompletno prerađivanje do tada postojećih i razdvojenih okvira ASP.NET MVC i ASP.NET Web API u jedinstveni programski model. Iako je novi programski okvir izgrađen kompletno od temelja, on ima veliku dozu konceptualnih istovjetnosti sa ASP.NET MVC programskim okvirom te kao takav nudi laku prilagodbu programerima koji su u prošlosti koristili ASP.NET MVC.

Prethodno spomenuti ASP.NET MVC i ASP.NET Web API su se koristili za izradu web aplikacija, web servisa te servisa u oblaku (engl. *cloud*), a s obzirom na činjenicu da ASP.NET Core 2 predstavlja unifikaciju ta dva modela, on savršeno odgovara za izradu naše poslužiteljske aplikacije. MVC je dobio naziv po uporabi MVC arhitekturalnog modela kojeg smo prethodno obradili te je, povijesno gledano, naslijedio Microsoft Web Forms, također programski okvir za izradu web aplikacija koji je imao više sličnosti sa izradom nativnih desktop aplikacija za Windowse te zbog toga nije polučio ni približno takav uspjeh kao MVC niti popularnost korištenja. ASP.NET Web API je od svojih konceptualnih začetaka bio usmjeren za stavaranje poslužiteljskih aplikacija, te je arhitekturalno približio REST model širem krugu korisnika (programera) uz podršku za druge starije tipove modela.

Poslužiteljski segment ASP.NET Core 2 je namijenjen izradi web servisa čijim se resursima može pristupiti preko različitih platformi i klijenata te time nam omogućuje prethodno spomenutu interoperabilnost sustava, a samim time i uporabu REST arhitekture koju smo detaljno obradili u prethodnom poglavlju (4.3). Kako se radi o aplikacijama koje ne čuvaju stanja o prethodno napravljenim zahtjevima (engl. *stateless*) mi se moramo pobrinuti o čuvanju podataka, bilo u samoj aplikaciji ili u bazi podataka s kojom aplikacija komunicira.

Kreiranjem novog projekta u ASP.NET Core 2 nudi nam se nekoliko tipova autentikacije koji se koriste ovisno o potrebama za tu specifičnu aplikaciju. To su:

- Bez autentikacije
- Individualni računi (engl. *individual accounts*)
- Organizacijski računi (engl. *organizational accounts*)

- Windows autentikacija

Bez autentikacije se koristi u slučajevima kada nema potrebe za zaštitom pojedinih resursa i funkcija aplikacije, svatko može pristupiti i koristiti sve segmente aplikacije.

Individualni računi se koriste kada svaki korisnik aplikacije ima vlastiti račun te se autenticira sa vlastitim korisničkim imenom i zaporkom. To je najčešće korišten oblik autentikacije korisnika. Podaci o korisniku se spremaju u bazu podataka što obuhvaća i segmente kojima korisnik može pristupiti, odnosno autorizacijska prava na pristup pojedinim resursima sustava. Postoje dva tipa individualnih računa, prvi je preko lokalne autentikacije gdje se korisnikovi podaci čuvaju u lokalnoj bazi podataka, a drugi je preko nekog vanjskog servisa koji ne mora biti razvijen od strane istog proizvođača nego mora poštivati protokole određene od strane aplikacije. Taj način omogućava da se korisnik autenticira sa postojećim korisničkim računom s vanjskog servisa te ne mora raditi novi račun kako bi pristupio resursima aplikacije. Za primjer možemo uzet neke od velikih igrača u IT sektoru za koje većina ljudi već ima izrađene korisničke račune kao što su Facebook, Instagram, Google, Microsoft, LinkedIn.

Organizacijski računi se koriste na način da se korisnik prijavljuje preko LDAP (engl. *Lightweight Directory Access Protocol*) imenika nazvanog Active Directory ili preko postojećeg korisničkog računa iz usluge Office 365 koja omogućuje korisnicima te usluge da koriste isti račun za autentifikaciju.

Windows autentikacija je posljednji izbor te kao što joj samo ime kaže koristi se lokalni Windows account koji se autenticira preko Windows autentikacijskg modula iz Microsoftovog web poslužitelja IIS (engl. *Internet Information Services*).

ASP.NET Core 2 podržava XML (engl. *EXtensible Markup Language*) i JSON (engl. *JavaScript Object Notation*) formate podataka. Kako je XML stariji format koji je teži za korištenje i ljudsko čitanje te zauzima više prostora od JSON – a, prepustio je palicu JSON kao standardu za razmjenu podataka između aplikacija. Iz tih razloga i mi ćemo koristiti JSON kao format podataka za razmjenu podataka.

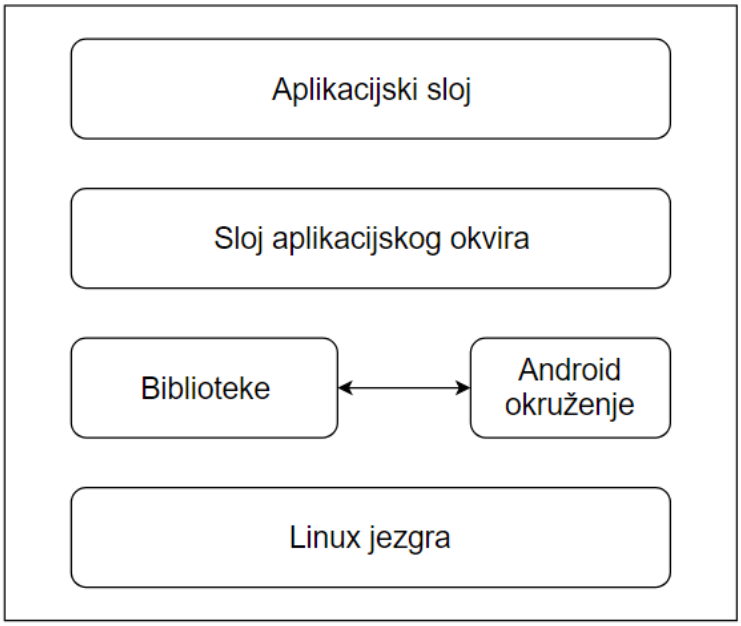
4.3.2. Mobilni operativni sustav Android

Android je mobilni operativni sustav razvijen od strane Googlea. Jezgra mu se bazira na Linuxu te je prvobitno namijenjen za uređaje s ekranima osjetljivim na dodir kao što su

pametni mobiteli i tableti. Glavni programski jezici s kojima je napisan su Java, C i C++. Njegova popularnost proširila se s vremenom na mnoge druge uređaje kao sto su televizori (Android TV), automobili (Android Auto) i satovi (Android Wear). Prvu verziju razvila je tvrtka pod nazivom Android Inc., no 2005 godine kupio ih je Google, te 2007 objavio njegov dolazak na tržište. U Rujnu 2008 godine krenulo je na tržište. 2011 godine je postao najpopularniji operativni sustav za pametne telefone, a 2 godine kasnije također najpopularniji operativni sustav za tablete. Svaka verzija Android operativnog sustava ima naziv po nekoj vrsti slatkiša, a zadnja verzija izdana u Prosincu 2017 godine zove se 8.1 Oreo.

Google Play je digitalni distribucijski servis od Google-a koji se koristi kao službena trgovina za kupnju i preuzimanje Android aplikacija. Osim samih aplikacija mogu se pronaći i razne vrste multimedije kao što su: glazba, filmovi, časopisi, knjige i igre. Ovisno o operacijskom sustavu te uređaju koji ga koristi tako na Google Playu imamo određeni spektar aplikacija koje uređaj može instalirati i pokrenuti, te one koje ne može. Na tržištu se pojavio 6. ožujka 2012 godine koji je zapravo bio spoj vise Googlovih servisa (Google Play Books, Google Play Games, Google Play Movies, Google Play Music, Google Play Newsstand) pod jedan unificirani naziv. 2016 godine Google Play trgovina imala je preko 82 milijarde preuzetih aplikacija te je u 2017 godini prešao broj od 3.5 milijuna aplikacija.

Glavna hardverska platforma za Android je ARM arhitektura sa x86 i MIPS arhitekturom također službeno podržanom u zadnjoj verziji Androida. Od 2012 na tržište se nalaze Android uređaji sa Intelovim procesorima koji su uključivali pametne telefone i tablete. Nakon izdavanja verzije 5.0 Lollipop uz prethodne 32 bitne verzije operativnog sustava pojavile su se i 64 bitne verzije.



Slika 4.3 Arhitektura Androida

Slika 4.3 prikazuje slojevitu arhitekturu. Kako bi se izvršavanje aplikacija na mobilnim uređajima moglo ispravno odvijati, slojevi moraju međusobno komunicirati.

Aplikacije koje se pokreću na samom uređaju definira aplikacijski sloj. One mogu biti sistemske i kao takve potencijalno potrebne svim drugim aplikacijama na uređaju, ali i aplikacije drugih proizvođača koje korisnik može skinuti sa prije spomenutog Google Play Servisa.

Za upravljanje aktivnostima, obavijestima, prozorima te svim ostalim resursima unutar sustava koristi se sloj aplikacijskog okvira.

Jezgrene biblioteke te java virtualni stroj čine sastavne dijelove Android okruženja. Ono se povezuje sa bibliotekama kako bi moglo pristupati funkcionalnostima samog sustava.

Najniži dio Androida platforme čini Linux jezgra koja je spomenuta u uvodu. Pogonski programi (engl. *drivers*) koji upravljaju radom sa samim hardverom mobitela koji koristi Android platformu se nalaze u jezgri. Jezgra je jedini dio sustava koji nije pisan u Java programskom jeziku te se piše sa programskim jezicima poput C i C++.

