Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Željko Plesac

Vpeljava družabne komponente pri GPS navigaciji

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2014

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Željko Plesac, z vpisno številko **63110380**, sem avtor magistrskega dela z naslovom:

Vpeljava družabne komponente pri GPS navigaciji

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom doc. dr. Dejana Lavbiča
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

V Ljubljani, 8. julija 2014 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Dejanu Lavbiču za vso strokovno pomoč pri izdelavi tega magistrskega dela. Za vso podporo pri študiju se zahvaljujem tudi svojim staršem ter punci Kristini.

Kazalo

Slike

Tabele

Seznam uporabljenih kratic

kratica	angleško	slovensko
GPS	global positioning system	globalni sistem pozicioniranja
API	application programming interface	aplikacijsko programljivi vmesnik
KPI	key performance indicator	ključni pokazatelji uspeha
M2M	machine to machine	komunikacija med napravami
MVC	model-view-controller	model-pogled-upravljalec
SQL	structured query language	strukturirani povpraševalni jezik
REST	representational state transfer	arhitekturni stil na svetovnem spletu
SOAP	simple object access protocol	protokol za izmenjavo strukturiranih informacij
HTTP	hypertext transfer protocol	protokol za izmenjavo hiperteksta
XML	extensible markup language	razširljivi označevalni jezik
JSON	javascript object notation	format zapisa podatkov v ključ-vrednost obliki
SDK	software development kit	paket za razvoj programske opreme
OSM	open street map	prosto dostopni zemljevidi
PBF	protocolbuffer binary format	binarni format zapisa, alternativa XML-u
ORM	object relational mapper	entitetno-relacijska preslikava
GCM	google cloud messaging	google potisna obvestila
KNN	k nearest neighbours	k najbližjih sosedov
MDL	minimum description length	princip najkrajšega opisa
SVM	singular vector machines	metoda podpornih vektorjev

CBA	classification based on associations	algoritem za klasifikacijo na podlagi asociativnih pravil
CBA-RG	classification based on associations - rule generator	algoritem za klasifikacijo na podlagi asociativnih pravil - iskanje razrednih pravil
CBA-CB	classification based on associations - classifier builder	algoritem za klasifikacijo na podlagi asociativnih pravil - generiranje klasifikatorja
SWOT	strengths, weaknesses, opportunities, and threats	prednosti, slabosti, priložnosti, nevarnosti

Povzetek

GPS tehnologija je danes osnoven del vsakega potovanja, z razvojem pametnih telefonov pa je postala dostopna večjemu številu uporabnikov. Skozi čas je razvoj tehnologije omogočil večjo dostopnost različnih informacij, ki vplivajo na potovanja in na večjo povezavo med uporabniki. Dejstvo je, da so obstoječe GPS aplikacije zastarele, zaradi česar smo si v okviru magistrske naloge zastavili cilj razviti GPS aplikacijo naslednje generacije, ki bo z močno uporabo socialne komponente ter spoznavanjem svojih uporabnikov izboljšala uporabniško izkušnjo mobilnih GPS aplikacij. Sistem je zasnovan na uporabi večih metod strojnega učenja, pri katerih se morajo zaradi posebne strukture sistema te metode prilagoditi za uporabo na mobilni platformi. Na podlagi predlaganega sistema smo izdelali tudi funkcionalen prototip, ki je dokazal, da se obstoječe metode lahko uporabljajo za izboljšavo uporabniške izkušnje mobilnih aplikacij, ki temeljijo na GPS tehnologiji.

Ključne besede:
Android, REST API, klasifikacija, navigacija, uporabniška izkušnja, socialna komponenta

Abstract

GPS technology has become an essential part of any travel and, with the development of smartphone technology, it has become accessible to a larger number of users. The development of technology over time has also enabled easier access to different information which can influence travel, and users of these applications have become more connected than ever before. All of this has resulted in one simple fact - existing GPS applications have become obsolete. For this reason, we have set the goal of developing the next generation of GPS applications, which will make significant use of a social component, as well as learning about its users' interests in order to improve the user experience. The system is designed to use different methods of machine learning, which have to be optimized for use on mobile platforms. We have also developed a functional prototype, which has demonstrated that the existing methods can be used to improve the user experience of GPS mobile applications.

Key words

Android, REST API, classification, navigation, user experience, social component

Poglavje 1

Uvod

Dandanes na tržišču obstaja veliko število naprav, ki omogočajo storitev GPS[1]. Te naprave delujejo na zelo enostaven način: uporabnika vodijo od točke A do točke B, in mu na predlagani poti posredujejo informacije, ki so povezane s samim potovanjem, kot so na primer bencinske črpalke, restavracije in avtoceste. Zaradi enostavne uporabe in nizke cene so takšne naprave postale osnovni del opreme, ki jo mora imeti vsak potnik v času potovanja. Zavedati pa se moramo, da v današnjih časih takšen način dela zagotovo ni ustrezen. Z razvojem tehnologije, kot so pametni telefoni in tablice, je lahko danes vsakdo lastnik naprave, ki jo je mogoče enostavno in hitro locirati kjerkoli na zemeljski obli. Takšne naprave imajo velik ekran na katerem je mogoče prikazati navigacijsko karto, poleg tega pa imajo tudi dovolj spomina za zagon različnih aplikacij, ki nam ponujajo informacije na našem potovanju (od različnih družabnih omrežij do aplikacij za ocenjevanje kakovosti ponudbe restavracij) in stabilno ter hitro povezavo z internetom, ki nam omogoča pridobivanje in objavo informacij. Proizvajalci GPS naprav so se prilagodili obstoječemu stanju in začeli proizvajati aplikacije, ki uporabljajo različne podatke iz interneta z namenom, da potovanje naredijo bolj hitro, varno in zabavno. Takšne aplikacije so v svetu zelo priljubljene in velja pričakovati, da bodo v prihodnosti samo še pridobivale na pomembnosti, bodisi zaradi tega, ker do sedaj ni bilo mogoče potovati v različne dele sveta na tako hiter, poceni in varen način, bodisi zato, ker bodo v prihodnosti imele največji delež na tržišču in bodo cenovno bolj

[1]GPS - ang. Global Positioning System

dostopne večjemu številu uporabnikov. Vse skupaj v veliki meri vpliva na motivacijo programerjev mobilnih aplikacij, ki na tem področju lahko vidijo možnosti za realizacijo zanimivih idej.

Z analizo obstoječih aplikacij, ki imajo največji tržni delež smo opazili, da imajo le-te v načinu dela eno večjo pomanjkljivost: poti vedno predlagajo na isti način, saj so objektivne in se ničesar ne naučijo o svojem uporabniku, prav tako pa svojega načina in metode dela posamezniku ne prilagajajo. Zanimiva je situacija v kateri aplikacija npr. prvemu uporabniku, ki raje uporablja javni prevoz in drugemu, ki raje uporablja svoj avtomobil, obema predlaga isto pot od točke A do točke B. Ali pa če uporabnik vsak petek potuje domov iz večjega mesta, in mu aplikacija vedno svetuje isto pot, brez da bi upoštevala gnečo na cesti in pretekle izkušnje na tej poti. Po drugi strani, z razvojem socialnih omrežij in eksponentno rastjo števila njihovih uporabnikov, so dandanes omrežja, kot so Facebook, Twitter ali Google+ postala največji prosto dostopen repozitorij različnih informacij. Uporabniki podajajo informacije o sebi, svojih interesih, slike z obiskanih lokacij, in še veliko več. Ti podatki so dostopni skozi uradni API[2] za vse vrste odjemalcev – kar velja tudi za mobilne aplikacije. Vse to nas pripelje do vprašanja, na kakšen način bi lahko uporabili informacije o uporabniku mobilne naprave za izboljšanje delovanja GPS aplikacij. Da bi našli odgovor, smo si v okviru magistrske naloge zastavili cilj izdelati funkcionalni prototip GPS aplikacije naslednje generacije, ki bi izboljšal uporabniško izkušnjo z uporabo socialne komponente. Slednjo definiramo kot prosto dostopne podatke o uporabniku, ki nam lahko podajo njegov dodaten opis. Z analizo in uporabo teh informacij, lahko izboljšamo obstoječe GPS sisteme na treh različnih področjih – (1) izboljšanje uporabniške izkušnje, (2) izboljšanje varnosti in (3) večja uporaba družabne komponente:

1. V okviru prve izboljšave bomo vsakemu uporabniku pošiljali informacije in mu generirali pot, pri tem pa bomo upoštevali tudi njegov profil. Za gradnjo profila bomo uporabili že razvite metode strojnega učenja, kot so asociativna pravila in razvrščanje v skupine. Pri učenju bomo uporabili več KPI-jev[3], s katerimi bomo spoznali uporabnika, in mu prilagodili način dela aplikacije. Nove uporabnike bomo poskušali spoznati skozi njihove profile na družabnih

[2]API - ang. Application Programming Interface
[3]KPI - ang. Key Performance Indicator

omrežjih(Facebook, Twitter in Google+). Z večjo uporabo aplikacije bomo še bolje spoznali uporabnika in mu tudi bolje prilagajali način dela.

2. Varnost lahko izboljšamo na več različnih načinov. Najbolj pomembno je, da pred samim začetkom potovanja poskušamo predvideti čim večje število težav, kot so vremenske okoliščine na cesti, kateri dan v tednu je, ali je gneča na cesti, in jih uporabiti pri generiranju poti. Cilj nam je, da se izognemo težavam še preden do njih sploh pride. Podatke, ki vplivajo na varnost, bomo pridobili iz različnih spletnih storitev (gre za podatke o vremenskih napovedih, stanju na cestah, poročanih težavah drugih uporabnikov, policijskih priporočilih...). Aplikacija bo tudi sledila potovanju in v primeru, da bi bila na poti objavljena kakšna težava, bo le ta samodejno spremenila pot.

3. Omogočili bomo medsebojno komunikacijo uporabnikov na način, pri katerem ne vplivamo na varnost slednjih. Zaradi tega bo komunikacija potekala preko potisnih obvestil. Vsak uporabnik bo v sistem lahko poslal obvestilo, ki bo kategorizirano v več kategorij(sporočena policijska kontrola, težava na cesti, dobre restavracije, hoteli, splošno in drugo). Vsi uporabniki, ki se bodo nahajali v bližini objavljene informacije, bodo na svojo aplikacijo prejeli obvestilo.

Za podporo delovanju sistema bomo razvili tudi spletno storitev, ki bo uporabnikom ponujala informacije.

Magistrska naloga je formalno sestavljena iz 8 poglavij. Po uvodnem poglavju smo podali pregled obstoječih tehnologij in raziskav, v tretjem in četrtem poglavju pa smo našteli in podali kratek opis vseh uporabljenih tehnologij in algoritmov. Zatem smo v petem poglavju podrobno razložili predlog rešitve, in v naslednjem poglavju opisali implementacijske podrobnosti. V sedmem smo predstavili rezultate testiranja, primerjavo s konkurenčnimi produkti in SWOT analizo. Zaključili smo s sklepnimi ugotovitvami in zaključkom, kjer smo tudi našteli težave pri razvoju, ter ponudili izboljšave in nadgradnje predlaganega sistema.

Poglavje 2

Pregled obstoječih pristopov

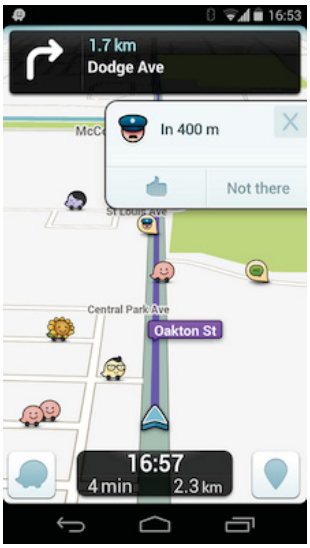
Področje GPS aplikacij in potovanj je za raziskovanje zelo zanimivo, zaradi česar je bilo v preteklih letih objavljenih veliko člankov in raziskav. Znanstveniki se precej ukvarjajo z izboljšavo varnosti pri potovanju. Tako avtorji v [1] in [2] predlagajo uporabo algoritmov strojnega učenja za predvidevanje gneče v prometu. S pridobivanjem podatkov iz spletnih storitev se zgradi model, ki usmerja algoritem generiranja poti na način, da poskuša poiskati pot z najmanjšo gnečo. Znanstveniki so razvili tudi prototip, ki pa ima nekaj velikih pomanjkljivosti in težav, kot so sinhronizacija podatkov iz interneta in njihov vpliv na aplikacijo, stalno aktivna internetna povezava, hitrost izvajanja algoritma in drugo. Avtorji opozarjajo tudi na problem optimizacije algoritmov strojnega učenja za uporabo na mobilnih napravah, ker smo pri teh zelo omejeni s procesorsko močjo in količino spomina. V [3] se ukvarjajo z izboljšanjem varnosti tako, da poskušajo omejiti interakcijo voznika z mobilnimi napravami. Predlagajo več možnih načinov, kot sta časovna zakasnitev izvajanja aplikacij, in zelo zanimiv pristop - uporabe konteksta aplikacije za avtomatsko svetovanje klicateljem. Bistvo je v tem, da aplikacija posreduje podatke o lokaciji, končnih točkah poti in času trajanja poti klicateljem z namenom, da vozniku naprave sploh ni potrebno uporabljati. Tudi pri tem načinu se pojavijo problemi, in sicer prioritizacija aplikacij in onemogočanje izvajanj nekaterih aplikacij. Avtorji uporabljajo koncept časovnega zamika in trdijo, da obstajata dve vrsti aplikacij: tiste, ki jih uporabnik potrebuje pred vožnjo, in tiste ki jih uporabnik uporablja, ko pride na cilj potovanja. V času potovanja uporabnik lahko uporablja le prototip aplikacije, kar pa večini uporabnikov predstavlja nesprejemljiv pogoj. V

[4] in [5] avtorji predlagajo način komunikacije med družabnimi omrežji in mobilnimi napravami z uporabo M2M[1] tehnologije. M2M je koncept komunikacije med dvema napravama z uporabo brezžičnih tehnologij. Avtorji v [4] so razvili prototip aplikacije, ki optimizira uporabo M2M koncepta za uporabo na mobilnih napravah, in sicer za prenos in uporabo velikih količin podatkov. Prototip so uporabili za prejemanje različnih informacij na poti, ki močno vplivajo na uporabniško izkušnjo. Obstaja tudi veliko raziskav, ki se ukvarjajo s tematiko družabnega faktorja v GPS aplikacijah. V [6] avtorji predlagajo pristop uporabe družabne komponente za svetovanje drugim uporabnikom aplikacije. Uporabnik se lahko prijavi v več različnih skupin in na njih oddaja ali prejema glasovna sporočila. Prototip se je med testiranjem izkazal za uspešnega - uporabniki so v več kot 70% primerov poslušali sporočila in prilagodili svojo pot, kar je posledično vplivalo na hitrejši prihod na končni cilj. Prototip ima velike težave z verodostojnostjo oddanih sporočil, ker ne predlagajo načina ocenjevanja kvalitete le-teh, in z izgledom uporabniškega vmesnika. V [7], [8] in [9] avtorji predlagajo zanimive pristope k reševanju in izogibanju gneč s kombiniranjem družabne komponente in naprednih algoritmov za analizo velikega števila podatkov iz spletnih storitev, ki opisujejo trenutno stanje v prometu.

Obstajajo tudi številne komercialne aplikacije, ki izboljšujejo način delovanja GPS naprav. Ena od njih je Waze [10], ki temelji na ideji crowdsourcinga[2]. Aplikacija je prosto dostopna in je tudi brezplačna. V veliki meri uporablja družabno komponento, kar pomeni, da so v delovanje aplikacije vključeni vsi uporabniki. Vsak uporabnik lahko na svoji poti vpiše informacije, pri čemer so le-te razdeljene v več kategorij, te informacije pa so vidne tudi ostalim uporabnikom aplikacije. Slednja ima eno veliko pomanjkljivost - GPS performance so žrtvovane za družabno komponento aplikacije. Ker zemljevide osvežujejo uporabniki sami, so pogosto nenatančni. Aplikacija je uporabna zgolj v večjih mestih, kjer obstaja veliko število uporabnikov. V manjših mestih ali področjih kjer ni veliko uporabnikov, aplikacija deluje slabše, saj so mape pogosto zastarele, na zemljevidu pa ni nobenih informacij. Vseeno je Waze (Slika 2.1) ena izmed najbolj priljubljenih aplikacij za

[1] M2M - ang. Machine To Machine

[2] Crowdsourcing - postopek pridobivanja potrebnih informacij, idej in servisov od uporabnikov, z namenom reševanja skupnega širšega problema

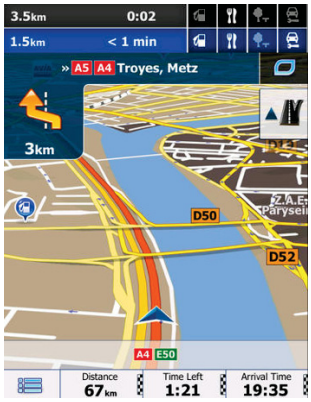


Slika 2.1: Primer navigacije - Waze mobilna aplikacija

potovanje, ki jo je podjetje Google leta 2013 kupilo za več kot milijardo ameriških dolarjev.

iGo Primo [11] ponuja številne možnosti za potovanja. Aplikacija (Slika 2.2a) svetuje poti na več različnih načinov in na poti daje številne informacije(vremenska napoved, gneče, omejitve hitrosti, pomoč pri obvladovanju križišč in drugo). Zemljevidi se redno osvežujejo s strani razvijalca, so zelo natančni, ampak so plačljivi. Aplikacija ponuja tudi napredne možnosti, kot sta pomnjenje lokacije avta in definiranje različnih scenarijev gradnje poti. Pomanjkljivost aplikacije se pojavi pri njeni ceni(obstaja več različnih modulov, pri čemer so vsi plačljivi), in ne obstaja nobena družabna komponenta(komunikacija z ostalimi uporabniki). Za razliko od konkurenčnih produktov, iGO Primo deluje zelo dobro tudi takrat, ko ni aktivne povezave za internetom.

Google Maps Navigation [12] je mobilna aplikacija, ki dodatno razširja delovanje Google Maps aplikacije in je trenutno na voljo v več kot 100 državah. Aplikacija (Slika 2.2b) omogoča navigacijo do ciljne točke na več različnih načinov – najhitrejša pot, najbolj optimalna pot, odvisno od prevoznega sredstva... Aplikacija tudi komunicira z Googlovim iskalnikom, in omogoča iskanje cilja z uporabo fraz (na primer »Poišči najbližjo restavracijo«) ali pa iskanje po imenu lokacije. Zemlje-



(a) Primer navigacije - iGO Primo mobilna aplikacija

(b) Primer navigacije - Google Maps Navigation

Slika 2.2: Primer komercialnih GPS aplikacij

vidi so brezplačni in se redno osvežujejo. Pomanjkljivost aplikacije v primerjavi s konkurenčnimi produkti je v tem, da lahko različne države vplivajo na natančnost Google Maps zemljevidov. Na primer, nekatere lokacije in stavbe so popolnoma zamegljene ali obarvane v črno barvo zaradi varnostnih razlogov.

Poglavje 3

Uporabljene tehnologije

Predlagano rešitev, ki jo bomo podrobneje predstavili v naslednjih poglavjih, sestavljajo štiri osnovne komponente:

1. **strežnik**,
2. **odjemalec**,
3. **vmesnik**,
4. **Zunanje storitve**.

Strežnik in vmesnik smo implementirali v .NET razvojnem okolju podjetja Microsoft zaradi enostavnosti implementacije. .NET platforma v aktualni različici 4.5 ponuja tudi vse komponente, ki jih potrebujemo za implementacijo:

1. **ASP .NET MVC 4** - implementacija strežnika;
2. **ASP .NET Web API 2** - implementacija vmesnika;
3. **Microsoft SQL Server 2012** - relacijska podatkovna baza.

Pomanjkljivost .NET razvojnega okolja je v tem, da le-to ni prosto dostopno in je za razvoj potrebno pridobiti plačljivo licenco.

Strežnik se izvaja na Somee spletni storitvi [13], ki nam omogoča omejen brezplačen paket. Paket se je v končni fazi izkazal kot primeren za uporabo in ni bilo potrebe za najem dodatnega strežnika ali plačevanje premium paketa.

