Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Krivec

Mobilna aplikacija za podporo kolesarjem z zalednim sistemom v oblaku

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Damjan Vavpotič

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1] V dogovorju z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00432/2013
Datum: 09.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **LUKA KRIVEC**

Naslov: **MOBILNA APLIKACIJA ZA PODPORO KOLESARJEM Z ZALEDNIM SISTEMOV V OBLAKU**

MOBILE APPLICATION FOR CYCLISTS WITH BACK-END SYSTEMS IN THE CLOUD

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V okviru diplomske naloge izdelajte prototip informacijske rešitve, ki bo podprla prikaz zanimivih točk za kolesarje na zemljevidu glede na njihovo trenutno lokacijo in beleženje prevožene poti. Rešitev naj bo sestavljena iz mobilne aplikacije in zalednega sistema, ki bo deloval na izbrani javni oblačni platformi kot npr. Google App Engine. Rešitev naj vključuje tudi povezavo na izbrane javno dostopne zunanje storitve, ki naj jih izkorišča za pridobivanje potrebnih podatkov (npr. za prikaz zanimivih točk za kolesarje na zemljevidu).

Mentor:

doc. dr. Damjan Vavpotič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Luka Krivec, z vpisno številko **63100043**, sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za podporo kolesarjem z zalednim sistemom v oblaku

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 26. marca 2013 Podpis avtorja:

Zahvaljujem se svoji družini, ki mi je vsa leta študija stala ob strani. Hvala tudi sošolcem in prijateljem, ki so mi pomagali pri študiju. Za pomoč in nasvete pri pisanju diplomskega dela se zahvaljujem mentorju doc. dr. Damjanu Vavpotiču.

Kazalo

Povzetek

V diplomskem delu je bila implementirana mobilna aplikacija za platformo `Android`, ki ima zaledni sistem na platformi `Google App Engine`. Namen aplikacije je kolesarjem ponuditi boljši pregled nad opravljenimi kolesarskimi izleti in jim pomagati pri izbiri novih zanimivih izletov. Osnovna funkcionalnost aplikacije je prikaz osnovnih podatkov o vožnji, kot sta hitrost in prevožena razdalja. Aplikacija omogoča tudi prikaz nekaterih drugih za kolesarje zanimivih podatkov na zemljevidu. Tako si lahko ogledamo kolesarske informacijske točke in kolesarske servise v bližini. Na zemljevidu pa lahko poiščemo tudi zaklade, ki so jih označili uporabniki spletne skupnosti `Geocaching`. V prvem delu diplomskega dela smo predstavili tehnologije, s pomočjo katerih smo razvili aplikacijo, v osrednjem delu pa smo pojasnili, kako smo se lotili njenega načrtovanja in implementacije, kakšen je izgled končnega izdelka ter kakšne so možnosti za nadaljnji razvoj aplikacije.

Ključne besede: Android, Google App Engine, kolesarjenje, mobilne aplikacije, računalništvo v oblaku.

Abstract

The thesis presents the implementation of a mobile application for the `Android` platform, which has a back-end system on the `Google App Engine` platform. The purpose of the application is to provide cyclists with a better review of their previous cycling trips and help them with selecting a new one. The basic functionality of the application is the ability to display the basic information about the cycling trip, such as the speed and travelled distance. Moreover, the application displays other interesting information for cyclists, who can search for cycling info points and cycling services located nearby. They can search for the treasures that have been marked by other users of the `Geocaching` web community. The first part of the thesis presents the technologies we used for the development of our application. The main part explains how we planned and implemented the application, discusses the appearance of the final product, and explores the options for further development of the application.

Key words: Android, Google App Engine, cycling, mobile applications, cloud computing.

Poglavje 1

Uvod

Razširjenost pametnih mobilnih telefonov je v zadnjih letih močno narasla. Prav tako se hitro povečujejo tudi zmogljivosti teh naprav. Razvoj aplikacij za te naprave je zato postal zanimiv za številne razvijalce. Zelo aktualna tema s področja računalništva je tudi računalništvo v oblaku. Uporaba računalniškega oblaka poveča fleksibilnost aplikacije in zmanjša stroške vzdrževanja. Prav zato številne aplikacije za mobilne naprave uporabljajo računalniški oblak. S tem pridobijo na fleksibilnosti, saj so podatki shranjeni na lokaciji, do katere lahko dostopajo aplikacije, ki so razvite za različne mobilne operacijske sisteme.

Zaradi prej naštetih prednosti smo se odločili, da tudi sami razvijemo mobilno aplikacijo, ki bo uporabljala računalniški oblak. Izbrali smo platformo `Google App Engine`. Tip storitve, ki jo omogoča, je platforma kot storitev. Omogoča nam razvoj aplikacij na tej platformi in tudi brezplačno gostovanje aplikacije z določeno domeno. Prav tako imamo več možnosti, kje in v kakšni obliki bomo shranjevali podatke. Mi smo si izbrali relacijsko podatkovno bazo `Google Cloud SQL`.