4.3.3. Xamarin

Xamarin je Microsoftov softver za razvoj mobilnih aplikacija. Kreiran je od strane istih inženjera koji su napravili Mono, Mono for Android i MonoTouch, koji su više platformske

implementacije CLI-a i .NET-a. Programerima je pomoću Xamarina omogućeno pisanje u jeziku C# te omogućava istovremeni razvoj aplikacija za operacijske sustave Android, iOS i Windows Phone. Prednost je i što podržava razvoj aplikacija na više od jednog operativnog sustava (Windows i macOS). Prema zadnjim podacima u svijetu, preko 1.4 milijuna programera koristi Xamarinove proizvode u 120 različitih zemalja svijeta (podaci iz Travnja 2017 godine).

4.3.3.1 Xamarin Test Cloud

Omogućava testiranje mobilnih aplikacija napisanih u bilo kojem jeziku na stvarnim uređajima u „oblaku“. Xamarin Test Cloud koristi objekt bazirani (engl. *object-based*) UI testiranje za simulaciju pravih korisničkih interakcija s aplikacijom.

4.3.3.2 Xamarin for Visual Studio

Jedini IDE koji omogućava razvoj Android, iOS i Windows Phone aplikacija unutar Microsoft Visual Studio alata. Xamarin pruža dodatke za Visual Studio koji omogućavaju programerima razvoj aplikacija na više platformi (Android, iOS, Windows Phone), te uz to mogućnost isporuke, traženja grešaka u razvoju te simulatora za testiranje.

4.3.3.3 Xamarin.Mac

Xamarin.Mac napravljen je kao alat za Appleove aplikacije koristeći C# kao programski jezik. Omogućava programerima zajedno sa Xamarin.iOS i Xamarin.Android da iskoriste čak do 90% prethodno napisanog koda za razvoj aplikacija na sve 3 platforme (Android, iOS, Windows Phone).

4.3.3.4 Xamarin.Forms

Predstavljeno u Xamarinu verziji 3. Omogućava jedan set prenosivih (engl. portable) kontrola koje se po standardu nalaze unutar operativnih sustava Android, iOS, Windows Phone.

4.3.4. Bootstrap

Bootstrap je besplatna i otvorenog koda (engl. *open-source*) biblioteka za razvoj prikaza (engl. *front-end*) web stranica i web aplikacija. Sadrži HTML i CSS predloške za tipografiju, forme, gumbe, navigaciju i ostale komponente prikazane na sučelju. Također ima opcionalne

JavaScript dodatke te je baziran strogo na prikaz (engl. *front-end*). Bootstrap je također drugi najocjenjivaniji projekt na GitHubu sa vise od 121.000 zvjezdica.

Bootstrap se originalno zvao Twitter Blueprint a razvili su ga Mark Otto i Jacob Thornton u Twitteru kao programski okvir (engl. *framework*) kako bi se održala konzistencija korištenja klasa i ostalih opcija koje nudi za interno korištenje unutar aplikacije. Prije Bootstrapa su korištene razne biblioteke koje su dovele do nekonzistentnosti i velikog tereta održavanja prikaza. Nakon par mjeseci razvoja mnogi programeri iz Twittera počeli su pridonositi Bootstrapu kao dio Hack Week događaja. Ubrzo je Twitter Blueprint preimenovan u Bootstrap i objavljen kao projekt otvorenog koda. 19. Kolovoza 2011. nastavili su ga održavati Mark Otto, Jacob Thornton i mala grupa programera kao i velika skupina ljudi što su također doprinijeli projektu.

Bootstrap 2 izdan je 31.1.2012, sadržavao je responzivnu mrežu od 12 kolona, podršku za ikonice (Glyphicons), neke nove komponente i izmjene na postojećima. 19.8.2013 izdan je Bootstrap 3 koji je uveo mobilno prvo (mobile first) pristup ka razvoju web dizajna.

29.10.2014 najavljen je Bootstrap 4 te je do sad izdano nekoliko verzija koje su još uvijek u razvoju. U principu je prepisana verzija 3 sa nekim značajnim izmjenama:

- Prijelaz sa less na Sass
- Izgubljena podrška za IE8, IE9 i iOS 6
- Dodana flexbox podrška i maknuta non-flexbox podrška
- Promjena sa pixela na em
- Povećana globalna veličina fotna sa 14px na 16px
- Maknute ikonice (Glyphicons)
- Maknuta komponenta za straničenje (pager component)
- Ponovno napisane sve komponente iz prethodne verzije, jQuery dodatci i dokumentacija

4.3.5. Klijentska biblioteka JQuery

JQuery je višeplatformska Javascript (engl. *JavaScript*) biblioteka dizajnirana da pojednostavi pisanje koda sa korisničke strane. JQuery je najpopularnija Javascript biblioteka u upotrebi koju koristi 65% od najposjećenijih web stranica na webu. JQuery je slobodan i softver otvorenog izvornog koda licenciran pod MIT licencom.

Sintaksa JQuery biblioteke je dizajnirana da pojednostavi navigaciju prema dokumentu, selektiranje DOM elemenata, izradu animacija, obrađivanje događaja i razvoj asinkronih (engl. Ajax) aplikacija. JQuery također pruža mogućnost programerima da naprave dodatke preko Javascript bilbioteke. Ovo omogućava programerima da naprave apstrakcije za interakciju niskog nivoa i animaciju, napredne efekte i dodatke visokog nivoa koji mogu imati teme. Modularni pristup JQuery biblioteci omogućava stvaranje moćnih dinamičkih web stranica i web aplikacija.

Komplet osnovnih mogućnosti JQuery biblioteke — selektiranje DOM (engl. DOM) elementa, promjena i manipulacija omogućena mašinom za selektiranje (nazvana Sizl (engl. *Sizzle*) iz verzije 1.3) kreirala je novi ”stil programiranja”, povezujući algoritme i strukture DOM podataka.

4.3.6. Klijentski programski okvir Angular 6

Angular 6 je besplatni klijentski programski okvir otvorenog koda baziran na Typescript jeziku (Typescript je programski jezik koji je superset Javascripta razvijen od strane Microsofta, a uvodi tipove i objektno orijentirano programiranje u Javascript svijet) za stvaranje klijentskih „Single page application“ aplikacija.

Razvijen je od strane Google-a, a s obzirom da je otvorenog koda razvoju su pridonijeli mnogi individualci i kompanije. Bazira se na uporabi komponenta koje možemo gledati kao Poglede unutar standardne MVC arhitekture. Komponente se sastoje od Html-a, CSS-a, te Typescript datoteka koje čine cjelinu. Separacija na posebne datoteke omogućava veću ponovnu uporabu te modularnost kao i separaciju odgovornosti.

Model predstavljaju Typescript klase koje su preslike JSON podataka koje dobivamo sa REST poslužitelja. Odgovornost za dohvat podataka se vrši unutar dediciranih servisa koji su preslike Upravitelja unutar MVC arhitekture.

Unutar servisa se obavlja i bilo kakva logika za manipuliranje podacima ili spremanje podataka u lokalni storage kako bi bili kasnije dostupni.

4.4. Poslužiteljski dio sustava

Poslužiteljski dio sustava kao što smo spomenuli u prethodnim poglavljima koristi ASP.NET Core 2 programski okvir te unutar njega korisitimo C# programski jezik. Za spremanje

korisničkih podataka koristimo SQL Server s kojim se komunikacija vrši pomoću Microsoftovog ORM-a[3] (engl. *Object-relational mapping*) Entity Framework. U slijedećim poglavljima ćemo detaljnije obraditi autentikaciju korisnika na Identity Serveru (4.3), dohvaćanje resursa sa poslužiteljske aplikacije (4.3) te način komuniciranja sa bazom podataka uz korištenje Entity Frameworka (4.3).

4.4.1. Identity Server autentikacija

Kako bi korisnik mogao pristupiti resursima na poslužitelju mora biti autenticiran. U poglavlju (4.3) smo naveli kako ASP.NET Core 2 omogućava razne vrste autentikacije te ga je lako spariti sa samim Identity Serverom. Sve što je potrebno jest postaviti da se koriste individualni korisnički računi te postaviti valjan konekcijski niz (engl. *connection-string*) koji će usmjeravati na bazu Identity Servera. Sam programski okvir se zatim korištenjem kolačića brine o autentikaciji i autorizaciji korisnika.

Android mobilna aplikacija autentikaciju korisnika riješava pomoću tokena. Kako bi mogao koristiti aplikaciju, korisnik mora unijeti valjano korisničko ime i lozinku koje šalju Identity Serveru te ako su uneseni podaci ispravni, Identity Server vraća token za pristup (engl. *access token*), token za osvježavanje (engl. *refresh token*) koji se koristi kako bi bi mogli osvježiti token za pristup ukoliko se zahtjev pošalje prije isteka vremena te samo vrijeme isteka tokena u sekundama od trenutka primitka, na primjer 3600 što znači da token vrijedi sat vremena.

```
POST /connect/token HTTP/1.1
Host:
http://culturetrackerauthorizationserver20180114114704.azurewebsites.net
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Cache-Control: no-cache

username=and%40gmail.com&password=C%40blugY1&grant_type=password&client_id=ropcclient&client_secret=password
```

[3] ORM – ovi su sustavi za vršenje komunikacije sa bazama podataka na način da se tablice iz baze mapiraju na klase koje predstavljaju jedan redak tablice

Kôd 4.3 Primjer dohvaćanja tokena HTTP zahtjevom

Kôd 4.4 nam daje primjer zatjeva prema Identity Serveru za dohvaćanje tokena. Kako je vidljivo iz zahtjeva potrebno je namjestit valjani URL (engl. *Uniform Resource Locator*) do samog Identity Servera, zatim koju vrstu formata podataka klijent očekuje kao odgovor (zaglavlje `accept`) te tip sadržaja koji klijent šalje (zaglavlje `Content-Type`). Nadalj,e definira se tip sadržaja koji klijent šalje (zaglavlje `Content-Type`) te na koncu korisnički podaci sa defirniranim identifikacijskim nizom i lozinkom za klijenta što nam omogućava postavljanje različitih pravila ovisno o klijentu.