Odjemalec je implementiran kot Android mobilna aplikacija iz več razlogov:

1. Enostavnost implementacije - v programskem jeziku Java
2. Ogromen tržni delež (več kot 80% trga mobilnih naprav)
3. Več izkušenj z razvojem Android mobilnih aplikacij

Mobilna aplikacija je implementirana v Eclipse razvojnem okolju in testirana na LG Nexus 4 pametnem telefonu z Android 4.4.2. verzijo operacijskega sistema. Grafični vmesnik mobilne aplikacije in spletnega portala je implementiran z uporabo Bootstrap metode, ki nam omogoča uporabo enakih grafičnih komponent na večih različnih platformah, kar nam zagotavlja tudi ugodnejšo uporabniško izkušnjo.

3.1 Strežnik

3.1.1 ASP .NET MVC

ASP NET je programsko okolje, ki nam omogoča razvoj strežniških aplikacij [14]. Okolje je razvilo podjetje Microsoft in je bilo prvič predstavljeno leta 2001. Leto dni pozneje se je na trgu našla uradna različica .NET 1.0. Trenutna verzija je .NET 4.5. Programsko okolje nam omogoča razvoj v več programskih jezikov, kot so Visual Basic, Visual C#, Visual F# ... Najbolj priljubljena sta Visual Basic in Visual C#. Razvoj poteka v Visual Studio razvojnem okolju (trenutno je uradna različica iz leta 2013).

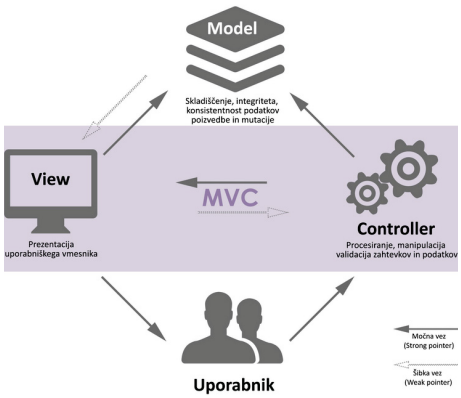
ASP NET MVC[1] je dodatek k ASP NET, ki nam omogoča razvoj strežniških aplikacij, uporablja pa model-view-controller vzorec.

3.1.1.1 Model-view-controller vzorec

Model-view-controller vzorec (Slika 3.1) se uporablja pri razvoju programske opreme za ločevanje arhitekture v tri medsebojno povezane komponente:

- **View (slo. Pogled)** – uporablja se za prezentacijo informacij o modelu (npr. z uporabo diagrama, tablice ali HTML kode.) Vse potrebne informacije za prezentacijo se pridobijo od upravljalca.

[1]MVC - ang. Model-View-Controller



Slika 3.1: MVC vzorec

- **Model (slo. Model)** - centralna komponenta, ki vsebuje podatke, poslovna pravila in poslovno logiko. Obvešča pogled in upravljalca o spremembi svojega stanja(podatki). Obstaja tudi t.i. "pasivna" implementacija, kjer upravljalec in pogled sprašujejo model, če je spremenil svoje podatke.

- **Controller (slo. Upravljalec**) - modelu pošilja zahtevke za spremembo njegovega stanja, pogledu pa pošilja zahtevke za spremembo prezentacije stanja modela. Procesira, manipulira in validira uporabniške zahtevke.

3.1.2 Microsoft SQL Server

Microsoft SQL[2] Server [15] je plačljiva programska oprema podjetja Microsoft, ki nam omogoča uporabo relacijskih podatkovnih baz. Brezplačna verzija je privzeto vgrajena v Visual Studio razvojno okolje, ampak je zelo omejena z maksimalnim številom relacijskih tablic in podatkov, ki jih lahko hranimo v bazi.

3.1.3 ASP .NET Web API 2

ASP .NET Web API 2 je ogrodje za implementacijo metod, s pomočjo katerih lahko strežnik izpostavlja svoje storitve in podatke različnim odjemalcem. Ogrodje

[2]SQL - ang. Structured Query Language

omogoča uporabo različnih metod za komunikacijo, kot so SOAP[3] ali HTTP[4], podatki pa so lahko v obliki XML-a[5], JSON-a[6], navadnega teksta... Privzeto je vgrajeno v .NET razvojno okolje, in je primarna tehnologija za implementacijo spletnega API-ja.

3.2 Vmesnik

3.2.1 REST protokol

REST[7] protokol je oblika programske arhitekture, ki se uporablja v razvoju programske opreme za komunikacijo in prenos podatkov. Temelji na šestih preprostih principih:

1. **Povezavo odjemalec-strežnik** - sestavljata jo dve ločeni komponenti - odjemalec in strežnik. Komponente imajo jasno definirane vloge:

   - Podatki se hranijo na strežniku – povečana neodvisnost in prenosljivost odjemalca;
   - Odjemalec je odgovoren za obliko uporabniškega vmesnika – preprostejša in bolj skalabilna uporaba strežnika.

   Odjemalec in strežnik medsebojno komunicirata skozi vmesnik, ki nam omogoča ločen razvoj vsake komponete.

2. **Stanje povezave** - strežnik ne shranjuje stanja povezave z odjemalcem. Vsak nov zahtevek odjemalca vsebuje vse potrebne informacije za komunikacijo s strežnikom.

3. **Pomnjenje zahtev** - odjemalec si lahko zapomni odgovor strežnika na zahtevek. Na ta način lahko zmanjšamo količino prometa med odjemalcem in strežnikom.

[3]SOAP - ang. Simple Object Access Protocol
[4]HTTP - ang. Hypertext Transfer Protocol
[5]XML - ang. Extensible Markup Language
[6]JSON - ang. JavaScript Object Notation
[7]REST - ang. REpresentational State Transfer

4. **Večnivojski sistem** – komunikacija med odjemalcem in strežnikom se lahko odvija tudi prek posrednikov (odjemalec se ne zaveda, če je povezan direktno na strežnik). Z uporabo posrednikov lahko omogočimo boljšo skalabilnost sistema, in zagotovimo hitrejšo odzivnost strežnika.

5. **Koda na zahtevo** - strežnik lahko direktno vpliva na funkcionalnost odjemalca na način, da mu jo lahko začasno razširi ali prilagodi. Slednje lahko dosežemo na način, da strežnik kot odgovor na zahtevo pošlje odjemalcu izvršljivo kodo (npr. JavaScript kodo). Implementacija tega principa ni obvezna.

6. **Enoten vmesnik** – odjemalec in strežnik komunicirata skozi enoten vmesnik, ki nam olajša neodvisen razvoj delov sistema. Je temeljni princip REST protokola.

Posebnost REST protokola je tudi v tem, da omogoča razvoj odjemalca in strežnika v različnih programskih jezikih, in je zaradi tega v svetu mobilnih aplikacij zelo priljubljen. Odjemalec in strežnik najbolj pogosto komunicirata prek HTTP protokola in izmenjujeta podatke v obliki XML-a, JSON-a, navadnega teksta ali HTML-a. Standardne akcije pri HTTP komunikaciji so:

1. **GET** - pridobivanje podatkov s strežnika;

2. **POST** - dodajanje ali shranjevanje podatkov na strežnik;

3. **PUT** - spreminjanje podatkov na strežniku;

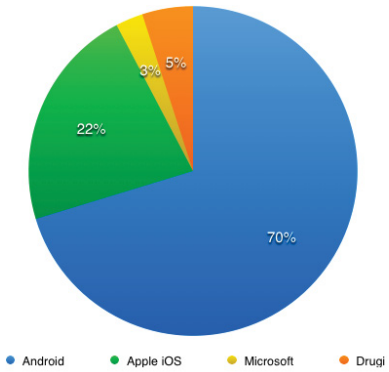
4. **DELETE** - brisanje podatkov na strežniku.

Podprte so tudi vse ostale funkcije HTTP-protokola. Protokol je bil razvit leta 2000 v okviru doktorske disertacije Roya Fieldinga na Univerzi v Kaliforniji, ZDA.

3.3 Odjemalec

3.3.1 Android operacijski sistem

Android je odprtokodna programska platforma in operacijski sistem, ki je zasnovan na Linuxovem jedru. Operacijski sistem je predvsem namenjen za mobilne naprave,



Slika 3.2: Tržni delež mobilnih operacijskih sistemov (marec 2014.)

ki uporabljajo zaslon na dotik. Na začetku so bili to mobilni telefoni, danes pa se je Android razširil tudi na tablice in televizije in je tako postal najbolj priljubljen mobilni operacijski sistem z več kot 70% deleža na globalnemu trgu (Slika 3.2 [17]).

Razvoj na platformi se je začel v podjetju Android Ltd, ki so ga ustanovili Andy Rubin, Rich Miner, Nick Sears in Chris White. Podjetje je leta 2005 prevzel Google, ki je platformo uradno predstavil 5.11.2007, ko je bila ustanovljena tudi Open Handset Alliance - zveza tehnoloških podjetji, ki si je za cilj zastavila razvoj odprtih standardov za mobilne naprave. Leta 2008 se je na trgu pojavil prvi pametni mobilni telefon, ki je uporabljal Android 1.0. - HTC Dream. Trenutna verzija Androida je 4.4.4, kodnega imena KitKat. Še vedno je najbolj priljubljena verzija JellyBean(4.0 - 4.2.2), z več kot 50% deleža.

Razvoj aplikacij za Android operacijski sistem poteka v programskem jeziku Java, ki ga nadgradimo z Android SDK[8] [16]. Platforma je zasnovana na Linuxovem jedru, in uporablja Dalvik navidezni stroj za just-in-time kompilacijo kode v Java-byte kodo. Omogočena sta tudi razvoj in optimizacija uporabe strojne opreme v programskih jezikih C/C++. Z uporabo različnih ovojnic je razvoj omogočen tudi v drugih programskih jezikih, kot je C#(MonoDroid) ali pa Python.

Posebnost Android operacijskega sistema, ki ga razlikuje od konkurenčnih produktov (iOS, Windows Phone 8) je njegova Apache Licence 2.0 [18] licenca, ki programerjem omogoča prost dostop do izvorne kode sistema, njegovo uporabo in razvoj.

[8]SDK - ang. Software Development Kit

3.4 Zunanje storitve

3.4.1 Google Geocoding API

Google Geocoding API [19] nam omogoča postopek geolociranja in vzvratnega geolociranja uporabniške lokacije. Geolociranje je postopek pridobivanja naslova s pomočjo geografskih koordinat, vzvratno geolociranje pa je postopek pridobivanja geografskih koordinat s pomočjo naslova.

Storitev je prosto dostopna kot spletna storitev. Omejitev je 2500 zahtevkov na dan, odgovor pa je lahko v obliki JSON ali XML-a (Slika 3.3).

3.4.2 Facebook API

Facebook [20] je brezplačno spletno družabno omrežje, razvito na začetku leta 2004 kot študentski projekt na Univerzi v Harvardu, ZDA. Začetniki so Mark Zuckenberg, Eduardo Saverin, Andrew McCollum, Dustin Moskovitz in Chris Hughes. Danes je Facebook najbolj priljubljeno družabno omrežje z več kot milijardo uporabnikov. Facebook svojim uporabnikom omogoča ustvarjanje profilov, povezavo z drugimi uporabniki, objavo in izmenjavo različnih informacij, slik, datotek in še veliko več. Danes je skupaj z Googlom največji prosto dostopni spletni repozitorij informacij.

Dostop do informacij je možen z uporabo uradnega API-ja, pri čemer pa morata dostop in manipulacija s podatki potekati v skladu z uradnimi pravili. Registracija na Facebook API je brezplačna in poteka v dveh korakih - registracija razvijalcev in registracija aplikacij na Facebook Developers konzolo. Po registraciji lahko dostopamo do API-ja z REST protokolom- z REST akcijami naredimo zahtevek, na katerega API vrne odgovor v obliki JSON-a ali XML-a (Slika 3.4).

Dostop do informacij je razdeljen v več kategorij, pri čemer morajo razvijalci za vsako kategorijo posebej pridobiti dovoljenje lastnikov informacij(uporabnikov). Vsak uporabnik ima na voljo tudi možnost zaščite zasebnih podatkov, kar pomeni da ti podatki niso več na voljo razvijalcem. Dostop je časovno omejen na 150 zahtevkov na uro, z uporabo oAUTH avtentifikacijskega protokola pa na 450 zahtevkov na uro.

```
{
   "results" : [
      {
         "address_components": [
            ...
                {
                    "long_name": "University of Ljubljana",
                    "short_name": "University of Ljubljana",
                    "types": [
                        "establishment"
                    ]
                },
            ...
         ],
          "formatted_address": "Trzaska cesta 25,
               University of Ljubljana, 1000 Ljubljana, Slovenia",
           "geometry": {
                "location": {
                    "lat": 46.0448994,
                    "lng": 14.4892307
                },
                "location_type": "ROOFTOP",
                "viewport": {
                    "northeast": {
                        "lat": 46.04624838029149,
                        "lng": 14.4905796802915
                    },
                    "southwest": {
                        "lat": 46.0435504197085,
                        "lng": 14.4878817197085
                    }
                }
            },
            "types": [
                "street_address"
            ]
      }
   ],
   "status" : "OK"
}
```


Slika 3.3: Primer odgovora na zahtevek na Google Geocoding API

```
{
  "id": "100006487747452",
  "name": "Zeljko Plesac",
  "education": [
    {
      "school": {
        "id": "117506898306940",
        "name": "Prirodoslovno-matematicka gimnazija Karlovac"
      },
      "type": "High School"
    },
    {
      "concentration": [
        {
          "id": "232375066953839",
          "name": "Magistrski studij Racunalnistva in informatike"
        }
      ],
      "school": {
        "id": "103673816367868",
        "name": "Fakulteta za racunalnistvo in informatiko"
      },
      "type": "College"
    }
  ]
}
```


Slika 3.4: Primer odgovora na zahtevek na Facebook API

3.4.3 Foursquare API

Foursquare [21] je brezplačno spletno družabno omrežje namenjeno predvsem za uporabo na mobilnih napravah. Uporabniki se v aplikacijo lahko prijavijo (ang. "check-in") na različnih lokacijah, in lahko tudi podajo komentar in oceno trenutne lokacije. S prijavo na določeni lokaciji, uporabniki zbirajo točke, ki jim lahko prinesejo nagrade na trenutni lokaciji (na primer s prijavo v kinodvorani uporabnik

lahko pridobi brezplačno vstopnico). Foursquare so leta 2009 ustanovili Dennis Crowley in Naveen Selvadurai kot nadgradnjo podobnega projekta Dodgeball (prevzet od Googlea leta 2005 in ukinjen 2009), ki ima danes približno 33 milijonov uporabnikov.

Podobno kot pri Facebook API-ju, je dostop do informacij o uporabniku in lokacijah možen z uporabo uradnega Foursquare API-ja. Po registraciji lahko izdelamo zahtevek za informacije z uporabo REST protokola, kjer dobimo odgovor v obliki JSON-a ali XML-a (Slika 3.5). Dostop do podatkov je časovno omejen na 150 zahtevkov na uro, z uporabo AUTH avtentifikacijskega protokola pa na 350 zahtevkov na uro.

```
{
    "meta": {
        "code": 200,
          ...errorType and errorDetail...
    },
    "notifications": {
          ...notifications...
    },
    "response": {
        ...results...
    }
}
```

Slika 3.5: Primer odgovora na zahtevek na Foursquare API

3.4.4 JSON.NET

JSON.NET je brezplačna knjižnica, ki nam omogoča deserializacijo in serializacijo JSON objektov v C# objekte z uporabo anotacij (Slika 3.6). Dostopna je na uradni spletni strani projekta [23] ali skozi NuGet packet manager, ki je privzeto dostopen znotraj Visual Studia.

```
public class Like
{
        [JsonProperty("category")]
        public String Category {get; set;}

        [JsonProperty("name")]
        public String Name {get; set;}

        [JsonProperty("created_time")]
        public String TimeCreated {get; set;}

        [JsonProperty("id")]
        public String Id {get; set;}

        [JsonProperty("category_list")]
        public  List<CategoryList> ListCategories {get; set;}
}
```

Slika 3.6: Primer C# objekta z JSON.NET anotacijami

3.4.5 Open Weathermap Map API

Open Weather Map API [22] je prosto dostopen programski vmesnik, ki posreduje podatke o vremenu s pomočjo spletne storitve. Podatki so zelo bogati in raznovrstni: število padavin v preteklih 3 urah, smer in hitrost vetra, temperatura, vlažnost ...

Zahtevek izdelamo z uporabo REST protokola, kjer odgovor sprejememo v obliki JSON-a ali XML-a (Slika 3.7).

3.4.6 Accord .NET

Accord .NET [24] je brezplačna knjižnica za .NET razvojno okolje, ki omogoča uporabo algoritmov s področja strojnega učenja, računalniškega vida, statistike in numerične matematike. Vsi algoritmi so posebej optimizirani za uporabo na strežniških aplikacijah. Knjižnica je na voljo tudi za Javo in Android kot Catalano

```
{
    "coord": {
        "lon": -0.13,
        "lat": 51.51
    },
    "sys": {
       ...
    },
    "weather": [
        {
           ...
        }
    ],
    "base": "cmc stations",
    "main": {
        "temp": 289.45,
        "humidity": 72,
        "pressure": 1013,
        "temp_min": 287.59,
        "temp_max": 291.48
    },
    "wind": {
        "speed": 2.06,
        "gust": 7.2,
        "deg": 43
    },
    "rain": {
        "3h": 0.5
    },
    "clouds": {
        "all": 80
    },
    ...
}
```

Slika 3.7: Primer odgovora na zahtevek na Open Weather Map API

Framework.

3.4.7 Prometno-informacijski center API

Prometno-informacijski center [25] omogoča dostop do podatkov o trenutnih razmerah na cesti skozi spletni servis. Podatki, ki jih lahko pridobimo, so lokacija izrednega dogodka,tekstualen opis dogodka, tip dogodka in podatek, če je neka cesta zaprta. Odgovor je lahko v obliki XML-a ali JSON-a (Slika 3.8).

3.4.8 OpenStreetMap

OpenStreetMap(v nadaljevanju OSM[9]) so brezplačni zemljevidi celotnega sveta, ki so na voljo v obliki PBF[10] in XML na uradni spletni strani projekta [26]. Zaradi vse večjih omejitev na prosto uporabo geografskih informacij, podprt z uspehom Wikipedije in razvojem cenejših navigacijskih sistemov, je Steve Coast leta 2004 začel z razvojem OSM zemljevidov. Projekt je zasnovan na "comunity based" principu, ki vsakemu uporabniku omogoča sodelovanje pri nadaljnjem razvoju zemljevidov. Danes ima OSM več kot milijon registriranih uporabnikov. Razvoj in nadzor projekta izvaja neprofitna organizacija OpenStreetMap Foundation, s sedežem v Suttonu, Anglija. Za lažje urejanje zemljevidov OSM je na voljo kar nekaj namenskih programov, izmed katerih sta najbolj priljubljena JOSM [28] ter Merkaartor [29]. Posodabljanje zemljevidov in programske opreme se odvija tedensko.

OSM uporablja topološko strukturo podatkov, in definira štiri temeljne tipe le-teh:

1. **Node** - Vozlišče, ki definira eno točko v prostoru na podlagi geografskih koordinat. Vozlišče predstavlja samostojno lastnost na zemljevidu, kot so na primer restavracije, trgovine, trgi ...

2. **Way** - Pot je urejen seznam vozlišč, ki lahko vsebuje med 2 in 2000 različnih vozlišč. Pot lahko predstavlja linearne značilnosti zemljevida, kot so reke ali ceste. Poleg tega lahko predstavlja področja znotraj poligona, ki ga opisuje množica točk, kot so na primer zgradbe ali gozdovi. V tem primeru morata biti prvo in zadnje vozlišče v seznamu enaka.