Med številnimi uporabniki pametnih mobilnih telefonov so tudi kolesarji. Ti jih med drugim uporabljajo tudi za analizo športnih aktivnosti. Prav tako marsikoga zanima, kje so zanimive lokacije, kamor bi se lahko naslednjič odpravili. Aplikacija, ki smo jo razvili, omogoča beleženje in pregled športnih

aktivnosti ter prikaz kolesarskih informacijskih točk, kolesarskih servisov in zakladov `Geocaching` na zemljevidu.

V drugem poglavju smo predstavili platformo Android, ki smo jo uporabili za razvoj mobilne aplikacije. Opisane so predvsem komponente, ki smo jih uporabili. V tretjem poglavju so predstavljene uporabljene tehnologije in storitve. V četrtem poglavju je opisan razvoj mobilne aplikacije. Predstavljeno je, kako je sestavljen informacijski sistem, podani pa so tudi primeri uporabe aplikacije. Ločeno sta predstavljena razvoja mobilne aplikacije in spletnega strežnika. Pri razvoju mobilne aplikacije smo predstavili aktivnosti aplikacije in uporabniški vmesnik, pri spletnem strežniku pa povezavo strežnika z mobilno aplikacijo. V zadnjem poglavju so podane še sklepne ugotovitve diplomskega dela.

Poglavje 2

Platforma Android

2.1 Operacijski sistem Android

`Android` je odprtokodna platforma, ustvarjena za mobilne naprave, zaradi česar jo lahko proizvajalci mobilnih naprav in drugi razvijalci prilagodijo po svojih željah. Velika prednost te platforme je, da ločuje strojno opremo od programske. Prav zato lahko veliko število različnih mobilnih naprav poganja isto aplikacijo. Omenjena platforma temelji na operacijskem sistemu `Linux`. Ker je ta sistem že dolgo časa prisoten in preizkušen v številnih okoljih, tudi platforma Android uporablja veliko njegovih funkcionalnosti. Za njen razvoj skrbi ameriško podjetje `Google`. [1]

Veliko število razvijalcev programske opreme razvija aplikacije prav za operacijski sistem `Android`. S tem mobilnim napravam dodajajo nove funkcionalnosti. Aplikacije so dostopne v `Googlovi` spletni trgovini `Google Play`. Veliko aplikacij je brezplačnih, nekatere pa so plačljive.

Android je najbolj razširjen operacijski sistem za mobilne naprave. Po podatkih iz meseca marca 2013 ima 64-odstotni tržni delež. Seveda obstaja več različic operacijskega sistema. Prva različica ima oznako 1.5, zadnja različica pa 4.2. Največ mobilnih naprav trenutno uporablja različice 2.3.3, 4.0 in 4.1. [6]

2.2 Android SDK

Android SDK (angl. software development kit - paket za razvoj programske opreme) prevede programsko kodo, napisano v programskem jeziku Java, v arhivski paket Android. Ta paket je datoteka s končnico apk. Vsa koda aplikacije je tako združena v eno datoteko, prek katere lahko potem naprave z operacijskim sistemom Android namestijo aplikacijo. [7]

Paket vsebuje razhroščevalnik programske kode, knjižnice za razvoj aplikacij za Android, emulator, dokumentacijo in vzorčne primere uporabe orodij. Na voljo je za operacijske sisteme Linux, Mac OS X in Windows. Uradno podprto integrirano razvojno okolje za uporabo orodja Android za razvoj programske opreme je Eclipse. Uporablja se skupaj z vtičnikom ADT (angl. Android Development Tools - razvojna orodja Android). [8]

2.3 Komponente aplikacije za Android

Vsaka aplikacija za Android je sestavljena iz več osnovnih komponent. Komponenta predstavlja točko, prek katere lahko sistem vstopi v aplikacijo. Skupek vseh komponent določa obnašanje aplikacije. Obstajajo štirje glavni tipi komponent aplikacije, ki so predstavljeni v nadaljevanju.

2.3.1 Aktivnosti

Aktivnost (angl. Activity) predstavlja en zaslon aplikacije z uporabniškim vmesnikom. Vse aktivnosti tvorijo povezan uporabniški vmesnik kljub temu, pa je vsaka aktivnost neodvisna od drugih. Prav zato lahko vsaka aplikacija zažene katero koli od teh aktivnosti. Aktivnost implementiramo prek javanskega razreda Activity. [8]