4.4.2. REST sučelje i arhitektura

Kako smo naveli u prethodnim poglavljima, poslužiteljska aplikacija je izgrađena prema REST arhitekturalnom modelu. Ona izlaže klijentima sučelja kako bi mogli pristupiti i koristiti resurse i funkcije aplikacije. Da bi klijent mogao pristupiti traženim resursima mora biti autenticiran i autoriziran čime postižemo sigurnost aplikacije.

Sama arhitektura koda poslužiteljske aplikacije je napravljena u slojevima kako bi se razdvojile brige (engl. *separation of concerns*), to znači da je svaki sloj zadužen i obnaša točno određenu ulogu u sustavu, a time dobivamo modularnost te mogućnost drukčijih implementacija unutar samih slojeva te nam omogućava da ponovno koristimo te slojeve u drugim aplikacijama. Aplikacije sa slojevitom arhitekturom su lakše za održavanje i ažuriranje zbog prije navedene separacije briga.

Unutar izvršnog sloja aplikacije se nalaze kontroleri koji su zaduženi za komunikaciju sa klijentima. Svaki kontroler ima posebno definirane rute (engl. *routes*) koji specificiraju put do resursa kojima taj kontroler upravlja. Rute se mogu definirati na svakoj akciji kontrolera, no pošto mi koristimo REST arhitekturalni model, moramo definirati rute preko postojećih HTTP metoda kao što je navedeno u poglavlju (4.3). Svaki kontroler na sebi ima dva atributa[4] pomoću kojih definiramo dodatna svojstva. Prvi atribut `[Route ("api/Route")]` definira rutu kako bi se moglo pristupiti resursima i funckijama na kontroleru. Drugi atribut `[Authorize]` definira da je za pristup svim resursima i funkcijama unutar kontrolera potrebna autentikacija korisnika.

[4] Atributi su zasebne klase u C# programskom jeziku pomoću kojih definiramo dodatna svojstva samih klasa u kojima se koriste, svojstvima (engl. *properties*) i metodama iznad kojih su definirani.

Svaki kontroler u sebi ima servis koji je zadužen za baratanje podacima te je on taj koji komunicira sa bazom podataka preko Entity Framework ORM – a. Svi servisi su izgrađeni kao sučelja te korištenjem tehnike ubacivanja ovisnosti (engl. *dependency injection*) instancirani od strane samog programskog okvira. Ubacivanje ovisnosti nam omogućuje da na jednom mjestu definiramo koje implementacijske klase će se koristiti kao implementacija za svako sučelje te nam time omogućava da po potrebi mijenjamo implementaciju. ASP.NET Core 2 u sebi ima ugrađene mehanizme za ubacivanje ovisnosti te se mapiranje između sučelja i implementacijske klase vrši pri pokretanju aplikacije, odnosno definiramo koja implementacijska klasa će se koristiti za pojedino sučelje.

```
services.AddScoped<IRouteService, RouteService>();
```

Kôd 5.4 Primjer definiranja ovisnosti za sučelje IRouteService

Slijedeće ćemo demonstrirati način dohvata pojedinih korisničkih ruta.

```
GET /api/Route
Authorization: bearer bGGhJZ...
Content-Type: application/json
```

Kôd 4.4 Primjer HTTP zahtjeva za dohvat rute

Kôd 4.4 nam demonstrira zahtjev za dohvaćanje ruta prijavljenog korisnika. Prvo zaglavlje se odnosi na putanju do resursa, odnosno URL zahtjev prema resursu na lokaciji `/api/Route`. Kako smo prethodno napomenuli da se za pristup poslužiteljskoj aplikaciji koriste tokeni, drugo zaglavlje `Authorization` postavlja vrijednost primljenog tokena od strane Identity Servera. Na koncu imamo zaglavlje `Content-Type` koje definira koji format podataka klijent očekuje kao odgovor. Odgovor na zahtjev je niz objekata tipa RouteDTO koji sadržavaju sve informacije vezane uz pojedinu rutu.

```
[HttpGet]
public async Task<IActionResult> Get(){
    var result = await
this._routeService.GetAll(this.CurrentUserId);
```

```
        return this.FormatResult(result);}
```

Kôd 4.5 Metoda za dohvaćanje ruta

```
        [HttpGet]
    public async Task<IActionResult> Get(){
            var result = await
this._routeService.GetAll(this.CurrentUserId);
        return this.FormatResult(result);}
```

Kôd 4. prikazuje metodu koja se koristi za dohvaćanje ruta. Iz primjera vidimo da kontroler šalje zahtjev na obradu servisu te nakon toga formatira odgovor od strane servisa. Uzima objekte koje primio kao odgovor te zajedno s njima šalje svojstvo koje označuje da li je zahtjev uspješno izvršen, svojstvo je HTTP statusni kod (primjer `200 OK`), a u slučaju da nije šalje poruku u kojoj je definirano što je pošlo krivo zajedno sa potencijalnim objektom koji je prouzročio grešku.

4.4.3. Komuniciranje s bazom podataka

Kako smo u prethodnom poglavlju (4.4.2) objasnili i demonstrirali (`[HttpGet]`

```
    public async Task<IActionResult> Get(){
            var result = await
this._routeService.GetAll(this.CurrentUserId);
        return this.FormatResult(result);}
```

Kôd 4.), svaki kontroler ima servis ubačen tehnikom ubacivanja ovisnosti koji je zadužen za komunikaciju s bazom podataka.

```
public async Task<IServiceResult<IEnumerable<RouteDto>>> GetAll(string
                        identityId){
                            try{
    var route = await this._unitOfWork.RouteRepository.Find(x=>
              x.User.IdentityId == identityId);
    var routeDtos = this._mapper.Map<IEnumerable<RouteDto>>(route);
            return this._resultFactory.CreateList(routeDtos);}
                    catch (Exception exception){
                return this._resultFactory.CreateList(null,
    this.CreateErrorLog(exception), ServiceStatus.Fail);}}
```

Kôd 4.6 Metoda za dohvaćanje ruta unutar servisa

Kôd 4. prikazuje metodu unutar servisa koja je zadužena za dohvaćanje korisnikovih ruta. Metoda kao parametar prima identifikator korisnika te koristi vlastito svojstvo `[_unitOfWork]` preko kojeg komunicira sa bazom. Sama implementacija svojstva je učinjena preko jedinice rada programskog obrasca (engl. *Unit of Work*) koja grupira jednu ili više operacija prema bazi u jedinstvenu transakciju. Nakon što dohvati podatke iz baze mapira ih u objekt tipa RouteDTO koji se zatim dodatno omota (engl. *wrap*) sa informacijama o tome da li je operacija uspješno izvršena, a u slučaju da nije, vraća razlog zbog kojeg nije.

```
public virtual async Task<IEnumerable<TEntity>>
Find(Expression<Func<TEntity, bool>> predicate){
                        return await
Context.Set<TEntity>().Where(predicate).ToListAsync();}
```

Kôd 4.7 Metoda za dohvaćanje ruta unutar jedinice rada

Kôd 4. prikazuje metodu unutar jedinice rada koja je zadužena za dohvaćanje korisničkih ruta uporabom ORM-a. Dohvaćanje ruta se vrši prema poslanom predikatu koji je definiran u pozivajućoj klasi, u našem slučaju to je servis kako smo demonstrirali u Kôd 4.. Prednosti korištenja ORM – a su vidljivi u ovoj metodi, naime radimo sa klasama koje predstavljaju redak tablice unutar baze te nemamo zapravo osjećaj da se u pozadini obavlja SQL upit na bazu te da se vraćeni podaci automatski mapiraju u specificiranu klasu.

Jedinica rada u sebi ima definirano svojstvo `Context` koje omogućava komunikaciju sa bazom. Radi se o svojstvu tipa `DbContext` koje je most između domenskih klasa i same baze podataka u Entity Framework ORM–u.

```
public class Route{
public int Id { get; set; }
public int SearchRadius { get; set; }
public string Name { get; set; }
public string Description { get; set; }
public bool IsFinished { get; set; }
public DateTime Start { get; set; }
public DateTime Finish { get; set; }
public CultureTrackerUser User { get; set; }
public List<RouteEntry> Entries { get; set; }}
```

Kôd 4.8 C# klasa koja definira rutu u bazi podataka

Kôd 4. nam demonstrira klasu u C# programskom jeziku koja definira domenski model prema kojem će biti stvorena tablica u bazi podataka korištenjem ORM-a. Naziv klase i tablice je u ovom slučaju ekvivalentan iako se mogu postaviti drugačije vrijednosti na oba kraja, ali u tom slučaju je potrebno definirati atribut `Table` kojim govorimo ORM-u ime tablice unutar baze podataka. Tipovi svojstva unutar klase se preslikavaju u odgovarajuće tipove unutar baze dok se za imena atributa također koristi konvecija, odnosno ako nije drugačije definirano oni su ekvivalentni onima definiranim u samoj klasi. Iz ovog modela možemo vidjeti i primjer korištenja primarnog i stranih ključeva, primarni ključ je svojstvo `Id` dok su strani ključevi svojstva `Id` unutar klasa `User` i `Entries`. Entity Framework nam pruža mogućnost dohvaćanja i klasnih objekata koja su svojstva u nekoj drugoj klasi korištenjem `Include` parametra pri pozivu na bazu.

4.5. Klijentski dijelovi sustava

Klijentske dijelove sustava ćemo obraditi u slijedećim poglavljima. Prvo ćemo obraditi Android aplikaciju, njezinu arhitekturu i grafičko sučelje te ćemo nakon toga isto učiniti za web aplikaciju.

4.5.1. Arhitektura mobilne aplikacije

Mobilna aplikacija je izrađena u okvirima standarnog razvoja mobilnih aplikacija za Android platformu. Koristimo arhitekturalni model MVP koji je detaljnije obrađen u ranijim poglavljima (4.4.2). Za bazu podataka koristimo SQLite koji je kao i Entity Framework implementacija u poslužitelju učinjen preko programskog obrasca jedinice rada. Za komunikaciju sa poslužiteljem koristimo HTTP Client klasu koja je omotana klasom koja nam nudi sve potrebne akcije za pristup traženim resursima.