[9]OSM - ang. OpenStreetMap
[10]PBF - ang. Protocolbuffer Binary Format

```
"dogodek": [
            {
    "dovoljenjeDatKon": null,
    "y_wgs": 46.12872577775937,
    "kategorija": "R1",
    "zbrisano": null,
    "isMejniPrehod": false,
    "opis": "R1-210, Skofja Loka",
    "vir": "Direkcija RS za ceste",
    "operater_izbris": null,
    "id": 161120,
    "veljavnostDo": 1401918600000,
    "prioriteta": 1,
    "operater_sprememba": null,
    "prioritetaCeste": 15,
    "veljavnostOd": 1401810540000,
    "cesta": "R1-210, Skofja Loka - Gorenja vas",
    "vneseno": 1401810744140,
    "updated": 1401810744140,
    "x_wgs": 14.220155963675674,
    "dovoljenjeSt": null,
    "opisEn": "R1-210, Skofja Loka - Gorenja vas",
    "stacionaza": 9387,
    "operater_vnos": "Mesaric Nina",
    "odsek": "1110",
    "vzrokEn": "Other events",
    "icon": "http://kazipot1.promet.si/kazipot/services...",
    "spremenjeno": null,
    "vzrok": "Izredni dogodek",
    "dovoljenjeDatZac": null,
    "y": 109692.689263649,
    "x": 440117.155890579
},
```


Slika 3.8: Primer odgovora na zahtevek na Prometno-informacijski center API

3. **Relation** - Relacija je urejen seznam vozlišč, poti in relacij, ki opisuje povezavo med dvema ali več objekti, pri čemer ima vsak objekt definirano vlogo. Relacije se uporabljajo za opis povezave med vozlišči in potmi. Primer take povezave je kolesarska pot, ki je sestavljena iz večih osnovnih objektov tipa pot (way). Prav tako se relacija uporablja za opis omejitev na cestah in ostalih poti.

4. **Tag** - Vsak osnovni podatkovni tip lahko vsebuje oznake, ki ga dodatno opisujejo. Oznaka je tip podatka ki ne obstaja samostojno, ampak je vedno dodana na enega od ostalih osnovnih tipov. Struktura oznake je sestavljena iz dveh tekstovnih polj – ključa (vozlišče, relacija ali pot) in vrednosti. Slednja lahko vsebujeta poljubne vrednosti, vendar obstajajo smernice za vnos teh podatkov.

3.4.9 SQLite

SQLite [27] je odprtokodna relacijska podatkovna baza, ki je vgrajena v Android operacijski sistem. Od ostalih relacijskih podatkovnih baz se razlikuje v tem, da podatke vpisuje v navadne datoteke, ki so shranjene na trdem disku naprave. Na ta način lahko shranimo kompleksne baze podatkov, pri katerih uporabljamo zelo majhno količino spomina. Zaradi tega je idealna za uporabo na napravah, ki imajo na voljo majhno količino spomina (kot so mobilne naprave). Za poizvedbo podatkov se uporablja jezik SQL.

3.4.10 ActiveAndroid

ActiveAndroid je odprtokodna knjižnica za Android, ki nam omogoča uporabo ORM[11] načela v Android operacijskem sistemu. To omogoča na način, da definira razred, ki uporablja SQLite bazo podatkov kot privatni atribut in z uporabo javnih metod omogoča dostop do baze podatkov ter manipulacijo s podatki.

Knjižnico je razvil Michael Pardo, in je na voljo pod Apache Licence 2 licenco na spletni strani projekta [30].

[11] ORM - ang. Object Relational Mapper

```
public class APIJsonResponseModel {

        @SerializedName("status")
        private String status;

        public APIJsonResponseModel(){

        }

        public String getStatus() {
                return status;
        }

        public void setStatus(String status) {
                this.status = status;
        }
}
```

Slika 3.9: Primer Java objekta z GSON anotacijami

3.4.11 GSON

GSON je odprtokodna knjižnica, ki nam omogoča deserializacijo in serializacijo JSON objektov v Java objekte z uporabo anotacij (Slika 3.9). Dostopna je na uradni spletni strani projeka [31] in preko Maven centralnega repozitorija.

3.4.12 Google Cloud Messaging

Google Cloud Messaging [32] (v nadaljevanju GCM[12]) je brezplačna storitev, ki nam omogoča pošiljanje in sprejemanje potisnih obvestil na Android pametne telefone (Slika 3.10). Potisna obvestila so definirana kot majhen del informacije, ki ga lahko s strežnika pošljemo na več mobilnih telefonov. Pogoj je, da vse mobilne naprave uporabljajo enako aplikacijo in imajo naloženo zadnjo različico Google Play Services aplikacije.

[12]ang. Google Cloud Messaging

Slika 3.10: Notifikacije v notifikacijskem meniju

Pošiljanje notifikacij je omogočeno skozi POST zahtevek na spletni naslov [33]. V glavi zahtevka so obvezne naslednje informacije:

- **Content-type** – določimo, če je zahtevek definiran v JSON obliki (application/json), ali v obliki navadnega teksta (application/text).
- **Authorizaton-key** - enkratno določeni ključ naše aplikacije, ki nam ga za vsako aplikacijo posebej določi Google. Ključ lahko pridobimo v Google razvijalski konzoli [34].

V telesu zahtevka definiramo:

- **Registration_ids** – seznam enkratno določenih ključev, kjer vsak ključ predstavlja eno mobilno napravo. Ključi se avtomatsko določijo na vsaki napravi z implementacijo GoogleCloudMessaging vmesnika.
- **Data** – definiramo vsebino sporočila, ki ga pošljemo na vse mobilne naprave določene v registration_ids.

GCM ne postavlja omejitve na število poslanih obvestil, ampak je omejena velikost vsebine sporočila na največ 4KB podatkov.

Slika 3.11: Android Bootstrap vizualni gradniki

3.4.13 Android Bootstrap

Android Bootstrap [35] je odprto-kodna knjižnica za Android, ki nam omogoča uporabo Boostrap grafičnih komponent (Slika 3.11), posebej oblikovanih za uporabo na mobilni platformi. Trenutno je knjižnica podprta le za mobilne naprave, ki uporabljajo najmanj verzijo 4.0 Android operacijskega sistema.

3.4.14 Android Asynchronous Http Client

Android Asynchronous Http Client [37] je brezplačna odprtokodna knjižnica, ki nam omogoča izdelavo asinhronih zahtevkov na strežnik z uporabo REST metod. Zasnovana je na standardnem Apache HTTP odjemalcu. Posebnost knjižnice je v tem, da se vsi zahtevki opravijo v ozadju (ne obremenjujemo glavno UI nit) in z uporabo povratnih klicev lahko nadaljujemo delo na glavni niti po tem, ko sprejememo odgovor na zahtevek. Knjižnica uporablja več različnih tipov podatkov – JSON, XML, navaden text, binarne datoteke. Omogoča tudi uporabo kompresiranih podatkov (GZIP format), zaradi česar je zelo primerna za uporabo v primeru večjega števila podatkov. Knjižnico je razvil James Smith in je na voljo pod Apache

Licence 2 licenco na spletni strani projekta.

3.4.15 Android Universal Image Loader

Android Universal Image Loader [38] je brezplačna odprtokodna knjižnica, ki nam omogoča asinhrono nalaganje multimedijskih datotek s spletnih ali lokalnih virov (Slika 3.12). S pomočjo knjižnice skrbimo tudi za lokalno pomnjenje in manipulacijo z datotekami. V različici 1.9.0. je omogočeno tudi sinhrono nalaganje virov, pri čemer moramo bit zelo previdni, da ne obremenimo preveč glavne UI niti. Knjižnico je razvil Sergey Tarasevich, in je na voljo pod Apache Licence 2 licenco na spletni strani projekta. Je ena od najbolj priljubljenih Android knjižnic.

```
ImageLoader.getInstance().loadImage(model.getIcon(),
        new SimpleImageLoadingListener() {

        @Override
        public void onLoadingComplete(String imageUri, View view,
                        Bitmap loadedImage) {
                super.onLoadingComplete(imageUri, view,
                                loadedImage);
                mMap.addMarker(new MarkerOptions()
                                .position(destination)
                                .icon(BitmapDescriptorFactory
                                .fromBitmap(Utilities.resize(loadedImage,
                                        PlaceActivity.this)))
                                .title(model.getName()));
                loadedImage.recycle();
        }
});
```

Slika 3.12: Primer asinhronega nalaganja slike z oddaljenega spletnega vira

3.4.16 Android Maps Utils

Android Maps Utils [36] je brezplačna odprtokodna knjižnica, ki nam omogoča uporabo dodatnih komponent v Android mobilnih aplikacijah za interakcijo z Google Maps zemljevidi. Podprta so različna orodja, ki izboljšujejo interakcijo in uporabniško izkušnjo. Najbolj uporabna orodja so:

- **Gruče markerjev**
- »**Bubble icons**« - nadgradnja za obstoječe markerje
- **Heatmaps**
- **Geografski algoritmi** (razdalja do trenutne uporabniške lokacije, interpolacija...)

Razvoj knjižnice nadzoruje Google in je še vedno v beta različici. Podprta je za verzije Androida nad 2.3 (Gingerbread).

Poglavje 4

Uporabljeni algoritmi

Pri razvoju prototipa smo uporabili več že razvitih algoritmov, ki se uporabljajo v računalniški znanosti za strojno učenje, umetno inteligenco, preiskavo poti, statistiko...

Uporabljene algoritme lahko razdelimo v 2 skupini:

- **Algoritmi za klasifikacijo uporabnika v uporabniško skupino**;
- **Algoritmi za preiskavo poti**.

4.1 Algoritmi za klasifikacijo uporabnika v uporabniško skupino

Ideja našega prototipa je izboljšanje uporabniške izkušnje na način, da uporabniku predlagamo pot od točke A do točke B, pri čemer upoštevamo njegov profil in njegove interese. Za določitev profila uporabimo več algoritmov strojenega učenja za generiranje klasifikatorja, pri katerem je klasifikator definiran kot množica pravil, na podlagi katerih lahko kategoriziramo uporabnika v eno od vnaprej določenih uporabniških skupin.

Zaradi narave uporabljenih tehnologij (mobilna aplikacija, ki komunicira s strežnikom), smo si zastavili dva pogoja:

- Zaradi majhne učne množice, potrebujemo algoritme ki dosežejo čim boljše rezultate na zelo majhnem številu učnih primerov;

- Zaradi REST komunikacije med strežnikom in aplikacijo ter boljše uporabniške izkušnje, potrebujemo hitre algoritme.

Z analizo zahtev smo ugotovili da bomo v aplikaciji uporabili štiri različne algoritme in primerjali njihove rezultate. Za končno klasifikacijo uporabnika v uporabniško skupino bomo uporabili algoritem, ki ima največjo natančnost na testni množici.

4.1.1 K najbližjih sosedov

K najbližjih sosedov (v nadaljevanju KNN[1]) [39] je algoritem, ki se v umetni inteligenci in strojnem učenju uporablja za klasifikacijo in regresijo. Izhodi algoritma so:

- Pri KNN klasifikaciji je izhod algoritma klasifikacija objekta v enega od vnaprej določenih razredov;
- Pri KNN regresiji je izhod algoritma napovedana vrednost objekta.

Ideja algoritma je, da preiščemo okolico objekta in poiščemo k najbližjih sosedov. Objekt označimo z razredom, ki je najbolj zastopan v množici k najbližjih primerov. Kot mero za izračun razdalje do najbližjega soseda lahko uporabimo več različnih postopkov, najbolj priljubljena postopka pa sta Evklidska razdalja in korelacija.

D = $\{(x_1, c_1), \ldots, (x_N, c_N)\}$
x = $(x_1, \ldots, x_n)$ new instance to be classified **for** *each labelled instance* (x_i, c_i) **do**
| calculate d(x_i,x)
end
Order d(x_i, x) from lowest to highest, (i = 1, . . . ,N)
Select the K nearest instances to x: D_x^K
Assign to x the most frequent class in D_x^K

Algoritem 1: KNN algoritem - psevdokoda

[1]KNN - ang. K Nearest Neighbours

Kot mero za izračun razdalje smo uporabili Pearsonov korelacijski koeficien [40], ki izračuna linearno korelacijo(odvisnost) med spremenljivko X in spremenljivko Y. Koeficient je definiran v intervalu med -1 in 1, kjer 1 označuje popolno pozitivno korelacijo, -1 pa popolno negativno korelacijo.
Izbira optimalnega parametra k je odvisna od učne množice, najboljše rezultate pa dobimo pri vrednosti parametra med 5 in 10. KNN je zaradi enostavnosti implementacije eden od najbolj priljubljenih algoritmov v strojnem učenju. Ker učenja pri KNN skoraj ni, zanj pravimo, da uporablja postopek *lenega učenja*.

4.1.2 Odločitvena drevesa

Odločitvena drevesa [41] se pri strojnem učenju uporabljajo za klasifikacijo in regresijo. Algoritem (Slika 4.1) uporablja drevesno strukturo podatkov na način, da list definira kot razred, vozlišča kot posamezne atribute in veje kot podmnožice vrednosti atributov. Postopek klasifikacije se začne v korenu drevesa, ter se nadaljuje vse dokler ne pridemo do lista (ena pot predstavlja eno pravilo).

Pri gradnji odločitvenega drevesa uporabljamo več različnih mer za določanje najboljšega atributa, kot so informacijski prispevek, razmerje informacijskega prispevka, Gini-Index in ReliefF. Izbira optimalne mere je odvisna od strukture učne množice.

Pomanjkljivost odločitvenih dreves je, da so nižji nivoji drevesa pogosto nezanesljivi, saj se lahko preveč prilagajajo učni množici. Za preprečevanje prilagajanja uporabljamo več različnih pogojev, ki ustavijo gradnjo drevesa, ko ta postanejo nezanesljiva. Najbolj priljubljen postopek je postopek naknadnega rezanja drevesa - najprej zagradimo celotno drevo in potem porežemo vse liste, ki natančnost klasifikacije ne izboljšajo.

Drevo režemo v več korakih:

1. Sprehajamo se po drevesu od listov proti korenu;

2. Za vsako notranje vozlišče izračunamo pričakovano napako klasifikacije v poddrevesih;

3. Za vsako notranje vozlišče izračunamo še trenutno napako klasifikacije;

```
Tree-Learning (TR, Target, Attr)
        TR: training examples
        Target: target attribute

        Attr: set of descriptive attributes
        {
                Create a Root node for the tree.
                If TR have the same target attribute value ti,
                        Then Return the single-node tree, i.e.
                        Root, with target attribute = ti
                If Attr = empty (i.e. there is no descriptive
                attributes available),
                        Then Return the single-node tree, i.e.
                        Root, with most common value of Target
                        in TR
                Otherwise
                {
                        Select attribute A from Attr that best
                        classify TR based on an
                        entropy-based measure
                        Set A the attribute for Root
                        For each legal value of A, vi, do
                        {
                                Add a branch below Root,
                                corresponding to A = vi
                                Let TRvi be the subset of
                                TR that have A = vi
                                If TRvi is empty,
                                Then add a leaf node below
                                the branch with target
                                value = most common value
                                of Target in TR
                                Else below the branch, add
                                the subtree learned by
                                Tree-Learning(TRvi, Target,
                                Attr-{A})
                        }
                }
        Return (Root)
        }
```

Slika 4.1: Odločitvena drevesa - psevdokoda

4. Če je povprečna pričakovana napaka poddreves večja od pričakovane napake vozlišča, porežemo vsa poddrevesa in vozlišče spremenimo v list.

Za ocenevanje napake pri rezanju listov uporabimo ali m-oceno, ki ocenjuje klasifikacijo napako, ali pa MDL[2] pristop, ki ocenjuje dolžino kodiranja drevesa in porazdelitev razredov učnih primerov v listih.

4.1.3 Metoda podpornih vektorjev - SVM

Metoda podpornih vektorjev [42] (v nadaljevanju SVM[3]) je algoritem, ki ga v strojnem učenju uporabljamo za klasifikacijo in regresijo. Primeren je za učenje na velikih množicah z velikim številom atributov in je eden od najbolj uspešnih algoritmov za klasifikacijo in regresijo.

Algoritem se uporablja pri binarnih klasifikacijskih problemih (obstajata dva razreda), ampak ga lahko tudi razširimo za uporabo na večrazrednem problemu.

Bistvo algoritma je, da poišče optimalno hiperravnino med dvema podanima razredoma s transformacijo podatkov v večdimenzionalni prostor. Optimalna hiperravnina, ki ločuje dva razreda, je tista, ki je enako in najbolj oddaljena od najbližjih primerov obeh razredov. Najbližje primere obeh razredov potem označimo kot podporne vektorje, razdaljo optimalne hiperravnine in podpornih vektorjev pa označimo kot rob. Tako je optimalna hiperravnina tista, ki ima maksimalen rob. Vsako hiperravnino lahko zapišemo kot množico točk X, ki zadošča:

$$w * x - b = 0 \tag{4.1}$$

kjer * označuje skalarni produkt, w pa normalo na hiperravnino. Parameter b/||w|| določa odmik hiperravnine od izhodišča vzdož normale w. Potrebno je določiti takšna b in w, da maksimizirata rob.

[2]MDL - ang. Minimum Description Length
[3]SVM - ang. Singular vector machines

4.1.4 Algoritem za klasifikacijo na podlagi asociativnih pravil

Algoritem za klasifikacijo na podlagi asociativnih pravil [43] (v nadaljevanju CBA[4]) se v strojnem učenju uporablja za klasifikacijo in regresijo. Temelji na algoritmu Apriori in je sestavljen iz dveh delov:

1. Algoritem za iskanje razrednih pravil (CBA-RG[5]);
2. Algoritem za generiranje klasifikatorja (CBA-CB[6]).

Asociativna pravila definiramo kot:

$$\check{C}e\ velja\ X,\ potem\ velja\ tudi\ Y. \tag{4.2}$$

kjer sta X in Y definirana kot podmnožici vseh atributov v učni množici. Razredna asociativna pravila razširjajo asociativna pravila še s podatkom o ciljnem razredu:

$$(pogoj, y) \tag{4.3}$$

kjer je *pogoj* podmnožica vseh atributov v učni množici, ki so klasificirani v razred y. Vsakem pravilu lahko dodamo metrike, ki jih dodatno opisujejo – *podpora pravilu* in *zaupanje pravilu*. CBA algoritem podporo pravilu definira kot dva zasebna parametra: podpora pravilu in podpora pogoju. Podporo pogoju definiramo kot:

$$\frac{X}{D} \tag{4.4}$$

oziroma iščemo število pogojev (X) v celotni učni množici (D).

Podporo pravilu definiramo kot:

$$\frac{X, y}{D} \tag{4.5}$$

oziroma število vseh pogojev v učni množici, ki so označeni z razredom y.

[4]CBA - ang Classification Based on Associations
[5]CBA-RG - ang. Classification Based on Associations - Rule generator
[6]CBA-CB - ang. Classification Based on Associations - Classifier Builder

Zaupanje pravilu definiramo po formuli:

$$\frac{X, y}{\frac{X}{D}} \tag{4.6}$$

4.1.4.1 Algoritem za iskanje razrednih pravil (CBA-RG)

CBA algoritem razlikuje med štirimi vrstami razrednih asociativnih pravil:

1. **Pogosta pravila** – pravila, ki zadovoljujejo kriteriju minimalne podpore;
2. **Nepogosta pravila** – pravila, ki ne zadovoljujejo minimalnega kriterija podpore;
3. **Zanesljiva pravila** – pravila, ki zadovoljujejo kriteriju minimalnega zaupanja;
4. **Nezanesljiva pravila** – pravila, ki ne zadovoljujejo kriteriju minimalne podpore.

Ideja CBA-RG algoritma (Slika 4.2) je v tem, da z več prehodi čez učno množico poišče pogosta in zanesljiva pravila, kar dela na način, da s prvim sprehodom skozi učno množico poišče pravila dolžine 1 (v pogoju imajo en atribut). Potem zgradi nova pravila dolžine 2 z združevanjem pravil dolžine 1 in izračuna njihovo podporo ter zaupanje z novim prehodom čez učno množico. Ob koncu prehoda odstrani vsa pravila, ki niso pogosta in zanesljiva ter zgradi pravila dolžine 3. Opisani postopek se ponavlja, vse dokler obstaja možnost generiranja pravila dolžine *i+1*.

4.1.4.2 Algoritem za generiranje klasifikatorja (CBA-CB)

V drugem koraku CBA algoritma (Slika 4.3) iščemo klasifikator, ki vrne najmanjšo napako na učni množici. Za ta postopek bi morali preveriti vse podmnožice pravil, kar znese $2m$ podmnožic (m je število najdenih pravil). Ta postopek nam je popolnoma nesprejemljiv, ker od algoritma zahteva ogromno število prehodov skozi učno množico. Zaradi povedanega uporabljamo CBA-CB algoritem, ki je časovno manj potraten.

Prvi korak algoritma je razvrstitev najdenih pravil. Pravilo r_1 je pred pravilom r_2, če je uresničen eden od naslednjih pogojev:

1. r_1 ima večje zaupanje od r_2;
2. r_1 ima večjo podporo od r_2;
3. r_1 in r_2 imata enako zaupanje in podporo, ampak je r_1 generiran pred r_2.