Vsaka aktivnost dobi okno za prikaz uporabniškega vmesnika. Običajno aktivnost zapolni celo okno, lahko pa ima tudi manjše okno, ki lebdi nad drugimi okni. Aplikacija tipično vsebuje glavno aktivnost, ki je prikazana uporabniku ob zagonu aplikacije. Vsaka aktivnost lahko zažene drugo ak-

tivnost, ki izvede določeno akcijo. Vsakič, ko je zagnana nova aktivnost, se prejšnja aktivnost ustavi. Sistem si ustavljeno aktivnost shrani na sklad. Sklad deluje po principu `LIFO` (zadnji notri, prvi ven). Ko uporabnik pritisne gumb "nazaj", sistem iz sklada vzame zadnjo aktivnost in jo uniči. Na vrh sklada pride prejšnja aktivnost, ki gre v izvajanje. Ko je aktivnost ustavljena, ker se je zagnala druga aktivnost, je ta o tem obveščena prek metod življenjskega cikla aplikacije. [9]

Življenjski cikel aplikacije

Aktivnost se lahko nahaja v naslednjih stanjih: v nadaljevanju, začasno ustavljena, ustavljena. Ko aktivnost prehaja med temi stanji, se prožijo metode iz življenjskega cikla aplikacije. Te metode so: `onCreate`, `onStart`, `onResume`, `onPause`, `onStop`, `onDestroy`. Cel življenjski cikel aplikacije se dogaja med klicema metod `onCreate` in `odDestroy`. Aplikacija v metodi `onCreate` zasede vire in jih sprosti v metodi `onDestroy`. Vidni del cikla aktivnosti se dogaja med metodama `onStart` in `onStop`. Ta čas lahko uporabnik uporablja uporabniški vmesnik aktivnosti. Ko pa je aktivnost v ospredju, se lahko sprožita metodi `onResume` in `onPause`.

2.3.2 Storitve

Storitev (angl. `Service`) je komponenta, ki teče v ozadju in izvaja dalj časa trajajočo operacijo. Storitev nima uporabniškega vmesnika. Druge komponente, na primer aktivnosti, lahko zaženejo storitev ali pa se na njo povežejo in potem z njo komunicirajo. Tudi če uporabnik zapre aktivnost, se storitev še vedno izvaja v ozadju.

Storitev ustvarimo tako, da ustvarimo podrazred razreda `Service` iz knjižnice `Android SDK`. Nato moramo napisati povratne metode storitve. Te metode so: `onStartCommand, onBind, onCreate, onDestroy`. [10]

2.3.3 Ponudniki vsebine

Ponudniki vsebine (angl. `Content providers`) upravljajo s shranjenimi podatki aplikacije. Aplikacija lahko podatke shrani na datotečni sistem, v podatkovno bazo `SQLite`, na splet ali v kakšno drugo trajno podatkovno lokacijo, do katere lahko aplikacija dostopa. Prek ponudnikov vsebine lahko tudi druge aplikacije poizvedujejo po podatkih in jih urejajo. [7]

2.3.4 Sprejemniki sporočil

Sprejemniki sporočil (angl. `Broadcast receivers`) so komponente, ki se odzivajo na zunanja opozorila. Sistem jim lahko tako sporoči, da ima naprava izpraznjeno baterijo, da se je zaslon naprave zaklenil itd. Sprejemniki nimajo uporabniškega vmesnika, kljub temu pa na vmesniku velikokrat prikažejo statusno vrstico za opozoritev uporabnika. [7]

2.4 Google API

`Google API` je dodatek orodja `Android SDK`. Omogoča razvoj aplikacij, ki vsebujejo Googlove komponente, knjižnice in storitve. Osrednja značilnost dodatka `Google API` je dodatek `Maps`, ki omogoča delo z zemljevidi. Skupaj s tem dodatkom dobimo tudi sistemsko sliko za `Android`, ki jo lahko poganjamo v emulatorju. Tako lahko v emulatorju testiramo vse funkcionalnosti, ki nam jih ta dodatek omogoča. [11]

2.4.1 Google Maps API

Dodatek `Google Maps API (Application programming interface)` omogoča dodajanje Googlovega zemljevida v aplikacijo za `Android`. `API` poskrbi za dostop do strežnikov `Google Maps`, prenos podatkov, prikaz zemljevida in odziv na dotike po zemljevidu. Omogoča pa tudi dodajanje oznak in poligonov ter spreminjanje geografskega pogleda zemljevida. Ti objekti omogočajo

dodatno informacijo o lokacijah na zemljevidu in uporabniku omogočajo interakcijo z njimi. [12]

Pridobitev ključa

Pred uporabo zemljevidov `Google Maps` v svoji aplikaciji je treba pridobiti ključ `Google Maps API`. Najprej se je treba strinjati s pogoji uporabe `API-ja`. Če testiramo uporabo zemljevidov na emulatorju `Android` ali mobilni napravi `Android`, priključeni na računalnik, moramo poiskati razhroščevalni certifikat, ki se na operacijskem sistemu `Windows 7` običajno nahaja v mapi `C:/Users/<username>/.android`. Ime datoteke je `debug.keystore`. To je certifikat, s katerim se podpiše naša aplikacija, ko je zagnana v emulatorju ali na mobilni napravi. Iz tega certifikata moramo pridobiti še odtis `MD5`, ki ga pridobimo z aplikacijo `Keytool.exe`. Ta aplikacija je v paketu `Java JDK`. Pridobljen odtis potem uporabimo na strani `https://code.google.com/apis/console`, ki nam vrne ključ za uporabo zemljevidov `Google Maps` v naši aplikaciji. [5]