Arhitekturalno gledano., aplikacija prati MVP model prema najboljim praksama za izradu aplikacija za Android platformu. U našem slučaju Pogled predstavljaju aktivnosti (engl. *activities*) i fragmenti (engl. *fragments*). Oboje implementiraju sučelja koja sadržavaju metode koje će se pozivati iz Prikazivača. Sam prikazivač koristi Model kada treba dohvatiti određeni resurs ili akciju. Modeli su obične klase sa poslovnom logikom ili Android servisi koji obavljaju potrebne akcije na drugoj niti od glavne niti. Nakon izvršavanja zahtjevanih

radnji, ako je potrebno osvježiti pogled Prikazivač zove metodu na sučelju kako bi se to učinilo.

Prethodno smo napomenuli da koristimo HTTP Client klasu za komunikaciju sa poslužiteljem koja je omotana klasom naše radinosti. Omotavajuća klasa se zove ApiClient i uz CRUD operacije na poslužitelju također je zadužena za logiranje korisnika, odnosno dohvaćanje tokena od strane Identity Servera te za obnovu tokena. Vraća objekt koji je sličan onome na poslužitelju, taj objekt ima svojstva koja definiraju odgovor poslužitelja te su mapirana na implementirajuće klase te skupa sa njima javlja da li je operacija uspješno izvedena te ako nije vraća poruku koja predstavlja razlog zbog kojeg operacija nije uspjela.

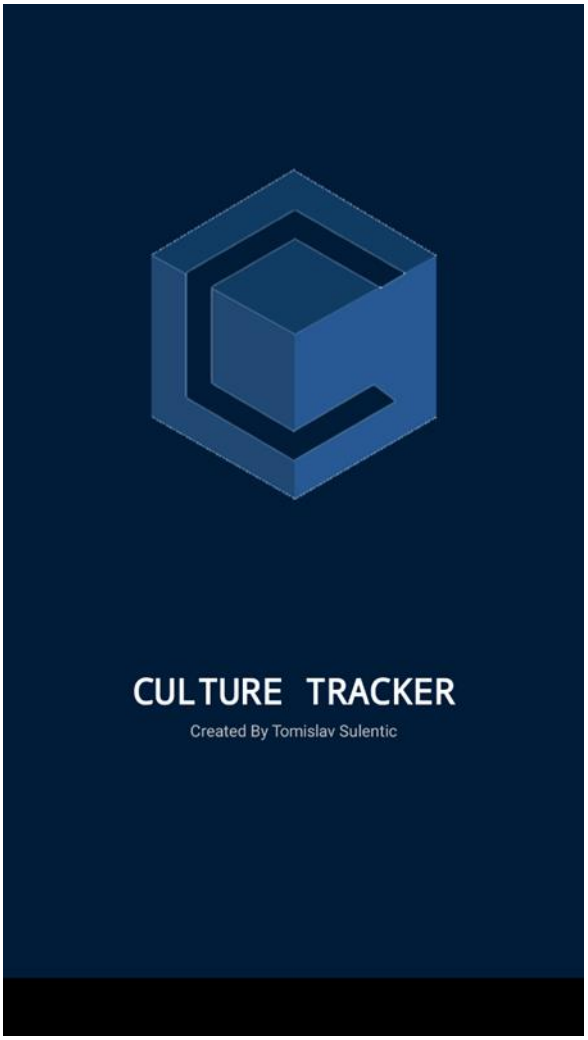
```
public async void StopTrackingLocation(){
                    _locationHelper.RemoveUpdates();
   var currentLogin = await _uow.LoginInfoRepository.GetCurrentLogin();
var _apiClient = new ApiClient(ConfigurationManagerHelper.ApiEndpoint);
                _apiClient.SetAccessToken(currentLogin.Token);
                    var route = _routesHelper.CurrentRoute;
                         route.Finish = DateTime.Now;
                     route.UserEmail = currentLogin.Email;
                  route.Name = DateTime.Now.ToLongDateString();
               var payload = JsonConvert.SerializeObject(route);
             await _apiClient.Post<RouteDto>(payload,
      ConfigurationManagerHelper.ApiEndpoint + "api/route");}
```

Kôd 4.5 Primjer metode unutar Android servisa za kreiranje rute na poslužitelju

Kôd 4.5 demostrira metodu za završetak praćenja trenutne korisnikove rute te slanje pripadajućih podataka na poslužitelj gdje se spremaju u bazu. Prvotno se poziva metoda `RemoveUpdates` koja govori Android platformi da prestane s uporabom lokacijskih servisa za praćenje korisnikove pozicije. Nakon toga se dohvaća trenutni korisnik aplikacije iz jedinice rada te instancira novi objekt tipa `ApiClient` kojeg smo opisali u ovom poglavlju. Taj objekt prvotno postavlja token trenutnog korisnika koji je postavljen pri fazi autentikacije u aplikaciju te nakon toga postavlja svojstva trenutne rute koja su ovisna o trenutnom korisniku i vremenu u kojem je ruta završena. Posljednji koraci uključuju pretvorbu objekta rute u JSON objekt koji se zatim šalje na poslužitelj na daljnju obradu.
Jedinica rada nam služi za pristup podacima o korisniku, njegovom trenutnom tokenu te se kao baza podataka koristi SQLite. Navedena se koristi preko SQLite .NET ORM-a koji je moguće koristiti na raznovrsnim platformama te nativno podržava Android platformu.

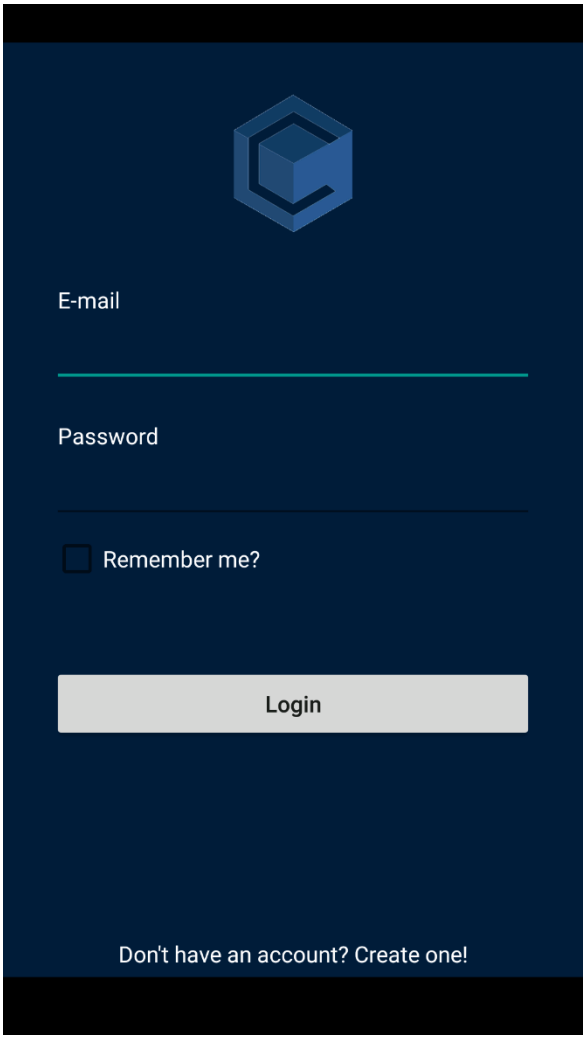
4.5.2. Grafičko sučelje mobilne aplikacije

Kako smo u poglavlju 4.4.2 (Funkcionalnosti sustava) opisali sve mogućnosti android mobilne aplikacije, u nastavku ćemo detaljnije obraditi sve dijelove te aplikacije i prikazati korisničko sučelje. Samo sučelje je rađeno prema uputama za izradu grafičkih sučelja na Android mobilnoj platformi koje se zove Google Material Design te se koriste nativne komponente sustava kako bi korisnik imao lakše vrijeme prilagodbe te jednostavnije korištenje [7]. Primarna svrha Android mobilne aplikacije je lociranje korisnika u prostoru na ruti koju sam korisnik pokreće i zaustavlja te se tada podaci o korisnikovim lokacijama u tom vremenu šalju na poslužitelj gdje se spremaju u bazu podataka.



Slika 4.4 Splash ekran Android aplikacije

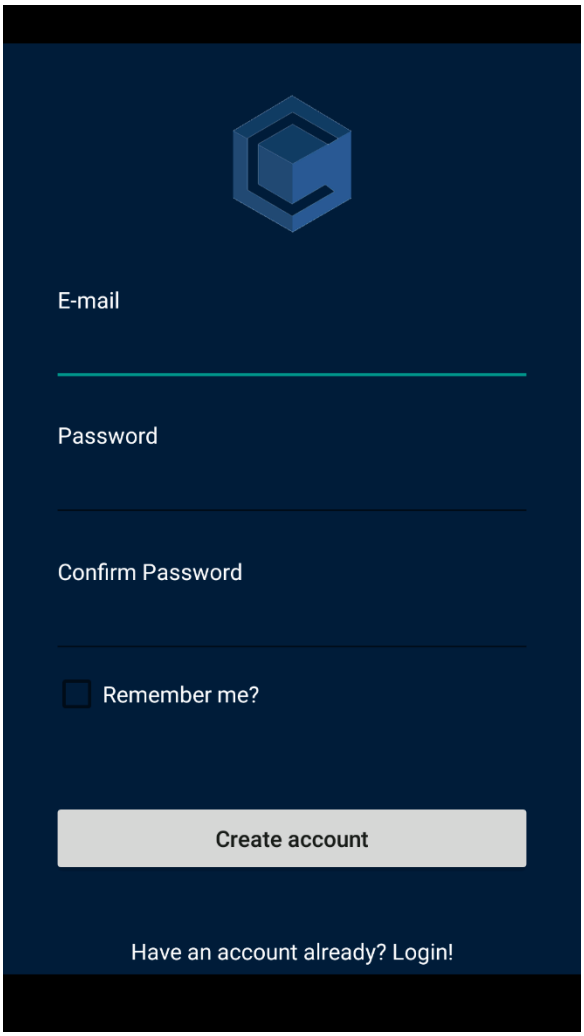
Slika prikazuje *splash* ekran koji dočekuje korisnika pri pokretanju aplikacije na kojem je prikazan logo aplikacije te ime autora ovoga rada. Ako je korisnik već bio logiran u aplikaciju i ako je pritom označio da se zapamte njegovi korisnički podaci, aplikacija povlači te podatke iz lokalne baze podataka te automatski vrši autentikaciju. U suprotnom, korisnik mora unijeti podatke kako bi se logirao.