V drugem koraku algoritma se sprehodimo skozi učno množico in poiščemo vse primere, ki so pokriti s pravilom r. Če r pravilno klasificira vsaj en primer, potem ga označimo in dodamo v množico C ter iz učne množice odstranimo vse primere, ki jih r pokriva. Zatem na učni množici definiramo privzeti razred, ki je najbolj frekvenčen razred v množici. Na koncu definiramo še napako, ki jo naredimo, če vso preostalo učno množico označimo s privzetim razredom. Ta korak ponavljamo,

```
F = {large 1-ruleitems};
CAR1 = genRules(F1);
prCAR1 = pruneRules(CAR1);
for (k = 2; Fk-1 != empty; k++) do
                Ck = candidateGen(Fk-1);
        for each data case d in D do
                Cd = ruleSubset(Ck, d);
                for each candidate c in Cd do
                        c.condsupCount++;
                        if d.class = c.class
                                c.rulesupCount++
        end
        end
        Fk= {c in Ck | c.rulesupCount >= minsup};
        CARk= genRules(Fk);
        prCARk = pruneRules(CARk);
end
CARs = Uk CARk;
prCARs = Uk prCARk;
```

Slika 4.2: CBA-RG - psevdokoda

```
R = sort(R);
for each rule r in R in sequence do
   temp = {};
   for each case d in D do
        if d satisfies the conditions of r
                store d.id in temp and mark r if
                it correctly classifies d;
        if r is marked then
                insert r at the end of C;
                delete all the cases with the
                ids in temp from D;
                selecting a default class for the current C;
                compute the total number of errors of C;
        end
 end
Find the first rule p in C with the lowest
 total numberof errors and drop all the rules after p in C;
Add the default class associated with p to end of C,
 and return C (our classifier).
```

Slika 4.3: CBA-CB - psevdokoda

dokler obstaja učna množica ali učni primer. V tretjem koraku algoritma odstranimo vsa pravila iz množice C, ki natančnosti postopka klasifikacije ne izboljšajo. Poiščemo prvo pravilo, kjer je najmanjša napaka in zanemarimo vsa ostala pravila. Klasifikator na ta način klasificira testni podatek v prizveto pravilo zadnjega pravila v množici C. Na ta način algoritem zadostuje dvema glavnima pogojema:

1. Vsi učni primeri so pokriti s pravilom, ki ima največjo prednost.
2. Vsako pravilo v C pravilno klasificira vsaj en primer iz učne množice.

4.2 Algoritem za preiskavo poti - A* algoritem

A* [44] je preiskovalni algoritem, ki je zasnovan na best-first preiskovalnem algoritmu. Ideja algoritma je v tem, da v grafu poiščemo najboljšo pot med vozliščem A

in vozliščem B, kjer so povezave med vozlišči utežene. Algoritem je v računalniški znanosti zelo popularen iz večih razlogov:

1. Lahko hitro preišče velik prostor – hitrost algoritma je O(n).

2. Lahko najde najboljšo pot – kjer pomen pojma "najboljša pot" definira sam uporabnik; to je lahko najhitrejša pot, najbolj optimalna pot z najmanjšo uporabo energije in drugo.

3. Algoritem uporablja hevristiko za vodenje preiskovanja, pri čemer hevristiko definiramo kot oceno razdalje med trenutno obravnavanim vozliščem, ter končno točko potovanja. Posledica tega je, da algoritem preišče manjše območje in tako hitrejše najde najboljšo pot do cilja. V magistrski nalogi smo za hevristično oceno uporabili Evklidsko razdaljo (v praksi se pogosto uporablja tudi Manhattan razdalja), ki smo jo še dodatno obtežili s parametrom *d*. Dodatno težo je v algoritem potrebno vpeljati, ker v realnem svetu razdalja med točko A in točko B ni Evklidska, ampak na njo vplivajo tudi ovire. Z upoštevanjem ovir raste tudi razdalja, to rast pa prikažemo s parametrom *d*. Empirično smo testirali hitrost iskanja poti v odvisnosti od parametra d in smo prišli do ugotovitve, da algoritem najhitrejše pride do rezultata pri $d = 10$.

V naši implementaciji algoritma težo povezave med vozlišči v grafu določamo sprotno, na naslednji način:

$$F = g + h * d \tag{4.7}$$

kjer g definiramo kot

$$\begin{aligned} g(n+1) = g(n) + razdalja\_med\_trenutno\_in\_naslednjo\_točko \\ + g_{vremenska\_napoved} + g_{razmere\_na\_cesti} \end{aligned} \tag{4.8}$$

Vrednosti g parametrov za vremensko napoved določimo po Tabeli 4.1, dokler je vrednost parametra g za stanje v prometu vedno 100. Razlog je v nepopolnih podatkih storitve, iz katere pridobivamo poročilo o stanju v prometu (struktura podatkov nam onemogoča razlikovanje problema - npr. ne vemo ali je cesta zaprta zaradi prometne nesreče, ali zaradi kakšne parade).

```
function A*(start,end)
    // Already processed nodes
    closedset := empty
    // Nodes for proccessing
    openset := {start}
    // Nodes and their predecessors
    came_from := the empty map
    // Cost of path from start using best path
    g_score[start] := 0
    f_score[start] := g_score[start] +
            heuristic_score(start, end)

    while openset not empty
        current_node := node in openset with smallest
                value of f\_score[]
        if current_node = end
            return reconstruct_path(came_from, end)

        remove current node from  openset
        add current node to closedset
        for each neighbour in neighbour_nodes(current_node)
            if neighbour in closedset
                continue
            g_current := g_score[current_node] +
                        distance_between(current_node,neighbour)
            if neighbour not v openset or g_current <
            g_score[neighbour]
                came_from[neighbour] := current_node
                g_score[neighbour] := g_current
                f_score[neighbour] := g_score[neighbour] +
                                heuristic_score(neighbour, end)
                if neighbour not in openset
                    add neighbour to openset

    return failure
```

Slika 4.4: A* algoritem - iskanje poti

Opis dogodka	**Vrednost**
Rahel dež	10
Zmeren dež	20
Močan dež	40
Rahelo sneženje	10
Sneženje	20
Močno sneženje	40
Nevihta z dežjem	50
Ekstremno neurje	1000

Tabela 4.1: Vrednost parametra g

```
function reconstruct_path(came_from, current_node)
    if current_node in came_from
        p := reconstruct_path(came_from,
                    came_from[current_node])
        return (p + current_node)
    else
        return current_node
```

Slika 4.5: A* algoritem - rekonstrukcija poti

Poglavje 5

Predlog rešitve

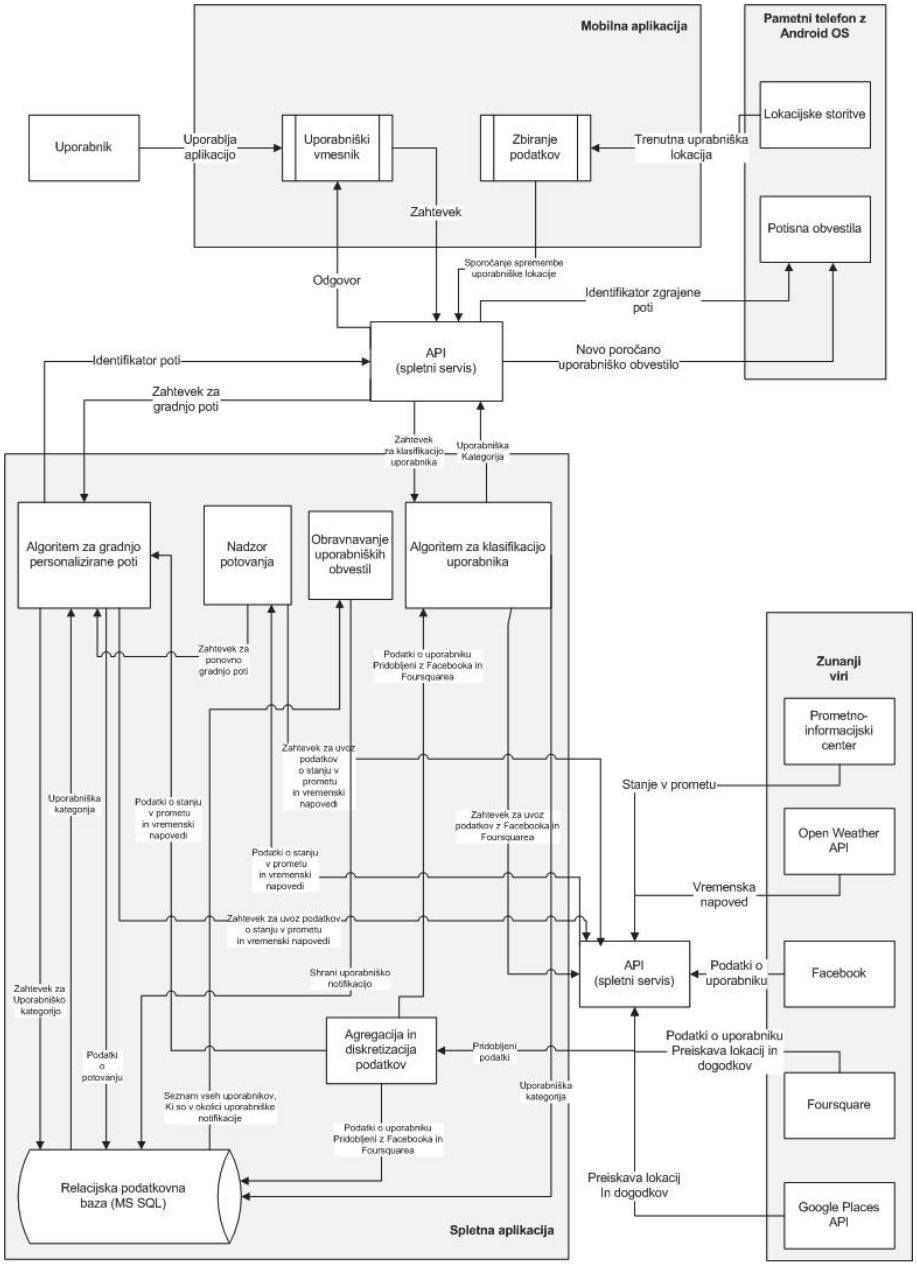
Ideja našega sistema je nadgradnja in izboljšava uporabniške izkušnje GPS aplikacij v treh različnih vidikih:

1. Prilagajanje načina dela aplikacije vsakemu uporabniku posebej;
2. Izboljšanje varnosti;
3. Močna uporaba družabne komponente.

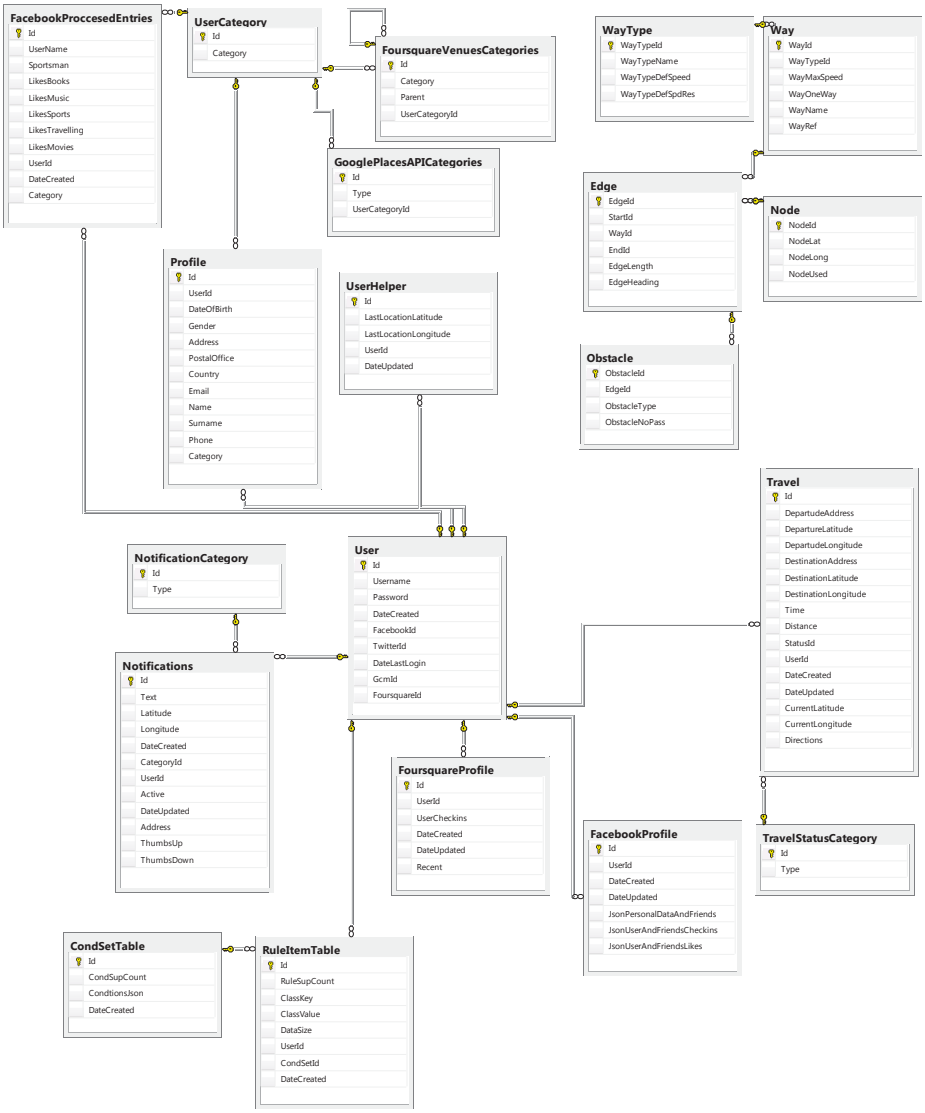
Zaradi same arhitekture sistema (spletna in mobilna aplikacija), smo si zastavili naslednja cilja:

1. Aplikacija mora s čim večjo natančnostjo določiti profil uporabnika na omejenem naboru informacij.
2. V sistemu bomo uporabljali enostavne in časovno nezahtevne algoritme, ker je potrebno zagotoviti hitro komunikacijo med vsemi komponentami sistema(spletna in mobilna aplikacija).

5.1 Shema sistema in podatkovni model



Slika 5.1: Shema sistema



Slika 5.2: Podatkovni model

5.2 Načrtovanje sistema

V nadaljevanju bomo podrobno predstavili predlagano rešitev, njeno arhitekturo in področja v katerih naša rešitev izboljšuje delovanje obstoječih GPS aplikacij.

5.2.1 Arhitektura

Sistem, ki smo ga razvili, je sestavljen iz dveh komponent – spletne aplikacije in mobilne aplikacije za Android pametne telefone.

Spletna aplikacija je temeljna komponenta, ki z uporabo javno dostopnega API-ja komunicira z odjemalcem – ki je mobilna aplikacija. Aplikacija je razvita v ASP.NET MVC tehnologij, in se izvaja na oddaljenem Microsoftovem strežniku.

Mobilna aplikacija je narejena za Android pametne telefone in podpira verzije Androida nad 4.0 zaradi uporabe Bootstrap grafičnih komponent, ki za uporabo na starejših verzijah niso primerne. Aplikacija je razvita v skladu z uradnimi Android smernicami za razvoj mobilnih aplikacij. Pogoj za uporabo je namestitev Google Play Services aplikacije ter aktivne lokacijske storitve v času uporabljanja aplikacije. Če mobilna aplikacija ne zadostuje obema pogojema, bo o temu obvestila uporabnika.

Vstop v sistem je možen le z uporabo uporabniških računov. Registracija novega računa bo omogočena v spletni in mobilni aplikaciji. Pri deaktivaciji uporabniškega računa se bodo vsi uporabniški podatki avtomatsko izbrisali.

V sistemu bomo uporabljali zasebne informacije o uporabnikih, pridobljene iz Facebook in Foursquare uporabniških računov. Uvoz podatkov bo možen le z uporabo spletne aplikacije, in sicer zaradi zagotavljanja večje varnosti. Od uporabnikov je potrebno pridobiti pooblastilo za dostop do informacij, slednje pa so shranjene v oddaljeni podatkovni bazi in se pri deaktivaciji uporabniškega računa avtomatsko izbrišejo. Sistem ne omogoča avtomatske sinhronizacije podatkov zaradi varnostne politike - uporabnik ima nadzor nad celotno komunikacijo med sistemom in socialnimi omrežji.

V sistemu bomo uporabili več odprtokodnih brezplačnih knjižnic, pri čemer bo uporaba samega sistema brezplačna. Izvorna koda spletne in mobilne aplikacije bo dostopna na Github povezavi [33].

5.2.2 Prva izboljšava - prilagajanje načina dela aplikacije uporabnikom

V okviru prve izboljšave vsakega uporabnika najprej razvrstimo v uporabniško skupino, in mu zatem v odvisnosti od uporabniške skupine generiramo personalizirano pot od začetne do končne točke potovanja.

5.2.3 Klasifikacija uporabnika v uporabniško skupino - določanje profila

Aplikacija omogoča uporabo n različnih profilov, ampak smo zaradi enostavnosti testiranja določili tri različne profile:

1. Traveller - oseba, ki rada potuje;
2. Sportsman - oseba, ki ima rada šport ali je športnik;
3. Urban - oseba, ki ima rada glasbo, knjige in filme.

Kategorije profilov smo določili s testiranjem testne množice oseb (n = 20). Osebe so izpolnile anketo, kjer so se na podlagi svojih Facebook in Foursquare uporabniških računov klasificirali v predlagane kategorije profilov: Traveller, Sportsman, Urban, Nature (oseba, ki svoj prosti čas rada preživlja v naravi), Racer (oseba, ki obožuje dirke), Archaeologist (oseba, ki ima rada zgodovino) ... Ker je odstotek uporabnikov, ki so se klasificirali v druge kategorije in ne v kategorije Traveller, Sportsman in Urban izredno majhen (manj kot 2 osebi), so te kategorije za potrebe testiranja zanemarjene.

Profil določimo z analizo podatkov, ki jih uporabniki izvozijo iz svojih Facebook in Foursquare uporabniških računov v naš sistem (skozi javno dostopen API). S Foursquarea se uvozijo podatki o obiskanih lokacijah in tudi podatki o lokacijah, ki so jih obiskali naši prijatelji. S Facebooka pridobimo veliko več podatkov, in ti so:

- Osebni podatki o uporabniku, kot so datum rojstva, spol, mesto prebivališča ...

- Podatki, ki dodatno opisujejo uporabnika – prebrane knjige, najljubši športniki, seznam športov s katerimi se uporabnik ukvarja, seznam predvajane glasbe, seznam pregledanih filmov in nanizank...
- Seznam obiskanih lokacij;
- Vsi uporabniški všečki.

Vsi podatki se uvozijo za posameznega uporabnika in tudi za njegovih 350 prijateljev[1]. Ko v sistem pridobimo vse informacije, poiščemo KPI-je, ki so pomembni za testno množico profilov:

1. isSporstman - ali je oseba športnik;
2. likesSports - ali ima oseba rada šport;
3. likesBooks - ali ima oseba rada knjige;
4. likesMovies - ali ima oseba rada filme;
5. likesMusic - ali ima oseba rada glasbo;
6. likesTravelling - ali ima oseba rada potovanja;
7. simmilarFriends - seznam podobnih prijateljev(po kategorijah všečkov).

KPI-ji so določeni na podlagi analize podatkov pridobljenih iz omrežja Facebook in njihove preslikave v obstoječe uporabniške kategorije. V primeru večih uporabniških kategorij (ali različnih kategorij), je potrebno določiti tudi nov seznam KPI-jev. Ker naš sistem za klasifikacijo uporablja metode strojnega učenja, se vsi pridobljeni podatki najprej diskretizirajo v KPI-je v 3 različne kategorije:

1. UNKNOWN - če podatek ni poznan (vrednost 0);
2. FALSE - če lahko zaključimo, da uporabnika te informacije ne zanimajo (vrednost -1);
3. TRUE - če lahko zaključimo, da uporabnika te informacije zanimajo (vrednost 1).

KPI	UNKOWN	FALSE	TRUE
isSporstman	∅	0	1
likesSports	∅	0-2	>=3
likesBooks	∅	0-2	>=3
likesMovies	∅	0-2	>=3
likesMusic	∅	0-7	>=8
likesTravelling	∅	0-2	>=3

Tabela 5.1: Diskretizacijski koši

Podatki se diskretizirajo z uporabo diskretizacijskih košev glede na Tabelo 5.1. Meje med diskretizacijskimi koši so postavljene različno – vsaka informacija ima svoje meje za koše zaradi različnih KPI-jev[2]. Določene so na omejeni množici uporabnikov, ki so izpolnili anketo in z oceno med 1 in 10 ocenili vsak KPI. Zatem smo analizirali njihove profile in z mediano[3] določili vrednosti mej med koši.