Poglavje 3

Uporabljene tehnologije in storitve

3.1 Računalništvo v oblaku

Izraz računalništvo v oblaku se navezuje na aplikacije in storitve, ki tečejo na porazdeljenem omrežju in uporabljajo virtualizirane vire, do katerih dostopajo prek internetnih protokolov in omrežnih standardov. Beseda oblak izhaja iz dejstva, da so viri virtualizirani in neomejeni ter da je fizični sistem, na katerem teče programska oprema, uporabniku neviden. Za boljši opis računalništva v oblaku je bilo definiranih več tipov oblakov. Razdelimo jih lahko glede na vrsto postavitve ali pa tip storitve, ki jo ponujajo. Glede na vrsto postavitve ločimo javne, zasebne, skupnostne in hibridne oblake, glede na tip storitve pa ločimo modele infrastruktura kot storitev (angl. `Infrastucture as a Service - IaaS`), platforma kot storitev (angl. `Platform as a Service - PaaS`) in programska oprema kot storitev (angl. `Software as a Service - SaaS`). Ti modeli so zgrajeni eden na drugem in določajo, kaj mora zagotoviti ponudnik in kaj odjemalec. [4]

3.1.1 Infrastruktura kot storitev

Ponudniki infrastrukture kot storitve odjemalcu zagotovijo virtualne strežnike, virtualni prostor za shranjevanje podatkov, virtualno infrastrukturo in drugo strojno opremo. Ponudnik mora zagotoviti vso infrastrukturo, odjemalec pa poskrbi za druge stvari pri postavitvi, kar vključuje postavitev operacijskega sistem, aplikacij in drugih orodij, prek katerih uporabniki dostopajo do sistema. Primera te storitve sta `Amazon Elastic Compute Cloud (EC2)` in `Eucalyptus`. [4]

3.1.2 Platforma kot storitev

Pri tej vrsti storitve ponudnik zagotovi virtualne strežnike, operacijski sistem, aplikacije, storitve, ogrodje za razvijanje aplikacij, transakcije in kontrolne strukture. Odjemalec lahko naloži aplikacije na infrastrukturo oblaka ali pa uporabi aplikacije, ki so bile razvite v jeziku in orodjih, ki jih podpira ponudnik. Razvijalcem aplikacij ni treba skrbeti za nadgradnje in vzdrževanje strojne opreme ter operacijskega sistema, zato jim ostane več časa za razvoj aplikacij. Morajo pa poskrbeti za njihovo namestitev in upravljanje. Najbolj znana primera te storitve sta `Google App Engine` in `Microsoft Azure`. [4] [3]

3.1.3 Programska oprema kot storitev

Programska oprema kot storitev je polno delujoče okolje z aplikacijami, upravljanjem in uporabniškim vmesnikom. V tem modelu je aplikacija dostopna odjemalcu prek lahkega odjemalca (angl. `thin client`), ki je običajno dostopen prek spletnega brskalnika. Odgovornost odjemalca pa zajema upravljanje s podatki in interakcijo z uporabniki. Za upravljanje drugih stvari, od aplikacij do infrastrukture, pa skrbi ponudnik. Primera take aplikacije sta `Google Drive` in `Microsoft Office Web Apps`. [4]

Model, ki vsebuje te tri storitve, je znan pod imenom model `SPI`. Poleg teh treh tipov storitev obstaja še veliko drugih, na primer skladišče kot storitev

(angl. `StaaS - Storage as a Service`), omrežje kot storitev (angl. `NaaS - Network as a Service`), identiteta kot storitev (angl. `IdaaS (Identity as a Service`). Vendar pa model `SPI` združuje vse druge in je zato najbolj znan. [4]

3.1.4 Prednosti in slabosti računalništva v oblaku

Prednosti:

- **nadgradljivost:** v odvisnosti od obremenitve aplikacije se poveča ali zmanjša število virov. Prav zato je delovanje aplikacije dobro tudi ob veliki obremenitvi;
- **nižji stroški:** vzpostavitev sistema je cenejša, ker nam oblak ponuja vire, ki bi jih sicer morali kupiti sami. Zakupimo lahko točno toliko virov, kot jih potrebujemo. Enostavno lahko tudi povečamo ali zmanjšamo količino zakupljenih virov;
- **zanesljivost:** večje število virov in zmožnost uravnavanja njihove količine naredita sistem zanesljivejši. Običajno veliko bolj, kot bi lahko dosegli v eni organizaciji. Podatki in aplikacija so shranjeni na več mestih, zato je manjša možnost njihove izgube;
- **dostopnost:** dostop do virov v oblaku je možen prek interneta, zato lahko dostopamo do njih z različnih operacijskih sistemov in naprav;
- **zunanje upravljanje:** upravljanje z infrastrukturo lahko prepustimo drugim, tako da imamo mi več časa za upravljanje posla. Posledično se zmanjšajo tudi stroški osebja.