Slika 4.5 Ekran za prijavu Android aplikacije

Slika prikazuje ekran za prijavu koji se korisniku prikazuje u slučaju da se korisnik nije prethodno autenticirao ili izabrao opciju da se zapamte njegovi korisnički podaci, u kojem je potrebno unijeti email adresu koja predstavlja korisničko ime te pripadajuću lozinku.

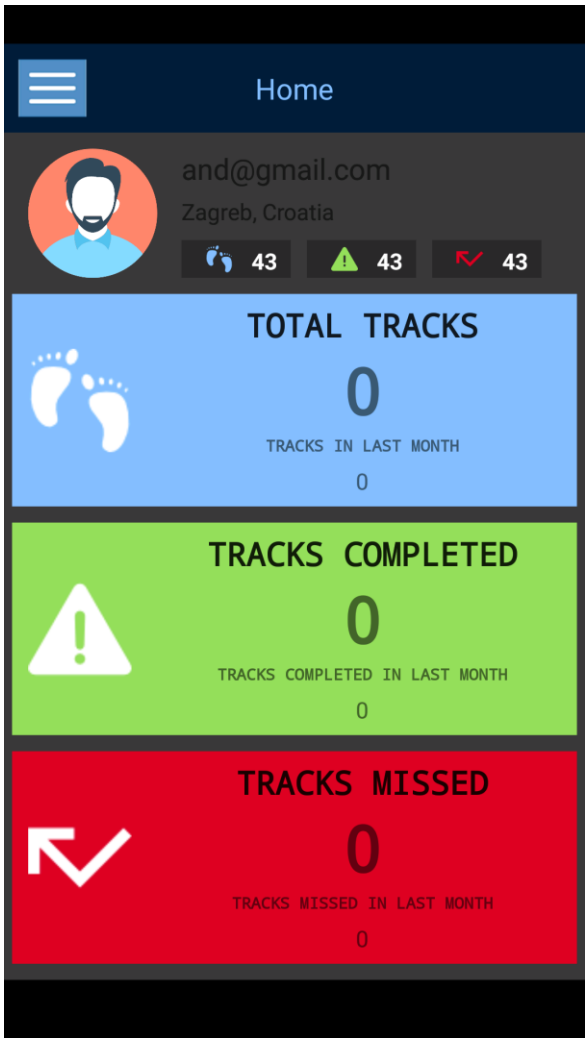
Na dnu ekrana se nalaze dva gumba, prvi služi za potvrdu prijave sa unesenim korisničkim podacima, dok drugi služi za odlazak na ekran za stvaranje novog korisničkog računa u slučaju da korisnik nema postojeći račun.



Slika 4.6 Ekran za stvaranje korisničkog računa Android aplikacije

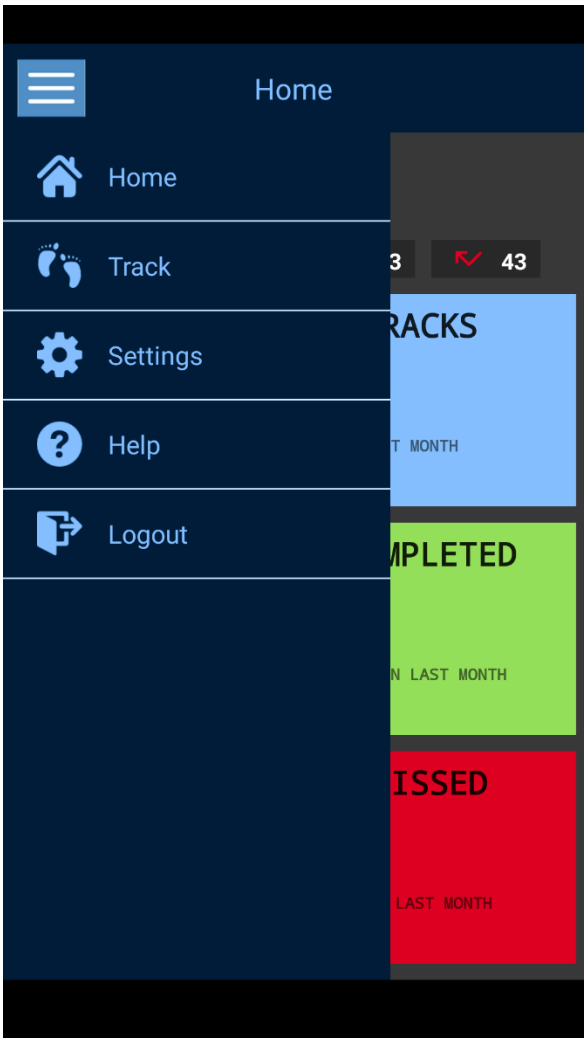
Slika prikazuje ekran za stvaranje korisničkog računa u kojem je potrebno unijeti email adresu koja se koristi kao korisničko ime, lozinku za pripadajući email te potvrda lozinke kako bismo bili sigurni da je korisnik zaista htio unijeti navedenu lozinku. Ovi korisnički podaci se koriste u obje klijentske aplikacije te time omogućavaju korisniku da se prijavi na jednom mjestu te koristi iste podatke u obe aplikacije.

Na dnu ekrana se nalaze dva gumba, prvi je za stvaranje korisničkog računa kada je korisnik zadovoljan podacima koji je unio, a na samom dnu je opcija da se vrati na login ekran u slučaju da ima postojeći korisnički račun s kojim se može valjano prijaviti u aplikaciju.



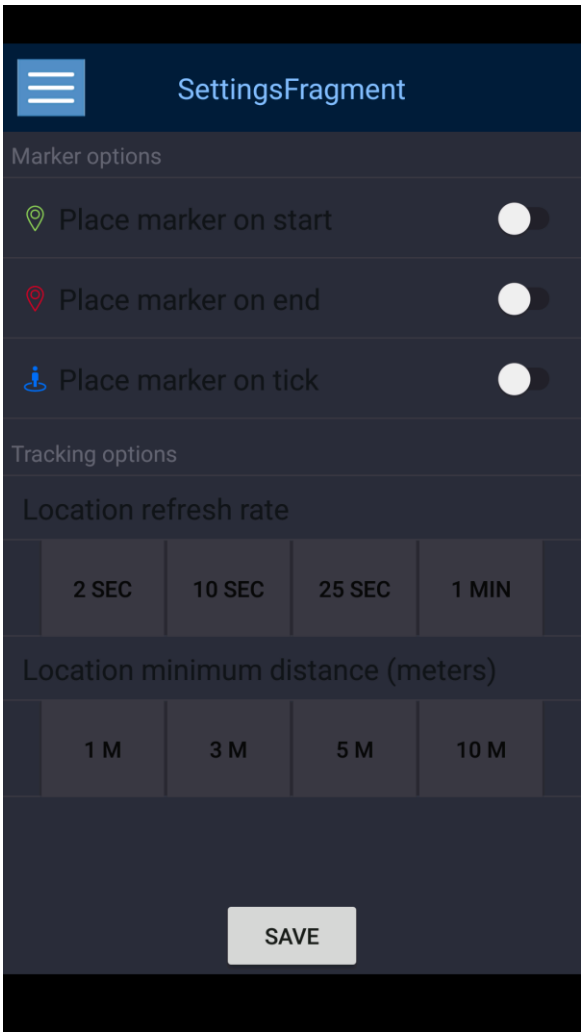
Slika 4.7 Početni ekran aplikacije

Slika prikazuje početni ekran aplikacije. Na njemu su prikazani osnovni podaci o korisniku te korisnikov avatar. Podaci i avatar se dijele između mobilne aplikacije i web aplikacije, a postavke za namještanja tih podataka se nalaze na samoj web aplikaciji te se propagiraju pri pokretanju na Android mobilnu aplikaciju. Također je prikazan broj totalnih praćenih ruta, ruta koje je korisnik označio završenim, te rutama koje korisnik još nije označio kao završene. Također se prikazuje broj ruta za svaku od prije navedenih kategorija u prethodnom mjesecu, što korisniku daje na uvid koliko je često koristio aplikaciju u tom periodu. Na vrhu ekrana se nalazi traka za navigaciju koja na pritisak gumba otvara opcije za navigiranje kroz aplikaciju.



Slika 4.8 Navigacija unutar aplikacije

Pogreška! Izvor reference nije pronađen. prikazuje navigaciju unutar aplikacije. Opcije koje se nude su početni ekran, ekran za praćenje korisnikove rute, ekran za postavke, ekran za pomoć te opcija za odjavljivanje iz aplikacije.