Ko so podatki diskretizirani (Tabela 5.2) jih shranimo v podatkovno bazo. Potem lahko klasificiramo uporabnika v uporabniško skupino z uporabo štirih algoritmov strojnega učenja, ki časovno in procesorsko niso zahtevni:

1. K najbližjih sosedov;
2. Odločitvena drevesa;
3. Metoda podpornih vektorjev;
4. Klasifikacija na podlagi asociativnih pravil.

Kot učno množico uporabimo vrednosti KPI-jev že obstoječih uporabnikov sistema (med njimi so lahko tudi uporabniški prijatelji s Facebook ali Foursquare

[1]Število prijateljev je omejeno na 350, ker je število brezplačnih zahtevkov na Facebook API časovno in lokacijski omejeno

[2]Na primer, lahko zaključimo da uporabnika zanimajo potovanja, če ima uporabnik naštetih več kot 10 različnih obiskanih lokacij. Po drugi strani, če se uporabnik ukvarja z enim športom, potem lahko zaključimo da je uporabnik športnik.

[3]Mediana - srednja vrednost zaporedja števil.

KPI	kategorija
isSporstman	FALSE
likesSports	TRUE
likesBooks	TRUE
likesMovies	TRUE
likesMusic	TRUE
likesTravelling	TRUE

Tabela 5.2: Primer vrednosti KPI-jev za uporabnika Željka Plesca, kategoriziranega v kategorijo TRAVELLER

omrežja). Ko zgradimo klasifikator, primerjamo rezultate vseh uporabljenih algoritmov in z najbolj natančnim algoritmom klasificiramo uporabnika v uporabniško kategorijo. Ta podatek tudi shranimo v bazo.

Postopek klasifikacije lahko tudi kadarkoli ponovimo. Ker se uvoženi podatki s Foursquarea in Facebooka ne sinhronizirajo avtomatsko (zaradi varnostne politike), je pred samo klasifikacijo potrebno še enkrat uvoziti nove podatke, ker obstaja velika možnost, da nas sistem zopet klasificira v isto uporabniško skupino.

5.2.4 Gradnja personalizirane poti

Ko smo klasificirali uporabnika v uporabniško skupino, lahko začnemo s postopkom gradnje personalizirane poti od začetne do končne točke za posameznega uporabnika.

Kot algoritem za iskanje poti smo izbrali A* preiskovalni algoritem zaradi njegove preprostosti in enostavnosti uporabe. Postopek iskanja poti se odvija v več korakih:

1. Najprej algoritem poišče najbolj optimalno pot od začetne do končne točke. Sistem omogoča definiranje pomena ”najboljša pot” za vsakega uporabnika, vendar smo zaradi enostavnosti testiranja omogočili dve možnosti (Slika 5.3):

   - najkrajša pot,

Foursquare Venues lokacija	**Kategorija**
DM Drogerie Markt	Cosmetics Shop
Avtobusni kolodvor Ljubljana	Bus station
Ljubljanski grad	Castle

Tabela 5.3: Primer zanimivih lokacij pridobljenih s Foursquare Venues za kategorijo TRAVELLER

- časovno nakrajša pot.

Kot vhodne podatke v A* algoritem uporabljamo lokalno shranjeni OpenMaps zemljevid v obliki seznama vozlišč, poti in relacij. Zaradi omejenih strežniških virov je v testiranju omogočena le navigacija po Ljubljani (več kot 69 000 vozlišč).

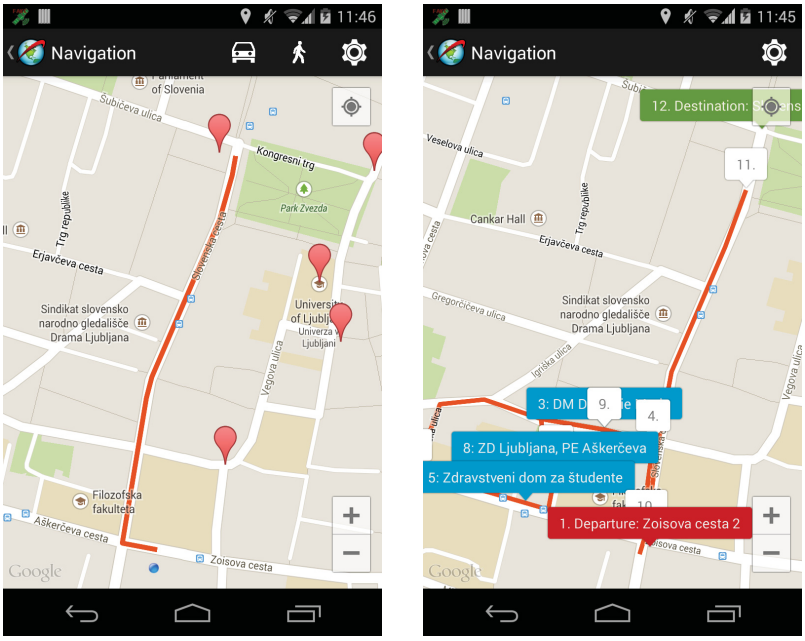
2. Ko smo zgradili najbolj optimalno pot se še enkrat sprehodimo skozi celotno pot in vsakih 300 metrov[4] pridobimo iz različnih spletnih virov informacije, ki so lahko uporabniku zanimive (po uporabniški kategoriji). Pri testiranju smo omogočili pridobivanje informacij z Google Places API in Foursquare Venues, ker je dostop brezplačen in so informacije kategorizirane. V Tabeli 5.3 smo podali primer preslikave dela Foursquare Venus kategorij v uporabniške kategorije našega sistema.

3. Ko pridobimo vse zanimive informacije (Tabela 5.4), jih najprej sortiramo po razdalji. Potem s ponovno uporabo A* algoritma poiščemo pot, ki nam poveže vse pridobljene lokacije in dogodke z najbolj optimalno potjo iz prvega koraka algoritma.

4. Na izhodu algoritma pridobimo seznam točk, ki nam predstavljajo pot od začetne do končne točke, ki obišče tudi vse uporabniško zanimive lokacije in dogodke.

Zaradi omejitve velikosti potisnih obvestil se seznam točk shrani lokalno v podatkovno bazo in se uporabniku posreduje obvestilo z enoličnim identifikatorjem.

[4]Privzeta razdalja, uporabnik lahko nastavi svojo vrednost



(a) Najkrajša pot (b) Personalizirana pot

Slika 5.3: Primerjava najkrajše in personalizirane poti

Google Places API kategorija	Uporabniška kategorija
book_store	URBAN
bowling_alley	URBAN
bowling_alley	URBAN
movie_theater	URBAN
bus_station	TRAVELLER
train_station	TRAVELLER
stadium	SPORSTMAN

Tabela 5.4: Preslikava Google Places API kategorij v uporabniške kategorije

Ko uporabnik klikne na prejeto obvestilo na mobilni napravi, se v ozadju opravi klic na spletno aplikacijo, ki uporabniku posreduje seznam točk.

5.2.5 Druga izboljšava - zagotavljanje večje varnost

V okviru druge izboljšave smo si zastavili cilj, da bomo s čim manjšo uporabniško interakcijo poskušali zagotoviti večjo uporabniško varnost. To smo omogočili na način, da poskušamo pred začetkom samega potovanja predvideti čim večje število morebitnih težav in tako ustrezno vplivati na postopek gradnje poti. Sistem omogoča pridobivanje informacij, ki bi lahko vplivale na potovanje, iz različnih spletnih virov. Zaradi enostavnosti testiranja smo omogočili interakcijo iz 2 spletnih virov:

- Open Weather Map - podatki o trenutnem vremenu,
- Prometno informacijski center - podatki o trenutnem stanju v prometu.

Pridobljene podatke uporabimo za nadgradnjo postopka iskanja poti z A* algoritmom na način, da dodatno obtežimo parameter g (ki usmerja hevristiko) v primeru, če se trenutno obravnavno vozlišče nahaja v okolici 15 metrov od lokacije morebitne nevarnosti. Dodatna teža v primeru prometne nesreče ali zaprte ceste znaša 100, v primeru slabih vremenskih razmer pa se teža določi na podlagi Tabele 4.1. Na ta način smo zagotovili, da trenutni odsek potovanja ni najbolj optimalen, in A* algoritem poišče drugo pot, ki se izogne morebitni nevarnosti.

Ker v času potovanja lahko pride do spremembe vremenskih razmer ali do prometne nesreče, smo sistem implementirali na način, da sledi lokaciji uporabnika na njegovi poti. Mobilna aplikacija (v primeru aktivnih lokacijskih storitev in povezave z internetom) ves čas sporoča trenutno lokacijo uporabnika spletni storitvi. Če se je uporabniška lokacija spremenila za več kot n metrov (privzeta vrednost je 20 metrov), se znova pridobijo podatki s spletnih virov. Sistem analizira pridobljene podatke in če se uporabnik nahaja v bližini nevarnosti (privzeta vrednost je 300 metrov), A* algoritem znova poišče novo pot, ki se bo zaradi večje teže izognila odseku poti kjer se nahaja nevarnost. Ko algoritem zgradi novo pot, najprej lokalno shranimo seznam točk v podatkovno bazo in potem pošljemo identifikator preko obvestila na mobilno napravo. Ko uporabnik klikne na prejeto obvestilo, se mu avtomatsko v mobilni aplikaciji osvežijo vsi podatki, vključno z novo predlagano potjo.

Z uporabo naštetih postopkov smo uresničili več ciljev:

- večja varnost uporabnika,
- manjša interakcija z aplikacijo,
- osveževanje informacij o poti v realnem času.

5.2.6 Tretja izboljšava - uporaba socialne komponente

Uporabo socialne komponente smo zagotovili na način, da smo omogočili asinhrono komunikacijo med uporabniki z uporabo privzetih Google-ovih GCM potisnih obvestil. Obvestilo smo v okviru magistrske naloge definirali kot krajšo informacijo(največ 200 znakov), ki lahko izboljša uporabniško izkušnjo drugim uporabnikom našega sistema. Vsako obvestilo je sestavljeno iz:

1. Lokacije obvestila (geografske koordinate);
2. Besedila obvestila - omejen na 200 znakov;
3. Ocene obvestila - število uporabnikov, ki so informacijo ocenili kot pozitivno ali negativno;
4. Datuma obvestila;
5. Avtorja obvestila.

Pogoj za uporabo obvestil je namestitev Google Play Services aplikacije[5] na mobilno napravo. V primeru, da uporabnik nima nameščene Play Services aplikacije ali pa je verzija aplikacije zastarela, se uporabniku pri uporabi mobilne aplikacije prikaže sporočilo o potrebni posodobitvi ali namestitvi Play Services aplikacije.

Zaradi zagotavljanja varnosti in boljše uporabniške izkušnje smo omogočili kategorizacijo in filtriranje obvestil. Sistem omogoča kategorizacijo obvestil v n različnih kategorij, vendar smo zaradi enostavnosti testiranja določili tri različne kategorije:

[5]GCM potisna obvestila niso del standardnega Android SDK-ja. Zaradi tega je potrebno na mobilni napravi zagotoviti Google Play Services aplikacijo, ki nam omogoča uporabo dodatnih Googlovih storitev, kot so GCM obvestila ali Google Maps zemljevidi. Vse Android verzije nad 4.1. imajo privzeto naloženo Play Services aplikacijo in njena odstranitev ni možna.

Slika 5.4: Primer sprejetja novega uporabniško posredovanega obvestila

1. Sightseeing - lokacije, ki jih uporabniki priporočajo za ogled,
2. Police - lokacija radarjev in policijskih kontrolnih točk,
3. Restaurant - gostilne in restavracije, ki jih priporočajo uporabniki.

Za ustvarjanje obvestila mora uporabnik najprej določiti lokacijo obvestila. Omogočen sta dve različni izbiri: (1) lokacijo izberemo s klikom na želeno lokacijo na zemljevidu ali (2) obvestilo postavimo na trenutno uporabniško lokacijo. Zatem vnesemo besedilo obvestila in izberemo njegovo kategorijo. V primeru aktivne povezave z internetom se obvestilo prenese v spletno storitev. Ko storitev prejme obvestilo, jo najprej shrani v podatkovno bazo in zatem pošlje vsem uporabnikom v radiju 5 kilometrov od lokacije obvestila.

Zaradi velikega števila obvestil smo sistem načrtovali na način, da se obvestila osvežujejo v mobilni aplikaciji, ko je trenutna uporabniška lokacija v uporabniško zastavljenem radiju od samega obvestila. Privzeto je radij definiran na 5 kilometrov. Kot že omenjeno, mobilna aplikacija ves čas sporoča svojo lokacijo v spletno storitev, kjer se analizira poročana uporabniška lokacija in pridobi seznam vseh aktivnih obvestil, ki so v bližini uporabniške lokacije. Ta seznam se zatem s pomočjo potisnih obvestil pošlje uporabniku na mobilno napravo. S klikom na obvestilo v Android nadzorni vrstici (Slika 5.4) se obvestilo prikaže na zemljevidu skupaj z besedilom, avtorjem in datumom obvestila.

Z analizo in testiranjem sistema smo hitro ugotovili, da je potrebno zagotoviti tudi kvaliteto obveščanja. V sistemu nimamo možnost preverjanja natančnosti tekstov in poročane lokacije obvestil, zato smo oceno kvalitete prepustili samim uporabnikom aplikacije. Vsako obvestilo ima svojo oceno, ki je definirana kot razmerje med številom uporabnikov, ki so obvestilo ocenili kot pozitivno, in številom uporabnikov, ki so obvestilo ocenili kot negativno. Ko je razmerje med negativno in pozitivno oceno večje od 20, se obvestilo deaktivira in ni več na voljo. V primeru več pritožb na obvestila posameznega uporabnika deaktiviramo vse njegove

informacije in mu začasno blokiramo uporabniški račun.

Življenjska doba obvestila ni omejena, s čimer smo zagotovili, da so aktivne kvalitetne informacije vedno na voljo vsem uporabnikom sistema.

Z uporabo vseh naštetih postopkov smo izpolnili več ciljev:

- informacije so kratke,
- kategorizacijo informacij,
- kvaliteto informacij in,
- dostop do obvestil, katerih uporaba je brezplačna in vedno na voljo.

Poglavje 6

Implementacija rešitve

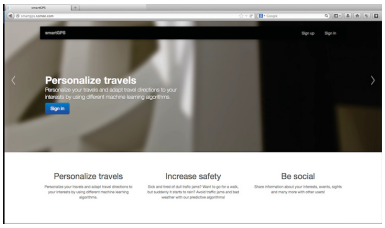
Rešitev smo implementirali kot sistem, ki je sestavljen iz 2 ločenih komponent:

1. Spletne aplikacije,
2. Mobilne aplikacije.

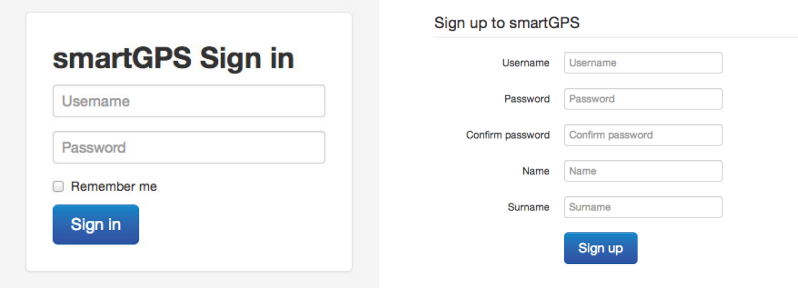
Komponente medsebojno komunicirajo z uporabo javno dostopnega REST API-ja. V nadaljevanju bomo podrobno predstavili implementacijske podrobnosti vsake komponente.

6.1 Spletna aplikacija

Spletna aplikacija je prosto dostopna na http://smartgps.somee.com. Naslovna stran vsebuje osnovne informacije o sami aplikaciji, povezavo na registracijo novih uporabnikov ter povezavo na prijavo obstoječih uporabnikov (Slika 6.1).



Slika 6.1: Naslovna stran spletne aplikacije



(a) Prijava (b) Registracija

Slika 6.2: Prijava in registracija v spletno aplikacijo

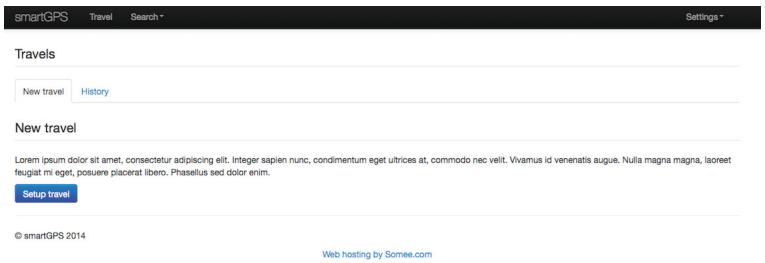
6.1.1 Registracija in prijava v aplikacijo

Vstop v aplikacijo je možen s smartGPS uporabniškim računom (Slika 6.2a). Če uporabnik že poseduje uporabniški račun se lahko prijavi v aplikacijo s klikom na gumb "Sign in". Zatem je preusmerjen v novo okno, kjer vpiše svoje uporabniške podatke. V primeru uspešne prijave strežnik registrira novo sejo in uporabnik je preusmerjen v svojo uporabniško ploščo (ang. User Dashboard).

Uporabnik lahko tudi ustvari nov uporabniški profil s klikom na gumb "Sign up". Odpre se novo okno z registracijsko formo (Slika 6.2b). Ko so vpisani vsi potrebni podatki, lahko potrdimo registracijo s klikom na gumb "Sign up".

6.1.2 Potovanja

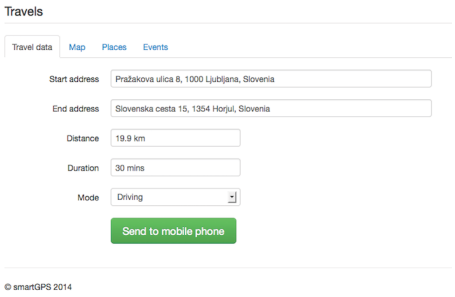
S klikom na opcijo "Travel" v nadzorni vrstici uporabniškega vmesnika lahko ustvarimo novo potovanje ali pregledamo zgodovino že opravljenih potovanj. (Slika 6.3)



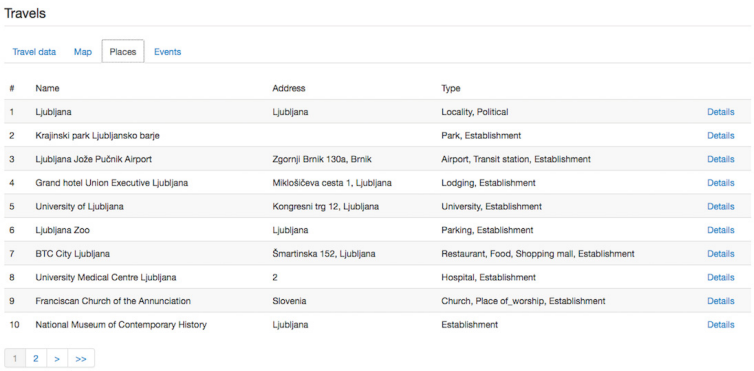
Slika 6.3: Spletna aplikacija, opcija "Travel"

6.1.2.1 Novo potovanje

S klikom na možnost "Setup travel" aplikacija ponuja možnost nastavitve začetne in končne točke našega potovanja bodisi preko naslova ali preko geografskih koordinat. Potovanje potrdimo s klikom na gumb "Continue" ter nas aplikacija preusmeri v novo okno, kjer so podane podrobnosti potovanja:



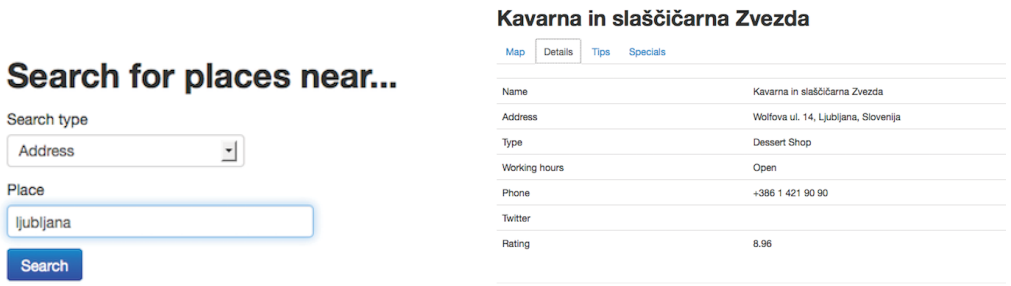
(a) "Travel data" zavihek - podrobnosti o potovanju



(b) "Places" zavihek - primer uporabniško zanimivih lokacij v bližini potovanja

Slika 6.4: Novo potovanje - podrobnosti

- **Travel data** – informacije o potovanju, kot so čas potovanja, razdalja, naslov začetne in končne lokacije (Slika 6.4a).
- **Maps** – zemljevid z izrisano potjo med začetno in kočno točko.
- **Places** – seznam zanimivih lokacij pridobljenih z Google Places, ki so v



(a) Primer iskanja lokacij v bližini Ljubljane

(b) Detajli o lokaciji, pridobljeni s Foursquare Venues AP

Slika 6.5: Iskanje lokacij in dogodkov

bližini našega potovanja (Slika 6.4b). S klikom na opcijo "Details" dobimo dodatne informacije o lokaciji.