Slabosti:

- **varnost in zasebnost:** ker so podatki shranjeni na zunanjem sistemu, ki ni pod našo kontrolo, se moramo za varnost podatkov zanašati na nekoga drugega. Večja je tudi možnost prestrezanja podatkov;

- **možni izpadi:** odvisni smo od zanesljivosti internetne povezave, ki jo potrebujemo za dostop do storitev v oblaku;

- **pomanjkanje podpore:** uporabljamo storitve, ki jih sami ne poznamo v celoti, zato občasno potrebujemo tudi podporo. Lahko se zgodi, da ta ni na voljo;

- **omejena kontrola:** ker aplikacija in storitve tečejo na tujih virih, nam lahko ponudnik omeji dostop do določenih funkcij sistema.

3.2 Geocaching

`Geocaching` je aktivnost, v kateri udeleženci s pomočjo naprav `GPS`, mobilnih naprav ali drugih tehnik navigacije skrivajo in iščejo zaboje, poimenovane `geocache` ali `cache`. Zaboj je po navadi vodoodporen in vsebuje knjigo gostov, v katero uporabniki zabeležijo, da so našli zaboj. Primer takega zaboja je na prikazan sliki 3.1.

Slika 3.1: Primer zaklada Geocaching

Ko oseba najde zabojnik, ga mora odložiti na isto mesto, kjer ga je našla. V večjih zabojnikih so ponavadi tudi predmeti za menjavo. To so običajno predmeti manjše finančne vrednosti. Če se odločiš za menjavo, moraš predmet zamenjati z drugim, ki je enake ali večje finančne vrednosti.

Največja spletna stran za to aktivnost je `Geocaching.com`. V osnovi je sodelovanje v tej skupnosti brezplačno, lahko pa doplačamo za več funkcionalnosti. Uporabniki prihajajo iz več kot dvesto različnih držav iz vseh koncev sveta, tudi iz Slovenije. V Sloveniji je trenutno postavljenih že več kot 2500 zabojnikov. [17] Zaklade lahko na spletni strani iščemo na dva načina. Osnovno iskanje poteka prek izpolnitve vnosnih polj na strani. Tu lahko vrnjene rezultate filtriramo po naslovu, koordinatah, postavljavcu itd. Druga možnost pa so poizvedbe, poimenovane `Pocket Queries`. To so poizvedbe, ki nam rezultate vrnejo v obliki primerni za prenos s spletne strani. Izbiramo lahko med formatoma `GPX` in `LOC`. Te datoteke lahko prenesemo v namenske aplikacije. Poizvedbam lahko nastavimo tudi periodično ponavljanje. Rezultate dobimo po elektronski pošti. Razvijalci aplikacij lahko do podatkov dostopajo tudi prek `API-ja`. Trenutno ne sprejemajo več novih prošenj za dostop do tega vmesnika.

3.3 Google App Engine

`Google App Engine` je platforma za razvijanje aplikacij v računalniškem oblaku. Tip storitve, ki jo ponuja, je platforma kot storitev. Omogoča razvijanje in gostovanje spletnih aplikacij. Vse aplikacije v oblaku tečejo na več strežnikih. Platforma je načrtovana tako, da lahko streže v istem času več uporabnikom, in to brez izgube učinkovitosti. To pomeni, da je aplikacija nadgradljiva (angl. `scalable`). Odvisno od obremenitve aplikacije se aplikaciji dodeli več virov ali pa se ji jih odvzame. V osnovi je `Google App Engine` brezplačen. Uporabniki pa lahko doplačajo za več prostora za shranjevanje podatkov, hitrejšo internetno povezavo itd. Podprti jeziki za razvijanje aplikacij so Python, Java, PHP in Go. Zadnja dva sta trenutno še v poskusni

dobi. Platformo bi lahko razdelili na tri dele:

- instanca aplikacije;
- nagradljiva shramba podatkov;
- nagradljivi servisi.