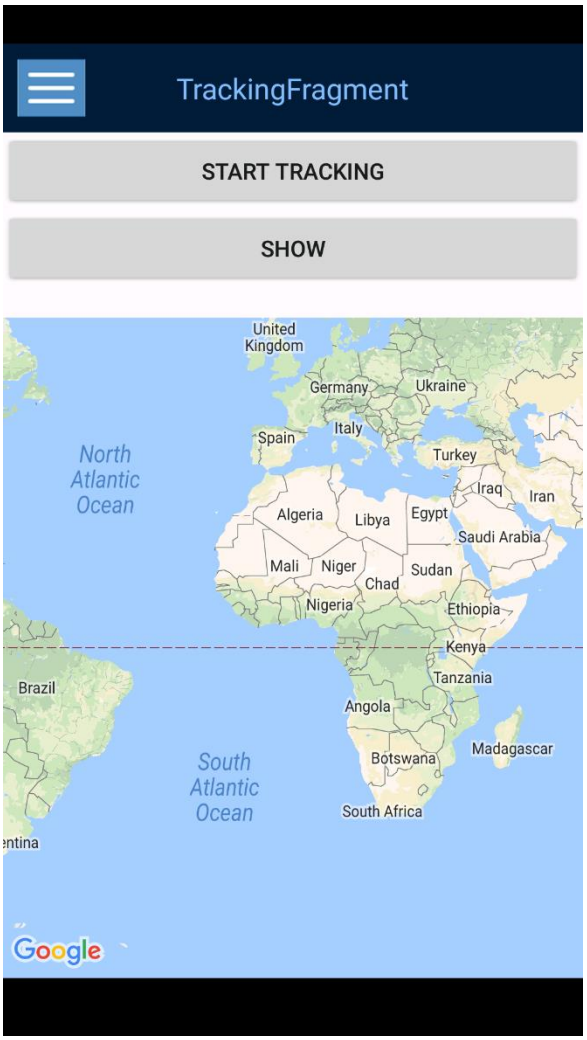


Slika 4.9 Ekran sa postavkama

Pogreška! Izvor reference nije pronađen. prikazuje ekran sa postavkama vezanim uz aplikaciju. Moguće postavke su postavljanje markera na početak i kraj rute, te postavljanje markera pri svakom događaju unutar Android lokacijskog servisa koji prati korisnikovu lokaciju, odnosno pri svakom zapisu korisnikove lokacije.

Nakon sekcije za markere imamo sekciju za postavljanje opcija oko samog lociranja korisnika. Moguće je postaviti vrijednosti za minimalno vrijeme između osvježavanja korisnikove lokacije, odnosno koliko često će se pratiti korisnikova lokacija na njegovoj zadanoj ruti. Potom imamo postavke vezane uz minimalnu distancu koju korisnik mora proći kako bi se zabilježila promjena njegove lokacije.

Na dnu ekrana se nalazi gumb koji omogućava korisniku da spremi namještene postavke u lokalnu bazu podataka kako bi se te postavke mogle koristiti pri sljedećoj njegovoj ruti.



Slika 4.10 Ekran za kreiranje ruta

Pogreška! Izvor reference nije pronađen. prikazuje ekran za stvaranje korisničke rute. Početno se prikazuje Google Mapa sa dva gumba na vrhu ekrana. Prvi omogućava početak praćenja korisnikove lokacije te samim time stvaranje nove rute. Nakon što korisnik pritisne prvi gumb *zoomira* se na trenutnu korisnikovu lokaciju te ako je u postavkama tako zadano stavlja se marker na tu lokaciju. Također se mijenja tekst gumba u „Stop tracking“. Drugi gumb omogućava prikaz broja do sada dobivenih lokacija na trenutnoj ruti od strane Android servisa za praćenje lokacije. Nakon što je korisnik odlučio da je njegova ruta gotova, ponovno pritišće prvi gumb te se time ta ruta smatra završenom, prestaje se sa praćenjem korisnikove pozicije te se šalju podaci na poslužitelj, a korisnik detaljnije informacije o ruti može dobiti na web aplikaciji.

4.5.3. Arhitektura web aplikacije

Web aplikacija je izrađena u Angular 6 programskom okviru uporabom MVC arhitekturalnog modela. Sastoji se od početne web stranice na kojoj se nalazi kratki pregled korisnikovih ruta, sa grafičkim prikazima i tablicama te statistikama.

Samoj web aplikaciji se pristupa uporabom valjanog korisničkog imena i lozinke ili u slučaju da korisnik navedeno nema, može stvoriti novi račun.

Arhitekturalno gledano sama aplikacija je izvedena po MVC modelu kojem smo detaljno opisali u poglavlju (4.3) što je ekvivalento početnim postavkama pri kreiranju projekta unutar Angular 6 programskog okvira.

Sama aplikacija se sastoji od preko 10 komponenti koje korespondiraju stranicama unutar aplikacije. Također koristi ekvivalentan broj servisa te autentifikacijski zaštitnik koji vrši komunikaciju sa Identity Serverom te postavlja token koji aplikacija koristi za komuniciranje sa poslužiteljskom aplikacijom te kolačiće koji se koriste za autentikaciju same web aplikacije.

```
login(username: string, password: string) {
let header = new HttpHeaders({ 'Content-Type': 'application/x-
                www-form-urlencoded' });

                let params = new HttpParams()
                  .append('username', username)
                  .append('password', password)
                .append('grant_type', 'password')
                .append('client_id', 'ropcclient')
               .append('client_secret', 'password')
                .append('grant_type', 'password')
        .append('scope', 'CultureTracker.API offline_access');

              let requestBody = params.toString();
return this.http.post<any>(`${config.authUrl}/connect/token`,
          requestBody, { headers: header })
```

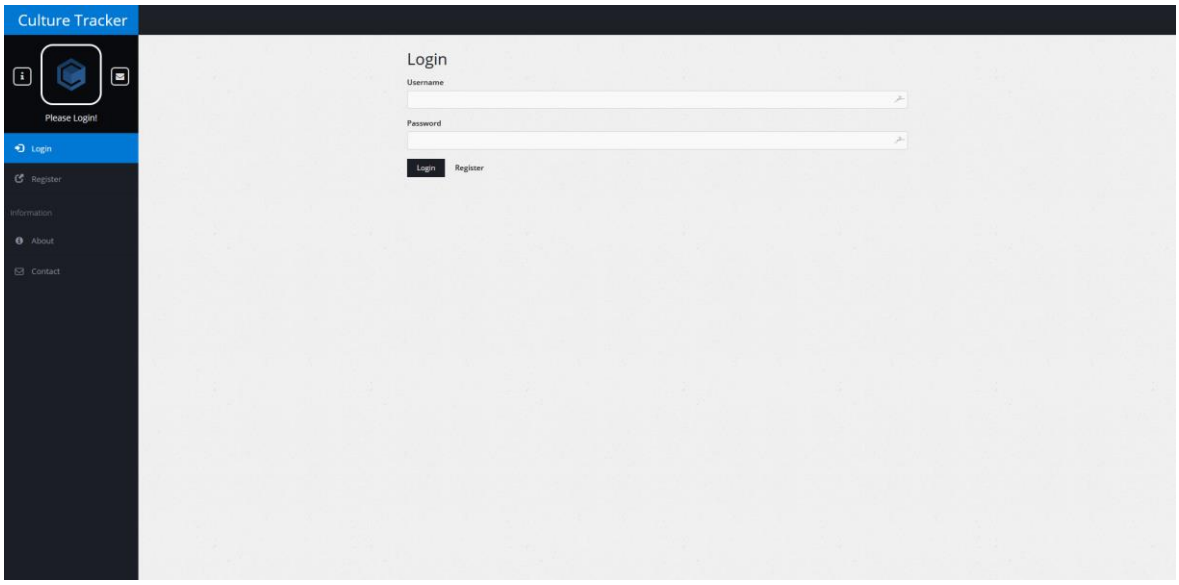
```
            .pipe(map(response => {
        if (response && response.access_token) {
                    var user = new User();
                  user.username = username;
                  user.password = password;
              user.token = response.access_token;

                  this.loggedIn.next(true);
              localStorage.setItem('currentUser',
        JSON.stringify(user));
                            }
                    return response;
                        }));
                    }
```

Kôd 5.6 Primjer postavljanja kolačića i tokena pri uspješnoj prijavi

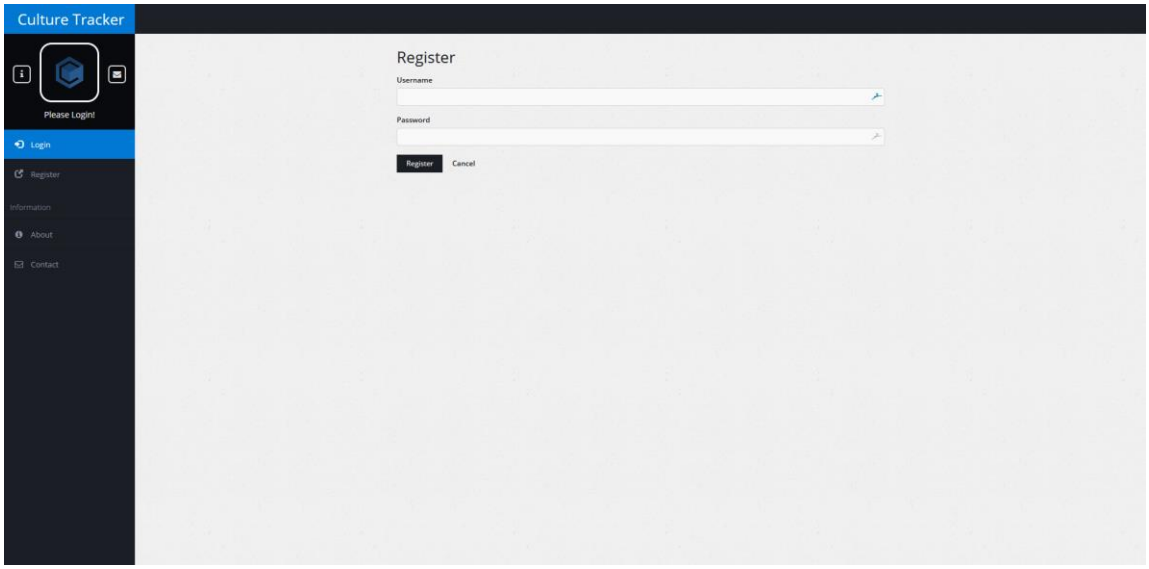
4.5.4. Grafičko sučelje web aplikacije

U poglavlju (4.3) smo definirali mogućnosti web aplikacije te ćemo u ovome poglavlju detaljnije obraditi njezino grafičko sučelje.