- **Events** – seznam zanimivih dogodkov in lokacij pridobljenih s Foursquarea, ki so v bližini našega potovanja. Podobno kot pri "Places" zavihku, s klikom na opcijo "Details" dobimo dodatne informacije o dogodku ali lokaciji.

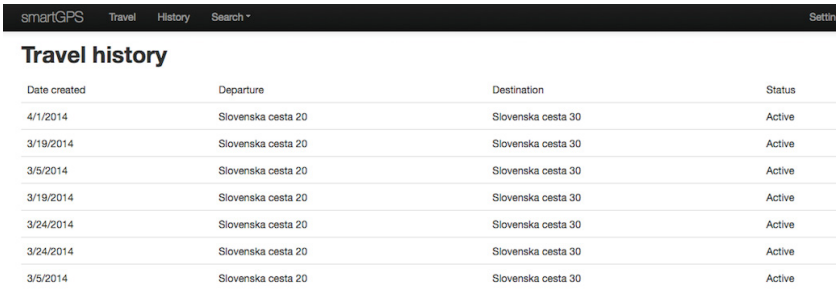
Če je uporabnik sinhroniziral svojo mobilno aplikacijo s spletnim portalom, potem si lahko s klikom na gumb "Send to mobile phone" pošlje podrobnosti novega potovanja na mobilno napravo z uporabo potisnih obvestil.

6.1.2.2 Iskanje lokacij in dogodkov

Uporabnik lahko tudi poišče vse lokacije in dogodke v bližini kateregakoli mesta z opcijo "Search" v nadzorni vrstici. Omogočena sta 2 načina iskanja:

- **Places** – iskanje po Google Places API,
- **Events** – iskanje po Foursquare Venues (Slika 6.5a).

S klikom na gumb "Search", aplikacija pridobi vse zanimive lokacije ali dogodke, ki so v radiju 500 metrov od vnesenega mesta. Dogodki in lokacije so prikazani v seznamu, sortirani po razdalji od iskanega mesta. Uporabnik lahko tudi pridobi dodatne informacije o dogodku/lokaciji s klikom na opcijo "Details" (Slika 6.5b).



Slika 6.6: Zgodovina potovanj

6.1.2.3 Zgodovina potovanj

Uporabnik lahko preveri zgodovino svojih potovanj s klikom na opcijo “History” v nadzorni vrstici. Zatem je preusmerjen na novo stran, kjer so na seznamu našteta vsa njegova potovanja (Slika 6.6) skupaj s kratkim opisom:

- **datum potovanja**,
- **začetna lokacija**,
- **končna lokacija**
- **Status** - kjer ”Active” pomeni, da je potovanje še vedno aktivno, ”Paused”, da je potovanje trenutno ustavljeno in ”Finished”, da je potovanje končano.

6.1.2.4 Uporabniški profil

Uporabniški profil je dostopen skozi “My profile” opcijo v “Settings” meniju. Podatki so kategorizirani v štiri različne zavihke:

- **smartGPS profile** - podatki o uporabniku, kot so uporabniško ime, ime, priimek, e-pošta . . .
- **Facebook** - zavihek nam omogoča uvoz podakov z uporabniškega računa na Facebooku. Ko je uvoz končan, se nad pridobljenimi podatki izvede algoritem za analizo uporabnika. Vsi pridobljeni podatki se uporabijo kot vhod v algoritem za klasifikacijo uporabnika v uporabniško skupino (Slika 6.7a).

List of similar friends by likes — Friends analyzed: 156

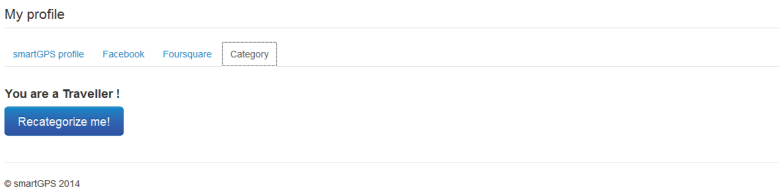
#	Friend	Number of shared likes
1	Martin Tutek	20
2	Anita Rehorić	18
3	Alen Furjanić	18
4	Ivan Dama Kardašijan	17
5	Damir Svrtan	17
6	Dražen Perić	16
7	Marko Kodrič	16
8	Ivan Kocijan	16
9	Dario Penić	15
10	Jernej Vrčko	15

(a) Seznam najbolj podobnih Facebook prijateljev, sortiranih po številu skupnih všečkov; uporabnik Željko Plesac

Most frequend user visited places

#	City	Number of checkins
1	Ljubljana,	6

(b) Seznam najbolj pogosto obiskanih lokacij pridobljenih iz Foursquarea; uporabnik Željko Plesac



(c) Primer uporabnika, kategoriziranega v skupino "Traveller"; uporabnik Željko Plesac

Slika 6.7: Analiza Facebook in Foursquare uporabniških profilov

- **Foursquare** - zavihek nam omogoča uvoz podatkov s Foursquare uporabniškega računa. Podobno kot pri Facebooku, ko je uvoz profila končan, se izvede algoritem za analizo uporabnika. Vsi pridobljeni podatki se uporabijo kot vhod v algoritem za klasifikacijo uporabnika v uporabniško skupino (Slika 6.7b).

- **Category** - zavihek, kjer lahko preverimo trenutno uporabniško kategorijo (v katero nas kategorizira algoritem za kategorizacijo uporabnikov). Pogoj za kategorizacijo je uvoz Facebook in Foursquare profila (Slika 6.7c).

  S klikom na gumb "Recategorize me!" lahko še enkrat izvedemo algoritem za klasifikacijo uporabnika. Če medtem nismo osvežili Facebook ali Foursquare podatkov s smartGPS aplikacijo, lahko pridobimo enak rezultat.

6.2 Mobilna aplikacija

Mobilna aplikacija je prosto dostopna na Github povezavi [46].

6.2.1 Registracija uporabnika in prijava v aplikacijo

Vstop v mobilno aplikacijo je možen s smartGPS uporabniškim računom (Slika 6.8a). V primeru novega uporabnika se lahko uporabniški račun ustvari skozi aplikacijo s klikom na gumb "Register".

Aplikacija nas preusmeri na obrazec za registracijo, kjer vpišemo vse potrebne osebne podatke (Slika 6.8b). S klikom na gumb "Register" se podatki prenesejo na strežnik in v primeru uspešne registracije aplikacija avtomatsko prijavi uporabnika v sistem in ga preusmeri v glavni meni. V primeru napake se na zaslonu pojavi okno z opisom napake.

6.2.2 Novo potovanje

Uporabnik lahko ustvari novo potovanje s klikom na opcijo "Travel" v uporabniškem meniju. Zatem nas aplikacija preusmeri v novo okno, kjer lahko definiramo končno točko potovanja bodisi po naslovu ali po geografskih koordinatah (Slika 6.9a). Za začetno točko se privzeto vzame trenutna uporabniška lokacija, ki jo pridobimo

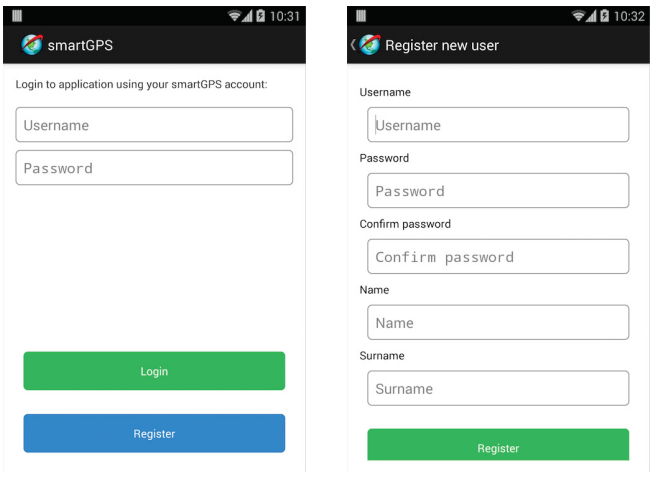
z uporabo lokacijskih storitev, ki so na voljo na mobilni napravi (GPRS, WI-FI, GPS). S klikom na gumb "Find destination" aplikacija poišče vse lokacije, ki ustrezajo naši poizvedbi.

Ko izberemo destinacijo, s klikom na gumb "Continue" nadaljujemo v naslednji ekran, kjer nam aplikacija ponudi kratek povzetek našega planiranega potovanja (Slika 6.9b).

Uporabnik lahko v nadzorni vrstici izbere način potovanja – hoja ali vožnja. Vsak način potovanja ima svoje podatke o razdalji in času potovanja, ki se ustrezno osvežujejo na ekranu.

Naprej lahko nadaljujemo z izbiro ene od treh ponujenih možnosti:

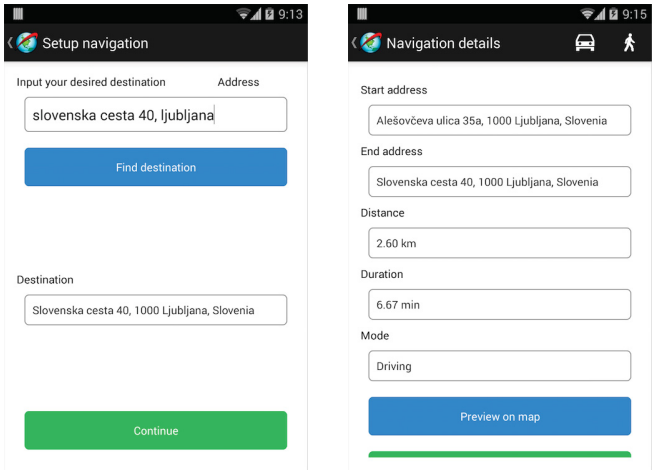
1. **Preview on map** - preverimo kakšno pot nam svetuje aplikacija pred začetkom potovanja.
2. **Regular navigation** – navigacija od začetne do končne lokacije, uporablja privzeti Google Directions API z ustreznim načinom potovanja (avto ali hoja).
3. **smartGPS navigation** – "pametni" način potovanja, kjer se potovanje na-



(a) Prijava v aplikacijo

(b) Registracija novega uporabnika

Slika 6.8: Prijava in registracija v aplikacijo



(a) Definiranje končne točke potovanja

(b) Povzetek planiranega potovanja

Slika 6.9: Novo potovanje

črtuje, upoštevajoč uporabniški profil.

Če izberemo drugo ali tretjo opcijo, nadaljujemo v naslednji, najpomebnejši ekran aplikacije – potovanje.

6.2.3 Potovanje

Aplikacija na začetku naloži pot med začetno in končno točko, uporabniško zanimive lokacije s Foursquarea in Google Places ter vsa obvestila, ki se nahajajo v uporabniško določenem radiju od trenutne lokacije. Če lokacijske storitve niso na voljo (npr. uporabnik je izklopil opcijo "Location settings" v nastavitvah telefona zaradi manjše porabe baterije), nas aplikacija obvesti z ustreznim sporočilom.

Začetna točka je označena z rdečim oblakom, končna točka pa z zelenim oblakom (Slika 6.10). V oblake so vpisane oznake lokacije ("Departure" ali "Destination"), ter njen naslov. Trenutna uporabniška lokacija je prikazana z modrim krogcem. Krogec se premika s spremembo uporabniške lokacije.

Predlagana pot je označena z oranžno bravo. Na poti sta dve različni vrsti oblakov:

- **Beli oblak** - označuje naslednji korak v potovanju. Uporabnik sledi kora-

Slika 6.10: Potovanje z mobilno aplikacijo

kom, in na ta način izpolnjuje potovanje. Aplikacija na nikakršen koli način ne nadzoruje uporabnika, temveč je na uporabniku samem, ali bo izpolnil predlagano pot, ali pa se bo izognil delu samega potovanja.

- **Moder oblak** – zanimive lokacije, ki jih aplikacija pridobi z različnih virov in z opisom ustrezajo uporabniškemu profilu. Prikazan je le v"pametnem" načinu potovanja, in je del samega potovanja. Vse druge lokacije so prikazane s svojo privzeto ikono iz Google Places ali Foursquara.

S klikom na ikono lokacije lahko uporabnik pridobi dodatne informacije o izbrani lokaciji (Slika 6.11).

Aplikacija vsakih 50 metrov sporoči svojo lokacijo na strežnik in v primeru spremembe poti (npr. zaradi poročane prometne nesreče v bližini trenutne lokacije) sprejme novo pot v obliki potisnega obvestila. S spremembo lokacije se tudi naložijo vse nove uporabniško sporočane notifikacije – če je uporabnik v določenem radiju od poročane lokacije.

Če uporabnik med potovanjem zapre trenutno okno, se status potovanja spremeni v "Paused" in lahko se nadaljuje z izbiro možnosti "Last travels" iz glavnega menija aplikacije. Potovanje je končano, ko uporabnik pride v končno lokacijo

Slika 6.11: Informacije o lokaciji na potovanju

(+/- 50 metrov zaradi napake v lokacijski storitvi). Status potovanja se spremeni v "Finished" in je na voljo za dodaten pregled v opciji "Last travels".

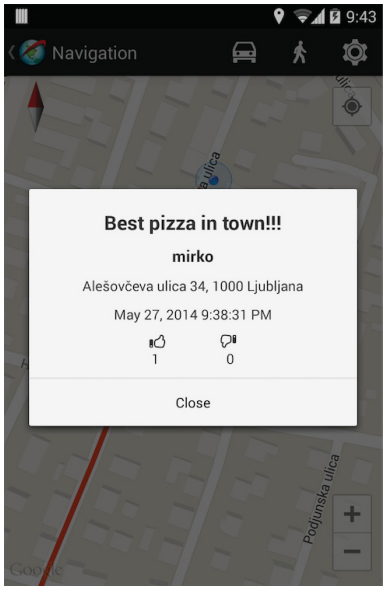
6.2.3.1 Uporabniško poročana obvestila

Uporabnik lahko v času svojega potovanja pošlje obvestilo - krajša informacija, ki lahko drugim uporabnikom izboljša uporabniško izkušnjo.

Novo obvestilo se ustvari z opcijo "Notification" iz "Settings" menija v nadzorni vrstici ali z dolgim pritiskom na izbrano lokacijo na zemljevidu. Zatem aplikacija prikaže okno, v katerega lahko vpišemo besedilo obvestila in izberemo kategorijo. Velikost besedila obvestila je omejena na 200 znakov.

Uporabniško poročana obvestila so na zemljevidu prikazana s privzetim Google Maps API v2 markerjem. S klikom na obvestilo dobimo dodatne informacije o poročanem dogodku (Slika 6.12):

1. **kratek opis**,
2. **uporabniško ime uporabnika, ki je sporočil obvestilo**,
3. **datum obvestila**,



Slika 6.12: Uporabniško poročano obvestilo

4. **ocena** - število uporabnikov, ki so obvestilo ocenili kot uporabno in število uporabnikov, ki so obvestilo ocenili kot neuporabno.

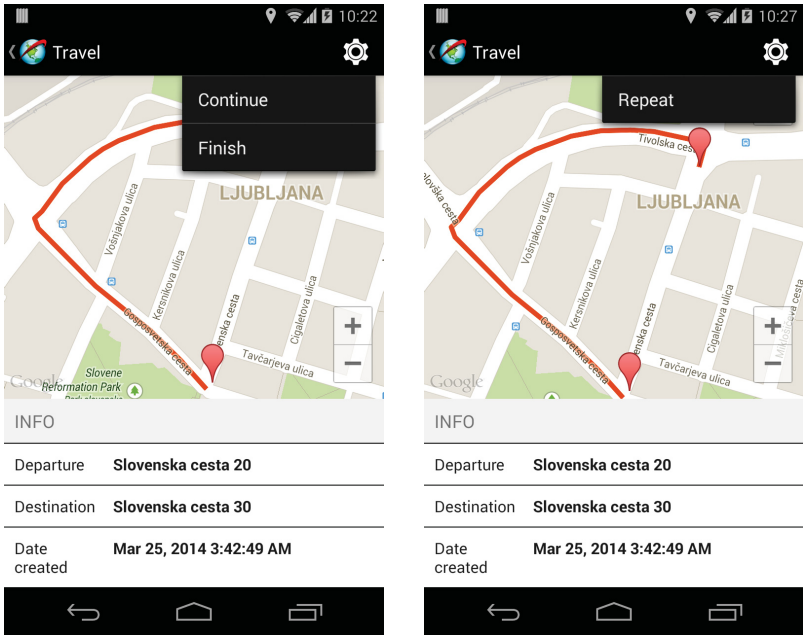
6.2.4 Zgodovina potovanj

Zgodovino naših potovanj lahko preverimo z opcijo "Last travels". Potovanja so razdeljena v 2 kategoriji:

1. **nedokončana potovanja**,
2. **končana potovanja**.

S klikom na poljubno potovanje se odpre novo okno s podrobnostmi potovanja (Slika 6.13a):

- **zemljevid s predlagano potjo**,
- **naslov začetne lokacije**,
- **naslov ciljne lokacije**,



(a) Podrobnosti nedokončanega potovanja

(b) Podrobnosti končanega potovanja

Slika 6.13: Podrobnosti o potovanju

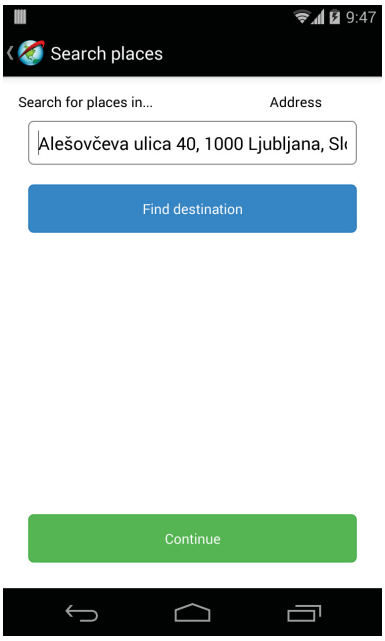
- **datum potovanja**.

S klikom na "Settings" ikono v nadzorni vrstici sta nam omogočeni naslednji akciji (Slika 6.13b):

1. Če je potovanje nedokončano, potem ga lahko nadaljujemo ali končamo,
2. Če je potovanje končano, ga lahko ponovimo.

6.2.5 Iskanje lokacij in dogodkov

Aplikacija omogoča iskanje zanimivih lokacij in dogodkov, ki so v uporabniški bližini in so pridobljeni s storitvijo Google Places API (opcija "Search places") in s storitvijo Foursquare Venues API (opcija "Search events"). Omogočeno je iskanje po trenutni uporabniški lokaciji, po poljubnem naslovu ali po geografskih koordinatah (Slika 6.14a).



(a) Iskanje zanimivih mest po trenutnem naslovu

(b) Zanimive lokacije pridobljene z Google Places API, sortirane po razdalji od trenutne lokacije

Slika 6.14: Iskanje lokacij

Ko potrdimo lokacijo, s klikom na gumb "Continue" pridobimo seznam vseh zanimivih lokacij (Slika 6.14b), sortiranih po razdalji (najbližje lokacije so pri vrhu seznama).