Poleg tega platforma omogoča tudi povezavo z nekaterimi drugimi storitvami podjetja `Google`, na primer `Google Apps`, `Google Accounts`, `Google Cloud Storage`. [2] [14]

3.3.1 Izvajalno okolje

Aplikacija `App Engine` se odziva na spletne zahtevke. Te zahtevke običajno ustvari uporabnik prek spletnega brskalnika, ko pošlje zahtevo `HTTP` za nalaganje spletne strani. Ko `App Engine` sprejme zahtevek, najprej identificira aplikacijo prek domene spletnega naslova. Nato izbere strežnik, ki bo odgovoril na zahtevo. Z vidika aplikacije izvajalno okolje obstaja, ko se pojavi nov zahtevek, in izgine, ko se ta konča. Ker se stanje med zahtevki ne ohranja, lahko `App Engine` preusmeri promet na toliko strežnikov, kot je potrebno. Vsak zahtevek se nahaja v svojem "peskovniku"(angl. `sandbox`). Prav zato lahko `App Engine` odgovori na zahtevek v najkrajšem času. Ni zagotovila, da bo ista aplikacija odgovorila na dva zahtevka, čeprav sta prišla od istega uporabnika v kratkem časovnem obdobju. [2]

3.3.2 Razvojna orodja

`Google` omogoča brezplačna orodja za razvijanje aplikacij `App Engine` v jezikih `Java` in `Python`. Orodja lahko prenesemo iz `Googlove` spletne strani v obliki razvojnega paketa (`SDK`). Za integrirano razvojno okolje `Eclipse` obstaja tudi vtičnik, prek katerega lahko pridobimo `SDK` za jezik `Java`. Vsak `SDK` vključuje spletni strežnik, ki omogoča poganjanje aplikacij na lokalnem

računalniku in simulira izvajalno okolje. Ta strežnik avtomatsko zazna spremembe v kodi, zato ga imamo lahko med razvojem vseskozi zagnanega. Prek okolja `Eclipse` lahko vstavimo prekinitvene točke v kodo, ki se izvaja na strežniku. `SDK` nam prav tako omogoča komunikacijo z aplikacijo na platformi `App Engine`. To uporabljamo predvsem za nalaganje kode aplikacije na strežnik. Poleg tega `SDK` omogoča tudi pregledovanje podatkov v shrambi `Datastore`.

3.4 Google Cloud SQL

`Google Cloud SQL` je spletni servis, ki omogoča kreiranje, konfiguracijo in delo z relacijsko podatkovno bazo, ki je v Googlovem oblaku. Spletni servis je polno upravljan. Tako se izvaja vzdrževanje, upravljanje in administracija podatkovne baze. Razvijalci se lahko zato bolj osredotočijo na izdelavo aplikacij. Spletni servis uporablja podatkovno bazo `MySQL`. Uporaba tega spletnega servisa ni brezplačna. Treba je plačati uporabo na uro ali po uporabi. [21]

`Google Cloud SQL` je trenutno na voljo za aplikacije `Google App Engine`, napisane v jeziku `Java` ali `Python`. Do baze lahko dostopamo tudi prek ukazne vrstice in drugih orodij, ki so na voljo.

Funkcionalnosti:

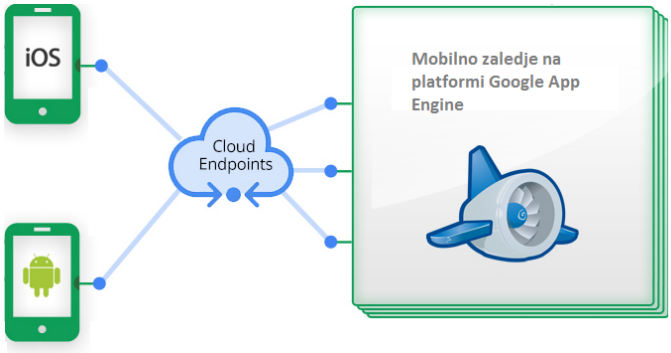
- instanca `Google Cloud SQL` ima lahko do 16 GB delovnega pomnilnika in 100 GB prostora za shranjevanje podatkov;
- dostop do baze prek spletnega vmesnika za izvajanje poizvedb `SQL`;
- vsi podatki so podvojeni na več lokacijah;
- kompatibilnost z jezikoma `Java` in `Python`;
- ukazna vrstica;

- podatki so shranjeni v Evropi ali Ameriki.

[22]

3.5 Google Cloud Endpoints

`Google Cloud Endpoints` je tehnologija, ki s pomočjo orodij in knjižnic omogoča izdelavo `API-jev` za dostop do podatkov aplikacij `App Engine`. Uporabniški dostop do podatkov se zaradi tega poenostavi. `API` lahko uporabljajo tako spletni klienti prek `Javascripta` kot tudi mobilni prek platform Android in iOS. Povezavo mobilnih odjemalcev na `Google App Engine` prikazuje slika 3.2.