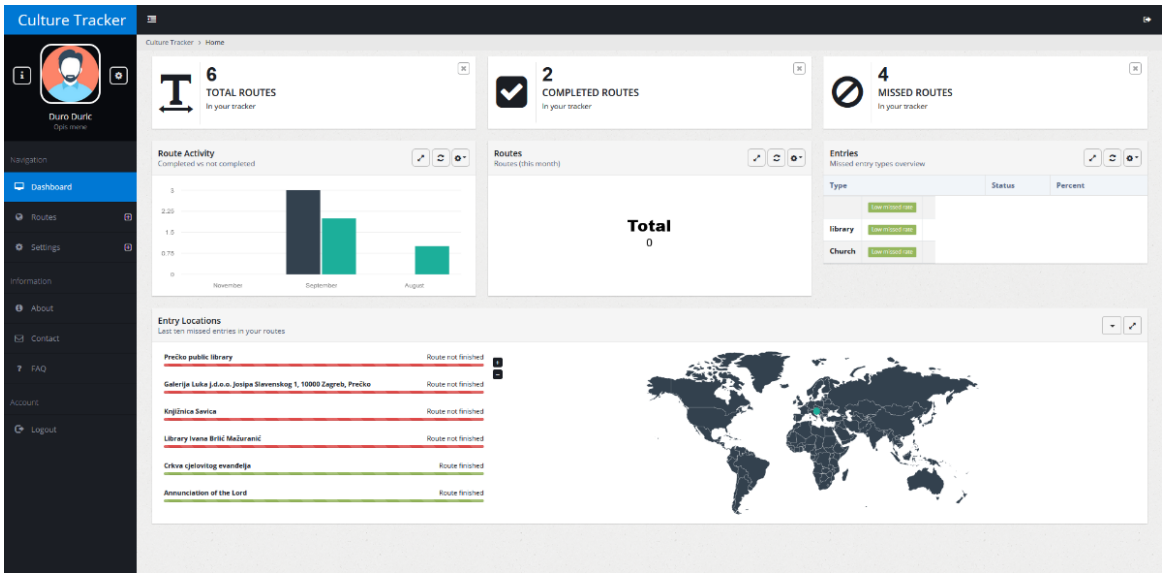
Slika 4.11 Stranica za autentikaciju

Pogreška! Izvor reference nije pronađen. prikazuje stranicu za autentikaciju te navigaciju unutar web aplikacije za korisnika koji nije prijavljen u sustav. Ako korisnik nije autenticiran, nudi mu se stranica za unos korisničkog imena i lozinke te poveznice na stranicu za registraciju. Sekcija „About“ vodi na stranicu na kojoj je kratki opis aplikacije te njenog autora, dok sekcija „Contact“ vodi na stranicu na kojoj su informacije za kontaktiranje autora aplikacije u slučaju na nalaženje greške u izvođenju aplikacije ili bilo kakvih drugih upita.



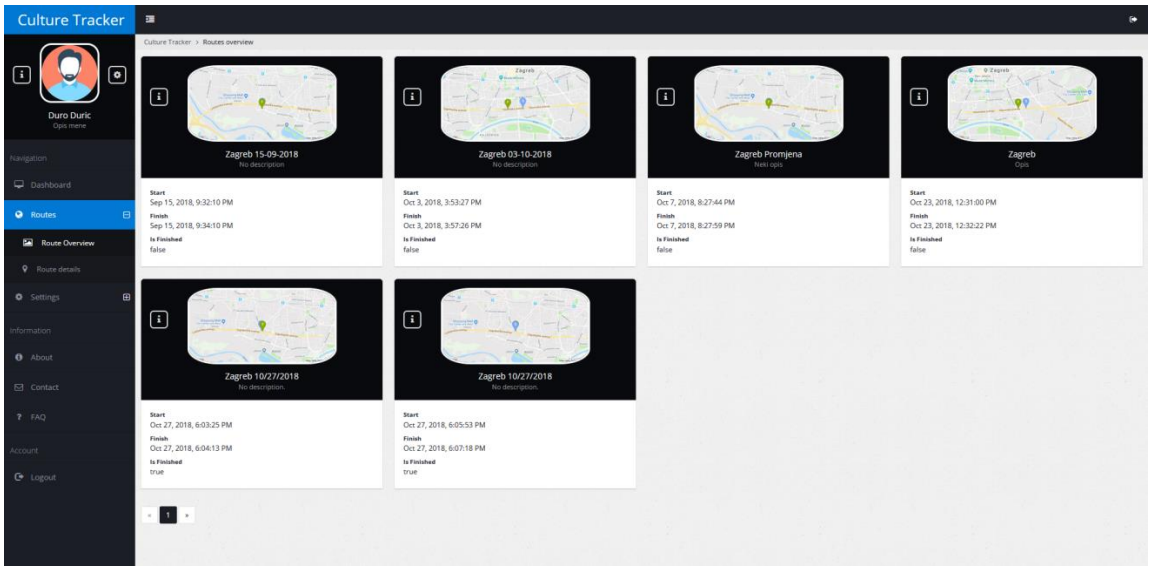
Slika 4.12 Stvaranje korisničkog računa unutar Web aplikacije

Pogreška! Izvor reference nije pronađen. prikazuje stranicu za stvaranje novog korisničkog računa. Pri unosu email-a te lozinke vrši se validacija unesenih podataka. To sprječava unos nevaljanog emaila te slabe lozinke.



Slika 4.13 Početna stranica unutar aplikacije

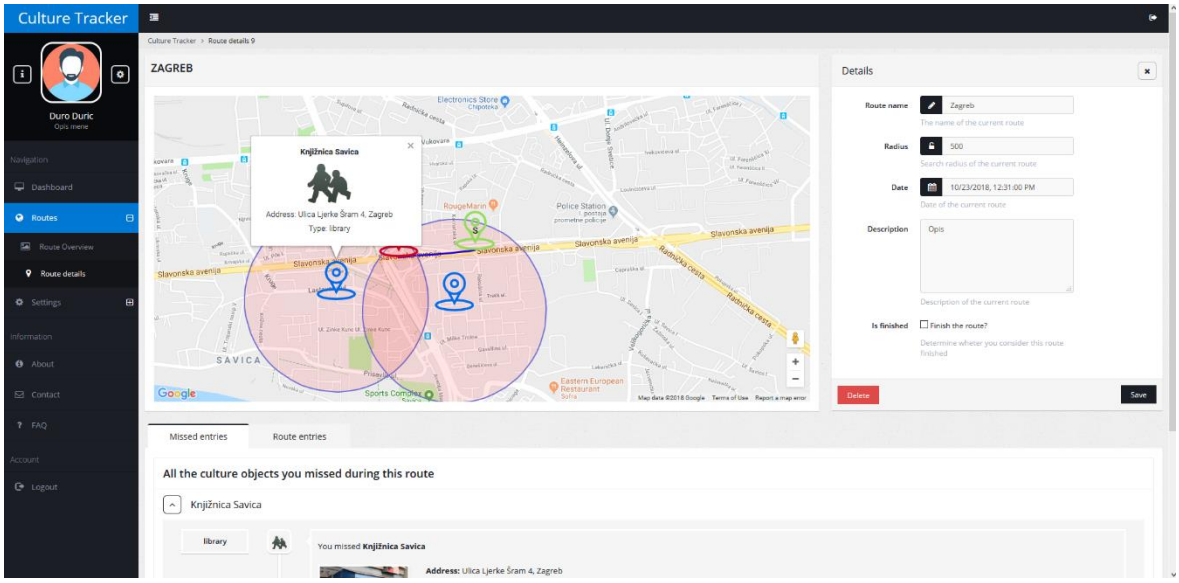
Pogreška! Izvor reference nije pronađen. prikazuje početnu stranicu u aplikaciji. Unutar nje su podaci o ukupnom broju korisnikovih ruta, broj završenih ruta, te broj propuštenih ruta na samome vrhu stranice. U srednjem segmentu stranice se nalazi grafički prikaz o odnosu završenih i propuštenih ruta unutar zadnja tri mjeseca, unutar zadnjih mjesec dana, te statistički prikaz o tipu propuštenih objekata. Na dnu se nalaze zadnjih deset propuštenih objekata.



Slika 4.14 Pregled ruta unutar Web aplikacije

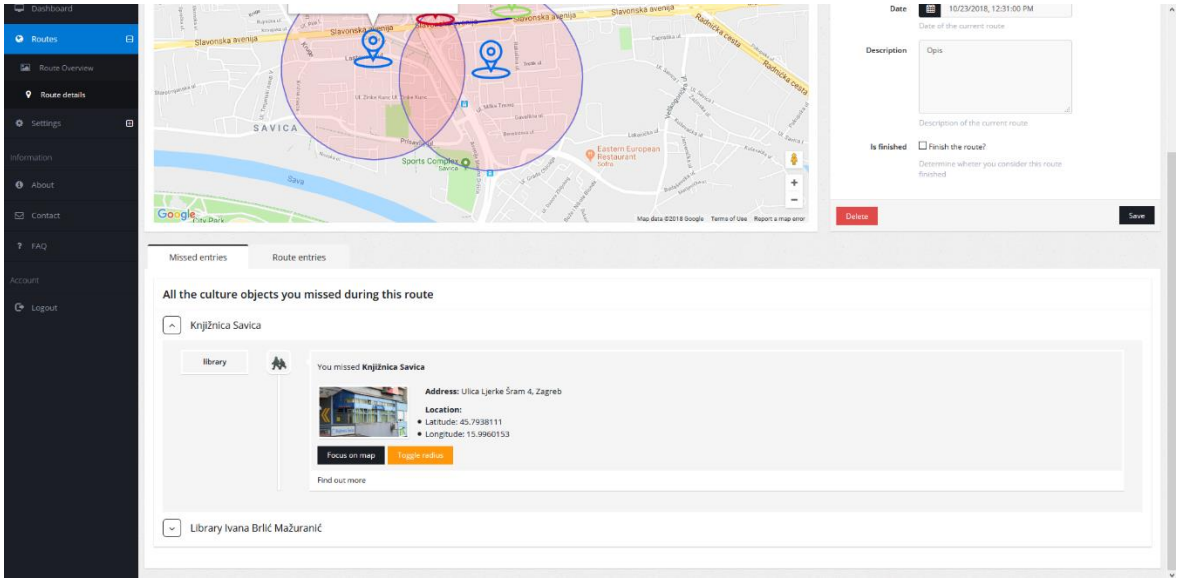
Pogreška! Izvor reference nije pronađen. prikazuje stranicu za pregled ruta. U njoj su vidljive sve korisnikove rute zajedno sa kratkim pregledom njihovih podataka. Nudi opciju

odlaska na stranicu sa detaljima za tu specifičnu rutu te, ako je broj ruta veći od 20, uvodi paginaciju.