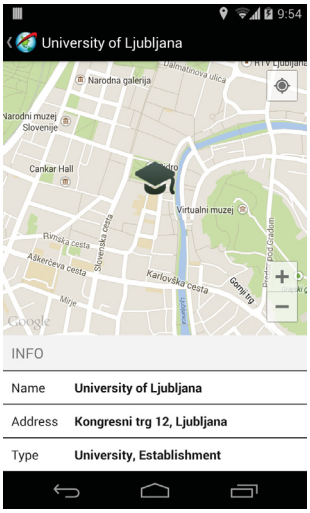
S klikom na katerokoli lokacijo pridobimo dodatne informacije o izbrani lokaciji – lokacijo na zemljevidu, ime lokacije, naslov in tip (Slika 6.15).

6.2.6 Nadzorna plošča

Nadzorna plošča je dostopna skozi "Settings" opcijo nadzorne vrstice glavnega menija, in ponuja 2 možnosti (Slika 6.16a):

1. pregled in spremembo uporabniškega profila,

2. odjavo iz aplikacije.



Slika 6.15: Informacije o lokaciji - Univerza v Ljubljani

6.2.6.1 Uporabniški profil

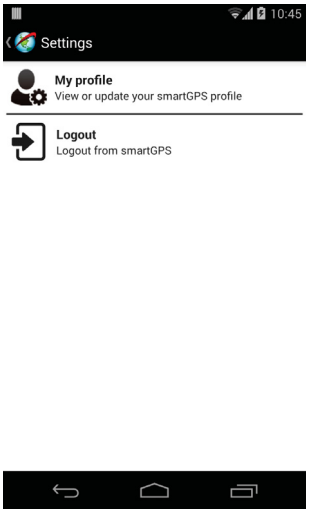
S klikom na opcijo "My profile" lahko pregledamo in spremenimo uporabniški profil. Uporabniški podatki so grupirani v 3 ločene zavihke:

1. **Basic** – uporabniško ime, ime, priimek, spol, datum rojstva (Slika 6.16b),
2. **Contact** – elektronska pošta, telefon,
3. **Address** –naslov, poštna številka, država.

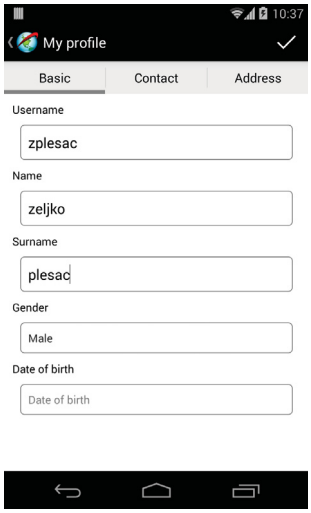
Uporabnik lahko vse spremembe prenese na strežnik s klikom na kljukico v nadzorni vrstici.

6.2.6.2 Odjava iz aplikacije

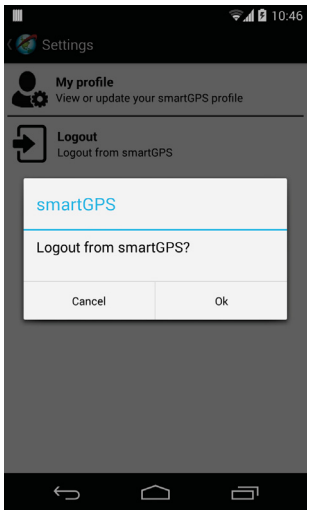
Uporabnik se lahko odjavi iz aplikacije z možnostjo "Logout" v nadzorni plošči. Zatem se pojavi pogovorno okno z vprašanjem "Logout from smartGPS?" (Slika 6.16c). S klikom na "OK" se uporabnik odjavi iz aplikacije in se prekine uporabniška seja.



(a) Nadzorna plošča



(b) Uporabniški profil



(c) Odjava iz aplikacije

Slika 6.16: Opcije nadzorne plošče

Poglavje 7

Evaluacija

7.1 Rezultati testiranja

Algoritme za klasifikacijo uporabnikov smo testirali na testni množici 200 uporabnikov, za katere poznamo uporabniško kategorijo. Vsi uporabniki so klasificirani v 3 uporabniške kategorije:

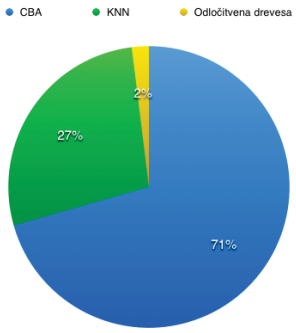
- Traveller - 135 uporabnikov,
- Urban - 49 uporabnikov,
- Sportsman - 15 uporabnikov.

Za primerjavo uspešnosti smo definirali tudi en poenostavljen model - vsi uporabniki so klasificirani v kategorijo Traveller.

Kot je razvidno iz Tabele 7.1, CBA algoritem doseže najboljše rezultate in je tudi za 30.8% boljši od poenostavljenega modela. KNN in odločitvena drevesa tudi dosežejo dobre rezultate, ampak KNN ima skoraj 20% večjo natančnost od odločitvenih dreves. Odločitvena drevesa v primeru več uporabniških primerov dosežejo veliko slabše rezultate, ampak če zmanjšamo število kategorij na dve različni kategoriji in problem postane binaren, potem se občutno zviša natančnost (tudi za 20%), medtem ko CBA in KNN algoritmi dosežejo podobne rezultate. Z večjim številom kategorij(štiri ali pet), spet pridemo do podobnih rezultatov. SVM algoritem se pri privzetih nastavitvah (Laplacianovo jedro) izkaže najslabše. S

Algoritem	**Natančnost**
CBA	67.8%
KNN	58.79%
Odločitvena drevesa	38.1 %
SVM	32.1 %
Poenostavljen model	37%

Tabela 7.1: Primerjava algoritmov in trivialnega modela



Slika 7.1: Delež največkrat uporabljenih algoritmov za končno klasifikacijo

spremembo tipa jedra na linearno ali na Gaussovo jedro pa algoritem pri binarnem klasifikacijskem problemu doseže boljše rezultate.

Pomemben podatek je tudi hitrost klasifikacije in izkaže se, da število uporabniških kategorij vpliva na hitrost izvajanja algoritma. Medtem ko se KNN izkaže kot najhitrejši algoritem, CBA in SVM potrebujeta več časa z večjim številom kategorij. Ker se CBA izkaže kot najbolj natančen algoritem, smo veliko časa porabili za njegovo optimizacijo, pri čemer smo dosegli dobre rezultate.

Pri testiranju smo tudi prišli do zaključka, da naš sistem največkrat uporabi CBA algoritem za končno klasifikacijo uporabnika (Slika 7.1). V primeru bolj homogene učne množice, kjer so si uporabniki bolj podobni (npr. družinske vezi ali prijatelji), se tudi večkrat uporabi KNN algoritem.

Za testiranje gradnje personalizirane poti z A* algoritmom smo definirali več različnih testnih scenarijev, naštetih v Tabeli 7.2. Testiranje smo izvedli na več

Pot	**Naslov začetne točke potovanja**	**Naslov končne točke potovanja**	**Zračna razdalja**
Krajša pot čez center	Slovenska cesta 58, 1000 Ljubljana	Stritarjeva ulica, 1000 Ljubljana	619 m
Daljša pot do centra	Remiza, 1000 Ljubljana	Celovška cesta 4, 1000 Ljubljana	2.23 km
Daljša pot ob robu	Tržaška cesta 145, 1000 Ljubljana	Tržaška cesta 5, Ljubljana	2.42 km

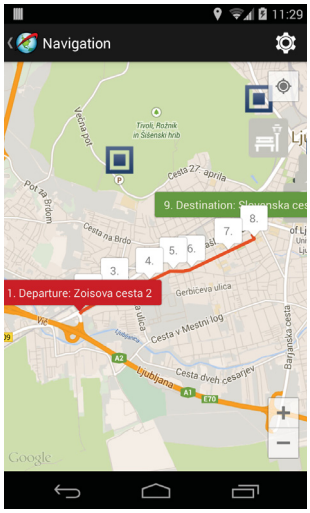
Tabela 7.2: Testni scenariji za A* algoritem

uporabnikih, ki so klasificirani v dve uporabniški skupini: Traveller in Urban. Rezultati testiranja so zelo optimistični in smo izredno zadovoljni z delovanjem algoritma.

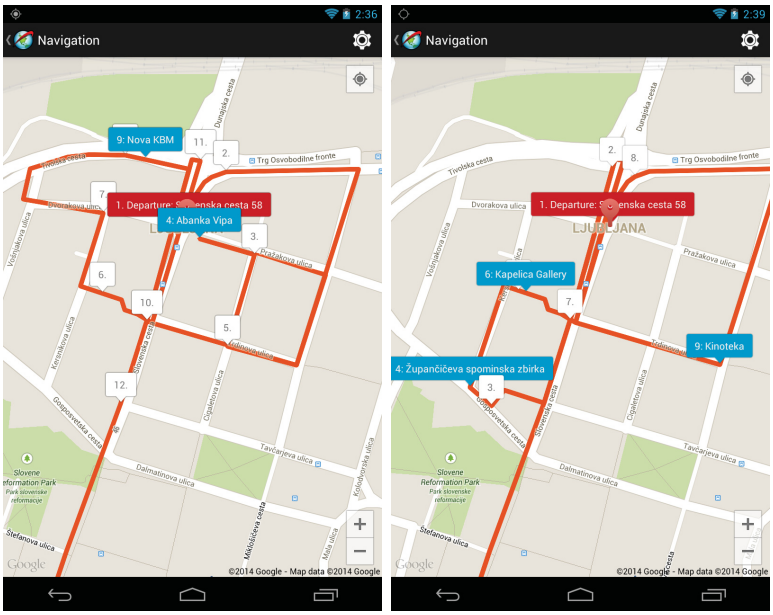
A* algoritem je v 2 od 3 testnih primerov zgradil različno pot za dva testna uporabnika. Z dodatno analizo rezultatov tretjega testnega primera (Slika 7.2) pridemo do zaključka, da nam spletne storitve ne posredujejo nobene informacije, ki ustrezajo profilom testnih uporabnikov (pri testiranju smo radij, v katerem iščemo zanimive informacije za oba testna uporabnika, postavili na 100 metrov). To je tudi glavni vzrok enakih predlaganih poti.

Na Sliki 7.3 in Sliki 7.4 lahko opazimo različne predlagane odseke poti za 2 testna uporabnika. Uporabnika klasificiranega v skupino TRAVELLER, bolj zanimajo informacije, ki mu lahko izboljšajo potovanje, npr. banke in bankomati (Slika 7.3a in Slika 7.4a), medtem ko uporabnika klasificiranega v skupino URBAN, bolj zanima kulturno življenje mesta, npr. lokacije plesnih centrov, opernih hiš, gledališč (Slika 7.3b in Slika 7.4b).

V Tabeli 7.3 smo podali primerjavo časa izvajanja A* algoritma za vsako uporabniško skupino. Kot je razvidno iz rezultatov, na čas izvajanja algoritma vplivajo 3 podatki: razdalja med začetno in končno točko, lokacija potovanja ter število uporabniško zanimivih lokacij in dogodkov. Razdalja je najpomembnejši faktor, ker z večjo razdaljo obstaja več različnih kombinacij predlaganih poti in algoritem potrebuje več časa, da poišče najbolj optimalno pot. Lokacija samega

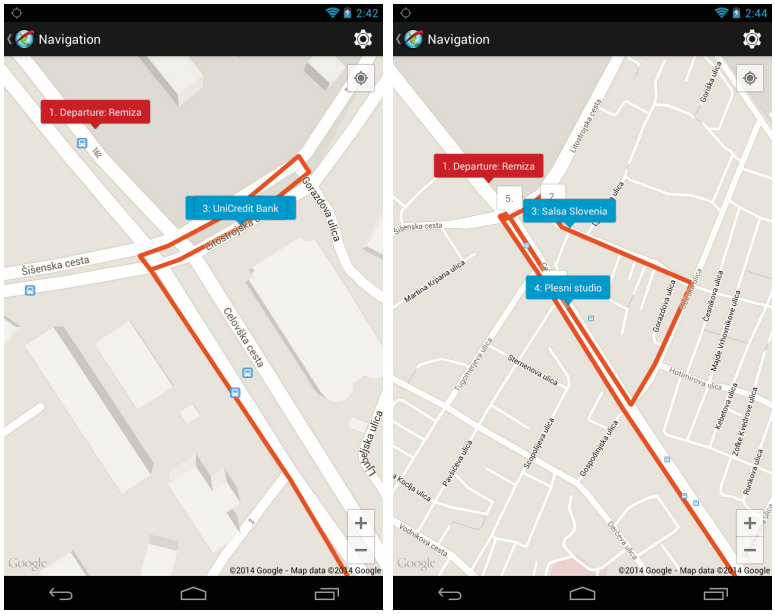


Slika 7.2: Tretji testni primer. Predlagana pot je enaka za oba testna uporabnika



(a) TRAVELLER (b) URBAN

Slika 7.3: Prvi testni primer - primerjava predlagane poti



(a) TRAVELLER (b) URBAN

Slika 7.4: Drugi testni primer - primerjava predlagane poti

potovanja je tudi precej pomembna, ker v centru mesta obstaja več majhnih cest in poti, medtem ko je ob robu mesta večje število dolgih cest in prometnih odsekov. Ta podatek direktno vpliva na število različnih kombinacij možnih poti - v centru mesta je večje število možnih kombinacij poti, medtem ko je ob robu mesta to število manjše. Število uporabniško zanimivih lokacij in dogodkov pa ima proporcionalen vpliv na čas izvajanja algoritma - vsaka lokacija in dogodek se povežeta z že zgrajeno potjo z novim izvajanjem A* algoritma, ki poišče najbolj optimalno pot med predlagano lokacijo in najbližjim odsekom že zgrajene poti.

Iz Tabele 7.3 lahko tudi analiziramo število uporabniško zanimivih lokacij in dogodkov. Medtem ko v tretjem testnem primeru za oba testna uporabnika ni nobenih zanimivih informacij, v prvem in drugem primeru s spletnih storitev pridobimo nekatere zanimive lokacije. Vzrok majhnega števila informacij je v zastavljenem radiju, ki znaša 100 metrov (v primeru večjega radija raste tudi število pridobljenih lokacij). Opazimo lahko tudi, da ima uporabniška skupina URBAN večje število lokacij od TRAVELLER (3:2). Razlaga je v tem, da ima skupina URBAN večje število preslikanih kategorij iz Foursquare Venues in Google Places

Pot	**Kategorija**	**Čas**	**Število zanimivih lokacij in dogodkov**
Krajša pot čez center	TRAVELLER	4min 59 s	2
Daljša pot do centra	TRAVELLER	4min 21 s	2
Daljša pot ob robu	TRAVELLER	13min 11 s	0
Krajša pot čez center	URBAN	5min 23 s	3
Daljša pot do centra	URBAN	5min 48 s	2
Daljša pot ob robu	URBAN	13min 21 s	0

Tabela 7.3: Rezultati testiranja - hitrost algoritma in število uporabniško zanimivih lokacij

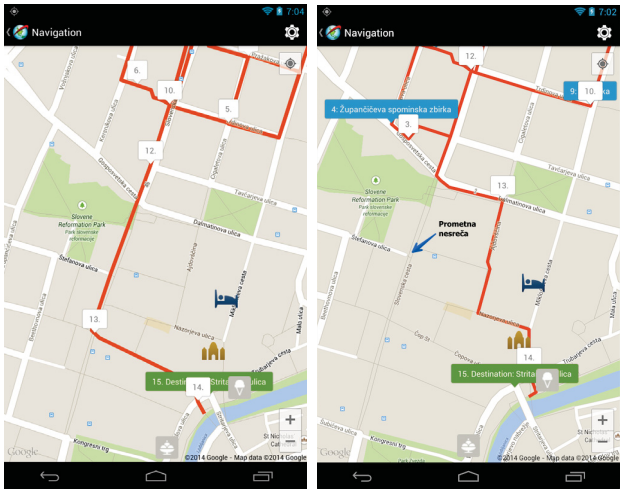
API kot skupina TRAVELLER (13:7).

Na Sliki 7.5 lahko na prvem testnem primeru primerjamo predlagano pot, ki jo A* algoritem zgradi v začetku potovanja (Slika 7.5a), in spremembo predlagane poti po zaznani prometni nesreči v času potovanja (Slika 7.5b). Spletna storitev je avtomatsko s spremembo uporabniške lokacije zaznala nevarnost na odseku poti, ki je še vedno pred uporabnikom. Lokacijo prometne nesreče je algoritem obtežil z dodatno težo in znova poiskal najbolj optimalno pot, ki se v popolnosti izogne lokalne okolice prometne nesreče. Mobilna aplikacija avtomatsko osveži predlagano pot po sprejetju potisnega obvestila in sporoči uporabniku, kaj je razlog spremembe poti.

7.2 Primerjava s konkurenčnimi produkti in SWOT analiza

Kot konkurenčne produkte smo definirali 3 različne aplikacije: (1) WAZE, (2) iGO Primo, (3) Google Maps Navigation. Naš produkt smo primerjali s konkurenco v več kategorijah:

- cena - ali je produkt plačljiv ali brezplačen,



(a) Pred nesrečo (b) Po nesreči

Slika 7.5: Primerjava predlagane poti v primeru poročane prometne nesreče

- predpogoji za uporabo - mobilna naprava, verzija Android OS, uporabniški račun,
- gradnja poti - produkt vedno predlaga isto pot ali je postopek gradnje poti personaliziran po več kriterijih (stanje v prometu, vremenska napoved, dan v tednu),
- nadzor potovanja - ali produkt nadzoruje potovanje uporabnika in mu v primeru morebitne nevarnosti predlaga drugo pot,
- uporabniško dodane informacije - ali produkt omogoča svojim uporabnikom posredovanje informacij, ki lahko drugim uporabnikom izboljšajo uporabniško izkušnjo,
- hitrost gradnje poti - hitrost algoritma za gradnjo poti,
- pokritost z zemljevidi - ali je pokrita celotna Zemlja, večje področje ali samo manjše področje,
- grafični izgled vmesnika - ali je v skladu s smernicami za razvoj Android mobilnih aplikacij.

7.2.1 SWOT analiza

SWOT[1] analiza [47] je ena od najpogostejših analiz v poslovnem svetu, ki nam omogoča podroben vpogled v delovanje našega sistema. To omogoča na način, da analiziramo sistem in določimo 4 kategorije značilnosti:

1. Prednosti - značilnosti sistema, ki nam omogočajo prednost pred konkurenčnimi produkti.

2. Slabosti - značilnosti sistema, ki omogočajo prednost konkurenčnim produktom pred našim produktom.

3. Priložnosti - značilnosti sistema, ki predstavljajo možnosti za pridobivanje prednosti pred konkurenčnimi produkti.

4. Nevarnosti - značilnosti sistema, ki predstavljajo možnosti za izgubo prednosti pred konkurenčnimi produkti.

V nadaljevanju bomo predstavili SWOT analizo razvitega sistema v primerjavi z konkurenčnimi produkti (po Tabeli 7.4).

7.2.1.1 Prednosti

Prednosti sistema so na več različnih področjih. Največja prednost je personalizacija gradnje poti med začetno in končno točko potovanja. Sistem spremlja in analizira, na kakšen način uporabnik uporablja aplikacijo, kaj so njegovi interesi in hobiji, in mu v skladu s tem spreminja predlagano pot pri potovanju. S pogostejšo uporabo aplikacije, sistem boljše spozna posameznega uporabnika in mu hitrejše in boljše predlaga pot.

Druga prednost je izboljšanje varnosti uporabnika v času potovanja. Ker aplikacija ves čas spremlja uporabniško lokacijo, lahko zazna nevarnost na odseku potovanja, ki je še vedno pred uporabnikom, in mu v skladu s tem predlaga novo pot. Sistem je popolnoma neodvisen od uporabniške interakcije, kar tudi močno vpliva na varnost uporabnika.

[1]SWOT - ang Strengths, Weaknesses, Opportunities, Threats

Kategorija	**WAZE**	**iGO Primo**	**Google Maps Navigation**	**smartGPS**
Cena	brezplačno	plačljivo	brezplačno	brezplačno
Predpogoji za uporabo	pametni telefon, mobilna aplikacija, uporabniški račun	pametni telefon, mobilna aplikacija(na voljo edino za poslovne uporabnike)	pametni telefon, mobilna aplikacija, uporabniški račun	pametni telefon, mobilna aplikacija, spletni brskalnik, uporabniški račun
Gradnja poti	ni personalizacije	ni perosnalizacije	delna personalizacija	popolna personalizacija
Nadzorovanje potovanja	da	da	da	da
Uporabniško dodane informacije	da	da	ne	da
Hitrost gradnje poti	sekunde	sekunde	sekunde	tudi do par minut
Pokritost poti	celoten svet	celoten svet	celoten svet	Ljubljana
Grafični izgled vmesnika	zastarel	zastrel	skladen z Android navodili	skladen z Android navodili

Tabela 7.4: Primerjava razvitega sistema s konkurenčnimi produkti

Tretja prednost je uporaba socialne komponente - sistem omogoča asinhrono komunikacijo med uporabniki in sami uporabniki izboljšajo delovanje sistema s svojimi poročanimi informacijami.