Slika 3.2: Povezava naprav na platformo Google App Engine prek tehnologije Google Cloud Endpoints

Z uporabo te tehnologije lahko razvijalci na enostaven način razvijejo `API-je` za dostop do podatkov. Ker je `API` še vedno del aplikacije `App Engine`, lahko razvijalci uporabljajo vse storitve in funkcionalnosti, ki jih ta omogoča. Razvijalcem prav tako ni treba skrbeti za sistemsko administracijo in vzdrževanje strežnikov. [15]

3.5.1 Notacije Endpoint

Notacije `Endpoint` se uporabljajo za konfiguracijo `API-ja`, metod, parametrov in drugih podrobnosti, ki določajo njihove lastnosti in obnašanje. Notacija, ki določa konfiguracijo celotnega `API-ja` je `@Api`. Za konfiguracijo posamezne metode se uporablja notacija `@ApiMethod`. [15]

Primer notacije:

```
@Api(
    version = "v1",
    description = "Sample API",
    scopes = {"ss0", "ss1"},
    audiences = {"aa0", "aa1"},
    clientIds = {"cc0", "cc1"},
    defaultVersion = AnnotationBoolean.TRUE
)
```

3.6 Java Http Servlet

`Servlet` je razred, napisan v programskem jeziku `Java`, in se uporablja za razširitev zmogljivosti spletnega strežnika. Čeprav se lahko odziva na več različnih zahtev, se običajno uporablja za odziv na zahteve, ki prihajajo s spletnih strežnikov. V principu lahko `servlet` komunicira prek katerega koli protokola odjemalec-strežnik, ampak najbolj pogosto se uporablja protokol `HTTP`. Spletni razvijalci lahko uporabljajo `servlete` za dodajanje dinamične vsebine na spletne strani. Generirana vsebina je običajno `HTML`, lahko pa je tudi kaj drugega, na primer `XML`. `Servlet` lahko ohrani stanje prek več transakcij v sejni spremenljivki, piškotku `HTTP` ali prek spreminjanja naslova `URL`. [19]

`Servlet` sprejema zahteve in generira odziv nanje. Za generiranje odzivov se uporablja `Java Servlet API`. Ta je vsebovan v platformi `Java EE (Java Enterprise Edition)` kot del paketa `javax.servlet`. Za servlete

HTTP je namenjen paket javax.servlet.http, ki definira za HTTP specifične podrazrede za servlete. [19]

3.6.1 Življenjski cikel servleta

Življenjski cikel servleta je sestavljen iz treh metod: init(), service() in destroy(). Implementirane so za vsak servlet in jih strežnik kliče ob določenem času. Metoda init() se kliče ob inicializaciji servleta in se je lahko poda kot parameter objekt, ki implementira vmesnik javax.servlet.ServletConfig. Prek tega objekta lahko servlet dostopa do parametrov spletne aplikacije. Po inicializiaciji lahko servlet sprejema zahteve. Vsaka zahteva je sprocesirana v ločeni niti. Za vsako zahtevo se kliče tudi metoda service(). Ob zaključku življenjskega cikla servleta pa se kliče še metoda destroy(), ki ustavi njegovo delovanje. [19] [20]

Znotraj metode service() se najpogosteje uporabljata metodi doGet in doPost. To sta metodi, ki določata odziv servleta HTTP na zahtevka GET in POST. Zahtevek GET pošljemo z namenom pridobitve podatkov s strežnika, zahtevek POST pa za procesiranje podatkov.