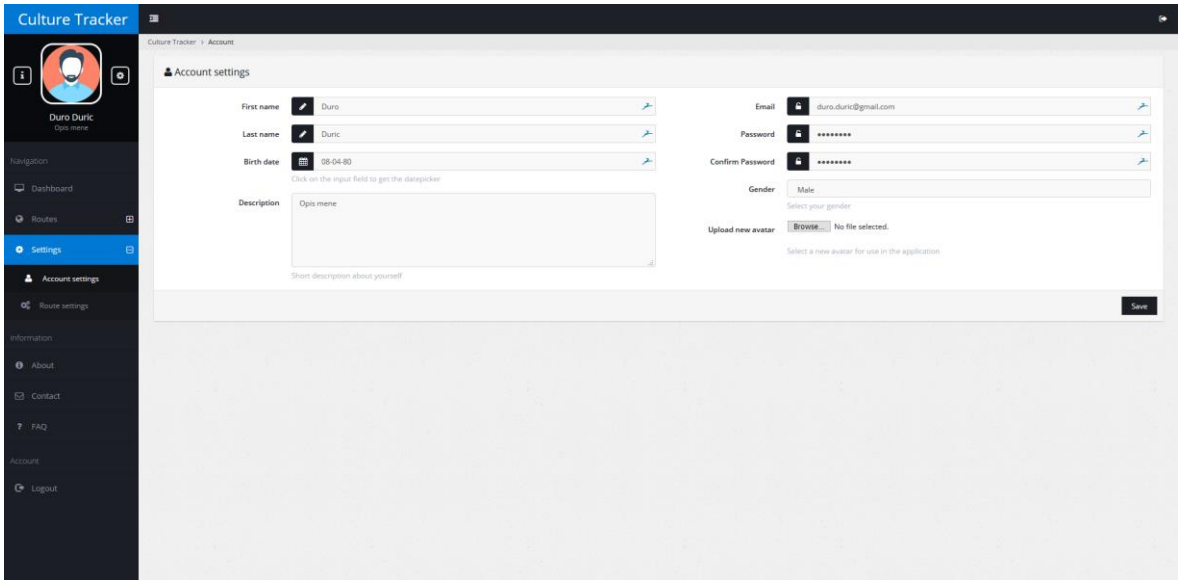
Slika 4.15 Detalji rute unutar Web aplikacije

Pogreška! Izvor reference nije pronađen. prikazuje detalje pojedine rute. Na Google Mapi se vide korisnikove kretnje unutar te rute te ako je propustio kulturni objekt. postavlja se marker sa kratkim opisom tog objekta. Na desnoj strani se nudi mogućnost promjene imena rute te dodavanje opisa. Također se može označiti rutu završenom, čime se smatra da smo naučili položaje svih kulturnih objekata na toj ruti.



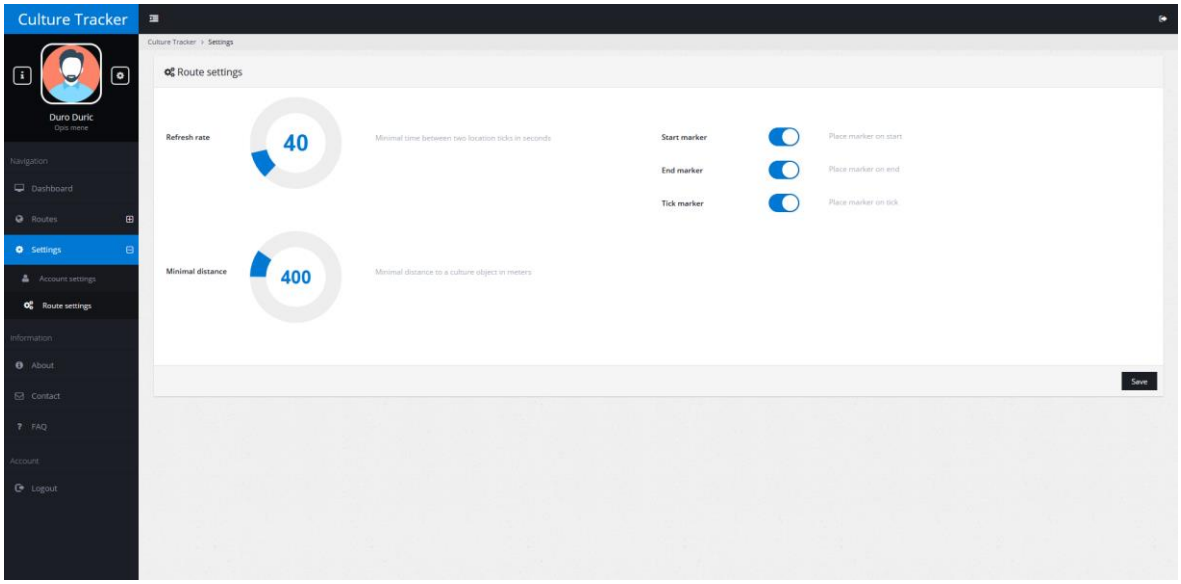
Slika 4.16 Detalji o kulturnim objektima

Pogreška! Izvor reference nije pronađen. prikazuje donji dio stranice sa detaljima rute na kojoj se nalazi popis svih kulturnih objekata propuštenih na toj ruti, njihov položaj, te mogućnost isključivanja njihovog radijusa. Također se nudi odlazak na pretražilicu „Google“ kako bi korisnik dobio još detaljnije podatke o objektu ako ga on zanima.



Slika 4.17 Korisnikove postavke

Pogreška! Izvor reference nije pronađen. prikazuje stranicu za promjenu korisničkih postavki. Nudi se mogućnost promjene ili unosa imena, prezimena, dana rođenja, opisa korisnika, email-a, lozinke te spola.



Slika 4.18 Postavke ruta

Pogreška! Izvor reference nije pronađen. prikazuje stranicu za promjenu postavki ruta. Nudi se mogućnost promjene minimalnog vremena potrebnog između praćenja lokacije, minimalna distanca za radijus unutar kojeg se traže kulturni objekti, te postavke za markere na startu, kraju, te svakoj lokaciji koja je zabilježena.

Zaključak

S ovim radom smo dokazali da uporabom tehnologije možemo približiti kulturu ljudima te ih poticati na kulturno uzdizanje. Kultura kao takva je bitan faktor svakog naroda te njihovog nacionalnog identiteta [1], a bilo koji način na koji se može podignuti razina kulturne osvještenosti je dobrodošla.

Ukazivanjem na propuštene kulturne znamenitosti te objekte potičemo ljude da istražuju navedene te ih time informiramo o postojanju istih u njihovoj sferi kretanja. Idealna situacija bi bila da nakon korištenja aplikacije korisnik zaista i posjeti te objekte te da informira, a samim time i educira svoju obitelj i prijatelje.

U poglavlju (2) smo postavili hipotezu rada te smo tamo kao glavni nedostatak kulturnih objekata istaknuli slabo reklamiranje i samim time slabo poznavanje njihovih lokacija te sadržaja. Kroz rad smo pokazali da je nedvosmisleno moguće napraviti aplikaciju koja bi pratila korisnikove kretnje te mu omogućavala pregled propuštenih kulturnih objekata na toj ruti, samim time smo dokazali hipotezu rada jel korisnici naše aplikacije će imati nadohvat ruke sve detalje o svim kulturnim objektima gdje god na svijetu pošli.

Kao i svaka aplikacija, i ova ima prostora za nadogradnju te napredak, ali smo dobro definiranom arhitekturom te smjerom omogućili da se potencijalne nadogradnje učine na bezbolan način te time olakšali stvaranje novih mogućnosti unutar aplikacije.

Iako trenutno ne postoji puno sličnih aplikacija, možemo očekivati da će potencijalni uspjeh naše aplikacije stvoriti novi segment na tržištu koji će biti koncetriran i usmjeren ka kulturi te kulturnim zbivanjima te iz toga razloga moramo biti budni te pomno pratiti zbivanja na tržištu kako bismo uvijek bili jedan korak ispred konkurencije.

Popis kratica

REST	*Representational state transfer*	arhitekturalni model na poslužitelju
API	*Application programming interface*	aplikacijsko programsko sučelje
MVC	*Model–View–Controller*	arhitekturalni model
MVP	*Model–View–Presenter*	arhitekturalni model
JSON	*JavaScript Object Notation*	format podataka
ORM	*Object-relational mapping*	sustav za komuniciranje s bazom
CRUD	*Create, Read, Update, Delete*	operacije nad podacima

Popis slika

Popis kôdova

Literatura

[1] Cicéron, Marcus Tullius Cicero; Bouhier, Jean (1812). Tusculanes. Nismes: J. Gaude. p. 273. OCLC 457735057.

[2] I. LJUBI, B. MIHALJEVIĆ; Interoperabilnost informacijskih sustava; Algebra d.o.o.; (2013)

[3] R. C. MARTIN; Clean Code: A Handbook of Agile Software Craftsmanship; Prentice Hall; (2008)

[4] S. J. METSKER; Design Patterns Java Workbook; Addison-Wesley Professional; (2002)

[5] R. PAUL; Dream(sheep++): A developer's introduction to Google Android; (2009)

[6] http://owin.org/; 18. prosinca 2017.

[7] https://material.io/guidelines/; 26. prosinca 2017.