Na koncu je prednost tudi v modernem in sodobnem grafičnem vmesniku ter v ceni.

7.2.1.2 Slabosti

Največja slabost sistema je njegova počasnost in omejenost delovanja na področje Ljubljane. Sistem se trenutno izvaja na omejeni strojni opremi, kar negativno vpliva na hitrost izvajanja algoritmov. Algoritem za generiranje poti tudi izvaja preveč sprehodov skozi množico točk, ki predstavljajo zemljevid, kar onemogoča uporabo OpenMaps zemljevidov skozi REST API. Zaradi tega smo se odločili, da bomo zemljevide shranili v lokalno podatkovno bazo, vendar je zaradi omejene velikosti baze trenutno omogočeno testiranje samo na področju Ljubljane.

Slabost sistema je tudi v tem, da uporabnik še vedno potrebuje spletni portal za uvoz podatkov iz Facebook in Foursquare uporabniških računov. Facebook ima zelo striktno varnostno politiko izdajanja informacij o svojih uporabnikih zunanjim storitvam in je kot predpogoj za popolno integracijo z mobilnimi aplikacijami potrebno posredovati izvorno kodo aplikacije na pregled in oceno, ali se aplikacija obnaša v skladu s pravili.

Na koncu je slabost sistema tudi v tem, da je v času potovanja potrebna aktivna povezava z internetom ter aktivne lokacijske storitve, kar tudi močno vpliva na porabo baterije.

7.2.1.3 Priložnosti

Z že razvito arhitekturo sistema lahko dosežemo interakcijo z večjim številom spletnih storitev, ki bi nam še dodatno izboljšale uporabniško izkušnjo in varnost uporabnikov. Predvsem tukaj mislimo na interakcijo z večjim številom socialnih omrežij (kot so Twitter, Google+, LinkedIn), ki bi nam posredovale dodatne informacije o posameznem uporabniku. Na ta način bi tudi izboljšali rezultate algoritmov za klasifikacijo uporabnikov.

Z interakcijo s storitvami, kot so prevoz.org, slovenskezeleznice.com, bicikelj.si, adria.si, bi lahko tudi omogočali uporabo več načinov prevozov in prevoznih sred-

stev.

Arhitektura sistema nam tudi omogoča, da lahko uporabniško posredovane informacije pri potovanju tudi izvozimo skozi spletno storitev v druge aplikacije. Na ta način bi pridobili več uporabnikov sistema, ki bi posredovali še večje število informacij.

7.2.1.4 Nevarnosti

Največja nevarnost našega sistema je v lokalnem shranjevanju občutljivih osebnih uporabniških podatkov. Vsi podatki, ki jih uporabniki uvozijo s Facebook in Foursquare računov se morajo shraniti na previden način v podatkovno bazo. Obstaja tudi problem sinhronizacije lokalnih podatkov z oddaljenimi podatki na socialnih omrežjih - če je uporabnik odstranil kakšen podatek na Facebooku, se mora ta sprememba tudi replicirati v našo lokalno bazo. Potrebno je tudi razviti sistem dovoljenj do podatkov - nekateri podatki so bolj občutljivi in se morajo v skladu s tem tudi dodatno zavarovati.

Nevarnost je tudi v tem, da je sistem preveč odvisen od aktivnosti uporabnikov. Z manjšim številom uporabnikov ali z večjim številom neaktivnih uporabnikov naš sistem hitro postane slabši od konkurence. Zaradi tega je potrebno razviti tudi sistem nagrajevanja aktivnih uporabnikov, ki aplikacijo pogosto uporabljajo in pošiljajo uporabne informacije.

Končno, z rastjo števila uporabnikov in števila potovanj, bomo potrebovali tudi boljšo strojno opremo, kar lahko v končni fazi tudi vpliva na ceno uporabe samega sistema.

Poglavje 8

Sklepne ugotovitve in zaključek

8.1 Težave v razvoju

Pri razvoju smo imeli več težav s strojno opremo in z nepopolnimi podatki. Ko smo začeli z razvojem mobilne aplikacije, smo podpirali vse verzije Androida nad verzijo 2.2 (verzija kode 8). Aplikacija ima aktivno komunikacijo s strežnikom v celotnem času porabe in v času potovanja aplikacija potrebuje še aktivno lokacijsko storitev (GPS/WiFi/GPRS), zaradi česa je potrebno veliko energije. Uporabili pa smo tudi Android Bootstrap knjižnico za grafično oblikovanje aplikacije, ki v času razvoja ni bila dostopna za verzijo Androida pod 4.0. Na koncu vse to vpliva na stabilnost aplikacije in uporabniško izkušnjo, zato smo se odločili, da bo aplikacija primarno optimizirana za JellyBean verzijo Androida. Veliko problemov smo tudi imeli s konfiguracijo strežnika – potrebujemo strežnik z dovolj procesorske moči in z veliko količino delovnega spomina, ker so algoritmi časovno potratni in delajo precej prehodov skozi podatke. Na testnem strežniku je algoritem za generiranje poti, kjer je razdalja med začetno in končno točko večja kot 2 kilometra, potreboval tudi do 15 minut za izračun poti. Slaba konfiguracija tudi vpliva na odzivnost spletne aplikacije na zahtevke z mobilne naprave – za odgovor na zahtevke večkrat potrebujemo tudi več kot 5 sekund, kar vpliva na uporabniško izkušnjo mobilne aplikacije. OpenStreetMap zemljevidom tudi primanjkuje veliko podatkov. Aplikacijo smo testirali na področju Ljubljane, ki je dovolj dobro pokrito, ampak testiranje v kakšni manjši vasici ali na večjem področju je bilo nemogoče,

ker podatki niso popolni. Vse ceste niso kategorizirane, manjkajo tudi podatki o tipu ceste, omejitve hitrosti in še več. To je tudi glavni vzrok za slabo delovanje sistema. Problem je tudi v sami velikosti zemljevidov – področje Ljubljane je shranjeno v relacijsko podatkovno bazo v 4 različne tabele, vse skupaj velikosti čez 15 MB. Če bi zajeli večje področje (npr. Zahodna Evropa), potrebujemo tudi podatkovno bazo z večjim prostorom, kar vpliva tudi na strojno opremo in samo ceno testiranja. Večjo težavo so predstavljale tudi spletne storitve, ker postavljajo omejitve na brezplačen dostop do svojih podatkov. Največji problem smo imeli s Facebook API, ker je v enem zahtevku omogočeno največ 600 informacij na 10 minut, kar je zelo majhna številka, če zajamemo podatke o uporabniških všečkih in všečkih naših prijateljev. Probleme smo imeli tudi s strukturo podatkov, posebej s podatki o razmerah v prometu Prometno-informacijskega centra Slovenije, ker so podatki o čakalnih dobah in dolžinah zastojev v tekstualni obliki.

8.2 Predlagane izboljšave in nadgradnje

Trenutno je sistem še vedno v alfa verziji razvoja in ga bomo izboljšali na več različnih načinov pred samo produkcijo. Prva in najpomembnejša izboljšava bo odprava prijavljenih napak ter izboljšava izgleda grafičnega vmesnika na mobilni in spletni aplikaciji. Mobilno aplikacijo bomo prilagodili tudi za tablične računalnike (trenutno je izgled prilagojen za 5-inčne mobilne naprave), spletno aplikacijo pa bomo prenovili z uporabo responzivnega dizajna. Aplikacije bomo tudi lokalizirali v več različnih jezikov (primarno slovenščina in hrvaščina).

Ena od nadgradenj obstoječih funkcionalnosti bo vpeljava javnih in privatnih obvestil. Uporabniki bodo imeli možnost pošiljanja in sprejemanja obvestil samo od določenih uporabnikov. Na ta način bomo zagotovili dodatno varnost uporabnikov in izboljšali uporabniško izkušnjo. Vpeljali bomo tudi možnost klasifikacije uporabnika v več uporabniških skupin, kjer bo število skupin določil sam uporabnik. Na ta način bodo uporabniki imeli na voljo več lokacij in informacij v času svojega potovanja, predlagana pot pa bo bolj prilagojena zahtevam samega uporabnika. Negativna stran pa je v tem, da bo algoritem za gradnjo poti potreboval več časa, lahko pa tvegamo tudi slabšo uporabniško izkušnjo, ker bodo uporabniki preplavljeni z velikim številom informacij. Izboljšali bomo tudi sam algoritem za

klasifikacijo uporabnika z uporabo več informacij o samem uporabniku. Dodatne informacije bomo pridobili s podporo za večje število socialnih omrežij, kot so Twitter, Google+ in LinkedIn. Vsako socialno omrežje opisuje uporabnika na svoj način – Facebook, Twitter in Google+ nam povedo več o zasebnem življenju uporabnika, LinkedIn pove o profesionalnem življenju (službi) in izkušnjah, Foursquare o potovanjih in interesih... Z uporabo več virov informacij bomo boljše spoznali naše uporabnike in jim na ta način tudi izboljšali uporabniško izkušnjo.
Algoritem za gradnjo poti bomo izboljšali tudi na način, da bomo vpeljali še dodatne vire informacij, ki lahko vplivajo na potovanje. Omogočili bomo tudi integracijo s ponudniki različnih tipov prevoza, kot so BicikeLJ, Slovenske železnice, prevoz.org... Na ta način bomo omogočili tudi potovanja na daljše relacije. Vse predlagane izboljšave in nadgradnje bodo zahtevale tudi več procesorske moči in delovnega spomina, zaradi česar bomo spletno aplikacijo migrirali v oblak.

8.3 Zaključek

V magistrski nalogi smo razvili funkcionalen prototip GPS aplikacije, ki z močno uporabo socialne komponente izboljša uporabniško izkušnjo obstoječih GPS aplikacij. Prototip je zasnovan na sistemu, ki izboljša obstoječe GPS aplikacije na treh različnih področjih - (1) izboljšanje uporabniške izkušnje, (2) izboljšanje varnosti in (3) večja uporaba družabne komponente.

Prototip sestavljata 2 komponenti - spletna in mobilna aplikacija, ki komunicirata po strežnik-odjemalec arhitekturnem vzorcu. Spletna aplikacija omogoča uvoz podatkov z različnih socialnih omrežij, na podlagi katerih lahko klasificiramo uporabnika v že obstoječe uporabniške skupine. Za postopek klasifikacije smo uporabili več metod strojnega učenja, ki smo jih posebej optimizirali za uporabo v takšnem arhitekturnem vzorcu. Vsako metodo testiramo na testni množici uporabnikov in za končno klasifikacijo uporabnika uporabimo metodo, ki doseže največjo natančnost klasifikacije. Spletna aplikacija tudi omogoča gradnjo personalizirane poti za vsakega uporabnika posebej na podlagi uporabniške kategorije z uporabo A* preiskovalnega algoritma. Na postopek gradnje poti vplivata dve kategoriji informacij, ki se uvozita iz različnih spletnih virov: (1) dogodki, ki predstavljajo nevarnost na poti (poskušamo se jim izogniti), ter (2) dogodki in mesta, ki so lahko zanimivi

uporabniku (poskušamo jih obiskati).

Mobilna aplikacija je prilagojena za Android pametne telefone in omogoča navigacijo po poti na način, da komunicira s spletno aplikacijo skozi REST API. V spletno aplikacijo sporoča trenutno uporabniško lokacijo, s čimer spletna aplikacija lahko nadzira potovanje uporabnika in lahko spremeni predlagano pot v primeru nevarnosti. Spletna aplikacija pošilja v mobilno aplikacijo tudi vse informacije, ki so jih sporočali drugi uporabniki in so v uporabniško zastavljenem radiju od trenutne lokacije. Mobilna aplikacija tudi omogoča preverjanje vseh mest in dogodkov, ki so v bližini uporabnika in odgovarjajo njegovi uporabniški skupini.

S testiranjem prototipa smo dokazali, da so izpolnjeni vsi zastavljeni mejniki in cilji. Hkrati smo tudi ugotovili, da obstoječi sistem ni idealen in da obstaja veliko možnosti za izboljšavo.

Izvorna koda rešitve je dostopna na [46].

Pri razvoju predlaganega sistema in prototipa smo uporabili znanje, pridobljeno na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, in verjamemo, da je razviti sistem korak naprej na področju GPS navigacije.

Literatura

[1] W. Min, L. Wynter, Y.Amemiya, “Road traffic prediction with spatio-temporal correlations”, IBM Research Report, 2007.

[2] W.Shen, L. Wynter, "Real-Time Traffic Prediction Using GPS Data with Low Sampling Rates: A Hybrid Approach", IBM Research Report, 2011.

[3] J. Lindqvist, J. Hong, “Undistracted Driving: A Mobile Phone that Doesn’t Distract”, Proceedings of of 12th Workshop on Mobile Computing Systems and Applications, Phoenix, Arizona, USA, 2011.

[4] I. Lequerica, M.G. Longaron, P.M. Ruiz, “Drive and share: efficient provisioning of social networks in vehicular scenarios”, IEEE Communications Magazine, vol. 48, pp. 90–97, 2010.

[5] G. Lawton "Machine to Machine Technology Gears up for Growth", IEEE Computer, vol. 37, no. 9, pp.12 -15, 2004.

[6] W. Sha, K. Daehan, N. Badri , I. Liviu. Šocial Vehicle Navigation: Integrating Shared Driving Experience into Vehicle Navigation", Proceedings of of 14th Workshop on Mobile Computing Systems and Applications, Jekyll Island, Georgia, USA, 2013.

[7] K. Perera, D. Dileeka. "An intelligent driver guidance tool using location based services"In Spatial Data Mining and Geographical Knowledge Services (ICSDM), 2011 IEEE International Conference, pp. 246-251. IEEE, 2011.

[8] M. Li, Z. Du, “Dynamic Social Networking Supporting Location - Based Services”, Proceedings of International Conference on Intelligent Human - Machine Systems and Cybernetics, vol. 1, pp. 149–152, 2009.

[9] L. Chang, W. Chen, "Data mining of tree-based models to analyze freeway accident frequency", Journal of Safety Research, vol. 36, pp. 365-375, 2005.

[10] Uradna spletna stran podjetja Waze mobile, dostopno na: http://www.waze.com. Datum dostopa: 24.04.2014.

[11] Uradna spletna stran podjetja NNG Software Developing and Commercial Llc, iGo primo, dostopno na: http://www.nng.com/igo-primo. Datum dostopa: 24.04.2014.

[12] Uradna spletna stran aplikacije Google Maps Navigation, dostopno na: http://www.google.si/mobile/maps/. Datum dostopa: 24.04.2014.

[13] Uradna spletna stran Somme spletnega servisa, dostopno na: https://somee.com/. Datum dostopa: 24.04.2014.

[14] Uradna spletna stran Microsoft ASP .NET MVC programskega okolja, dostopno na: http://www.asp.net/. Datum dostopa: 24.04.2014.

[15] Uradna spletna stran Microsoft SQL Server programskega okolja, dostopno na: www.microsoft.com/en-us/sqlserver/default.aspx. Datum dostopa: 24.04.2014.

[16] Uradna spletna stran Android SDK-ja, dostopno na: http://developer.android.com/sdk/index.html. Datum dostopa: 24.04.2014.

[17] Tržni delež mobilnih operacijskih sistemov, dostopno na: http://www.latinopost.com/articles/3205/20140116/ios-vs-android-market-share-europe-top-5-markets.htm. Datum dostopa: 24.04.2014.

[18] Uradna spletna stran Apache 2.0. licence, dostopno na: http://www.apache.org/licenses/LICENSE-2.0.html. Datum dostopa: 24.04.2014.

[19] Uradna spletna stran Google Geocoding API knjižnice, dostopno na: https://developers.google.com/maps/documentation/geocoding/. Datum dostopa: 24.04.2014.

[20] Uradna spletna stran Facebook socialnega omrežja, dostopno na: https://facebook.com/.Datum dostopa: 24.04.2014.

[21] Uradna spletna stran Foursquare socialnega omrežja, dostopno na: https://foursquare.com/. Datum dostopa: 24.04.2014.

[22] Uradna spletna stran Open WeatherMap spletne storitve, dostopno na: http://openweathermap.org/. Datum dostopa: 24.04.2014

[23] Uradna spletna stran JSON.NET knjižnice, dostopno na: http://james.newtonking.com/json . Datum dostopa: 24.04.2014.

[24] Uradna spletna stran Accord .NET knjižnice, dostopno na: http://accord-framework.net/ . Datum dostopa: 24.04.2014.

[25] Uradna spletna stran Prometno-informacijskega centra, dostopno na: http://www.promet.si/portal/sl/razmere.aspx. Datum dostopa: 24.04.2014.

[26] Uradna spletna stran OpenMaps projekta, dostopno na: http://openstreetmap.org/ . Datum dostopa: 24.04.2014.

[27] Uradna spletna strant SQLite relacijske podatkovne baze, dostopno na: http://www.sqlite.org/. Datum dostopa: 24.04.2014.

[28] Uradna spletna stran JOSM urejevalnika, dostopno na: https://josm.openstreetmap.de/ . Datum dostopa: 24.04.2014.

[29] Uradna spletna stran Merkaator urejevalnika, dostopno na: http://merkaartor.be . Datum dostopa: 24.04.2014.

[30] Uradna spletna stran ActiveAndroid knjižnice, dostopno na: http://www.activeandroid.com/ . Datum dostopa: 24.04.2014.

[31] Uradna spletna stran GSON knjižnice, dostopno na: https://code.google.com/p/google-gson/ . Datum dostopa: 24.04.2014.

[32] Uradna spletna stran Google Cloud Messaging spletne storitve, dostopno na: http://developer.android.com/google/gcm/index.html. Datum dostopa: 24.04.2014.

[33] Spletni naslov za pošiljanje potisnih obvestil, dostopno na: https://android.googleapi.com /gcm/send. Datum dostopa: 24.04.2014.

[34] Uradna spletna stran Google Developers Console, dostopno na: https://code.google.com/apis/console . Datum dostopa: 24.04.2014.

[35] Uradna spletna stran Android Boostrap knjižnice, dostopno na: http://www.androidbootstrap.com/ . Datum dostopa: 24.04.2014.

[36] Uradna spletna stran Android Maps Utils knjižnice, dostopno na: http://googlemaps.github.io/android-maps-utils/. Datum dostopa: 24.04.2014.

[37] Uradna spletna stran Android Asynchronous Http Client knjižnice, dostopno na: http://loopj.com/android-async-http/. Datum dostopa: 24.04.2014.

[38] Uradna spletna stran Android Universal Image Loader knjižnice, dostopno na: https://github.com/nostra13/Android-Universal-Image-Loader. Datum dostopa: 24.04.2014.

[39] T. Cover, P. Hart, “Nearest neighbor pattern classification", Information Theory, IEEE Transactions, vol 13, pp. 21-27, 1967.

[40] G-R. Xue, C. Lin, Q. Yang, W. Xi, H-J Zeng, Y. Yu, Z. Chen, ”Scalable collaborative filtering using cluster-based smoothing”, Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, pp 114-121, 2005.

[41] Y. Hu , T-H Ku, R-H Jan, K Wang, Y-C Tseng, S-F Yang, “Decision tree-based learning to predict patient controlled analgesia consumption and readjustment", BMC Medical Informatics and Decision Making, 2012

[42] V. Vapnik, C. Cortes, ”Support vector networks”, Machine learning, vol 20, pp 273-297, 1995.

[43] B. Liu , W. Hsu , Y. Ma, ”Integrating Classification and Association Rule Mining”, In proceeding of KDD, 1998.

[44] P.E. Hart, N.J. Nilsson, B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths", Systems Science and Cybernetics, IEEE Transactions , vol 4, pp. 100-107, 1968.

[45] Github repozitorij sistema smartGPS, dostopno na: https://github.com/wartual/. Datum dostopa: 24.04.2014.

[46] Github repozitorij mobilne aplikacije smartGPS, dostopno na: https://github.com/wartual/smartGPS-android. Datum dostopa: 24.04.2014.

[47] T. Hill, R. Westbrook, ”SWOT analysis: It’s time for a product recall”, Long Range Planning, volume 30, issue 1, pp 46–52, 1997.