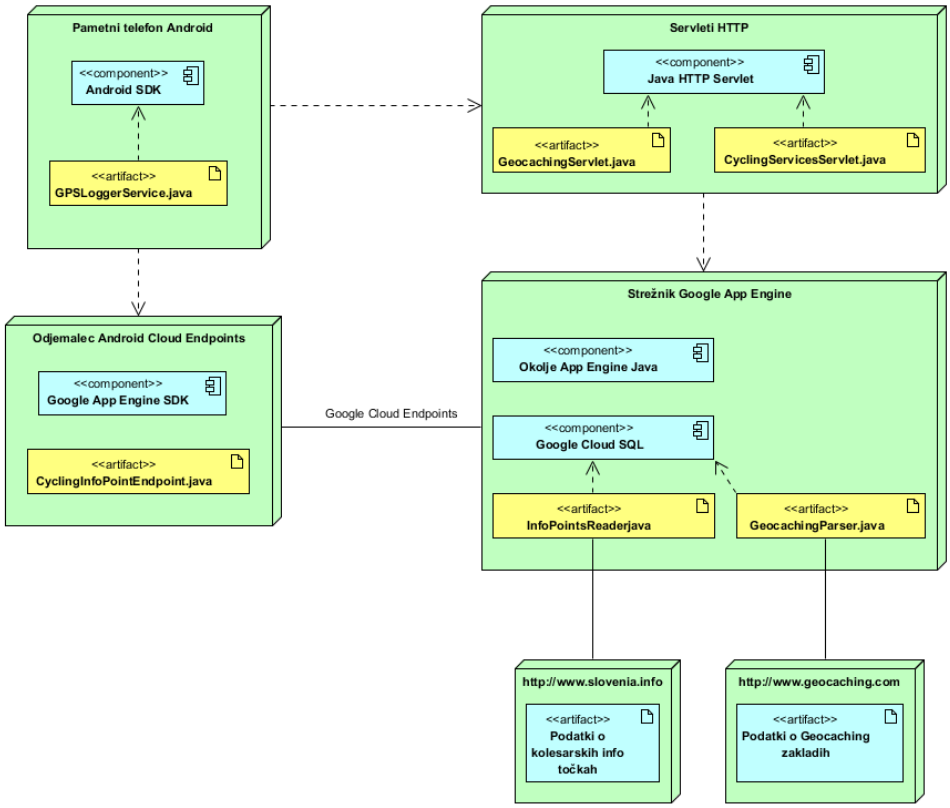
Poglavje 4

Razvoj informacijskega sistema

Informacijski sistem je sestavljen iz spletnega strežnika in mobilne aplikacije. Spletni strežnik je razvit na platformi `Google App Engine`, mobilni klient pa uporablja operacijski sistem `Android`. Spletni strežnik skrbi za shranjevanje podatkov in delo s podatki. Vsebuje podatkovno bazo `Google Cloud SQL`. Tu se hranijo podatki, do katerih lahko uporabniki dostopajo prek mobilne aplikacije. Spletni strežnik posreduje podatke mobilnemu klientu v enostavni strukturi. Zaradi tega je razčlenjevanje podatkov manj zahtevno.

4.1 Arhitektura sistema

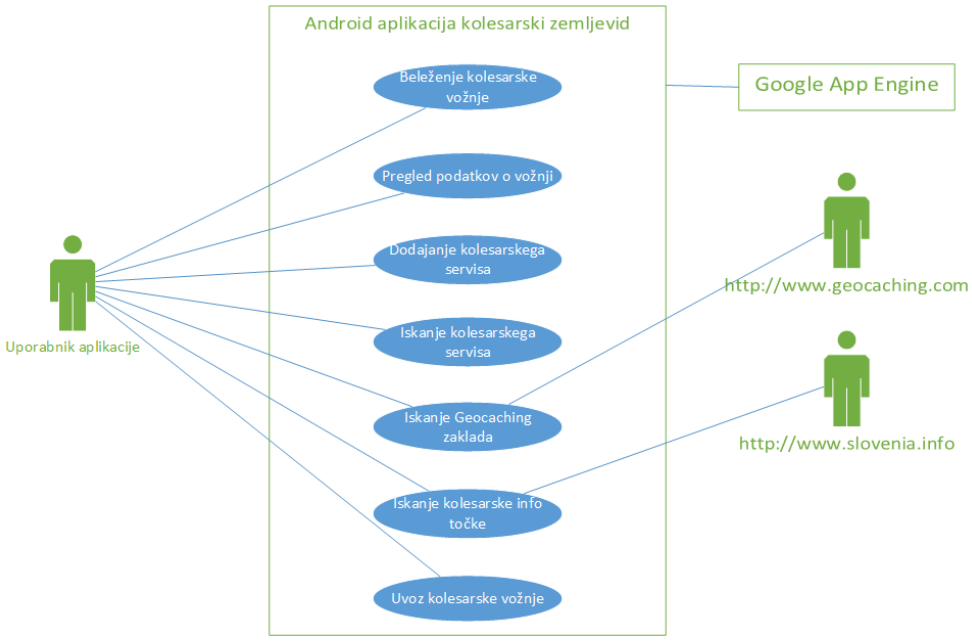
Povezava med mobilnim klientom in spletnim strežnikom je narejena prek `servletov HTTP` in odjemalca `Google Cloud Endpoint`. `Servleti HTTP` se uporabljajo za dostop do podatkov o zakladih `Geocaching` in kolesarskih servisih, odjemalec `Google Cloud Endpoint` pa za pridobitev podatkov o kolesarskih informacijskih točkah. Prikaz elementov sistema in njihove povezave znotraj informacijskega sistema prikazuje slika 4.1.



Slika 4.1: Diagram UML, ki prikazuje arhitekturo informacijskega sistema

4.2 Ključne funkcije sistema

Osnovni funkcionalnosti aplikacije sta beleženje in pregled kolesarskih voženj. Ko se uporabnik odpravi kolesarit, lahko v aplikacij zažene beleženje. Aplikacija mu sproti prikazuje trenutno pretečen čas, prevoženo razdaljo in povprečno hitrost. Pri pregledu kolesarskih voženj ima uporabnik več možnosti. Tako si lahko do sedaj shranjene vožnje ogleda v obliki seznama. Za vsako kolesarsko vožnjo lahko pregleda podrobnosti ali pa jo prikaže na zemljevidu. Drugi način za dodajanje nove vožnje je uvoz vožnje prek datoteke `gpx`. Na ta način lahko uporabnik uvozi v aplikacijo tudi druge vožnje. Uporabnik lahko v aplikaciji doda kolesarski servis ali pa ga poišče na zemljevidu. S tem si lahko uporabniki med sabo sporočajo o lokacijah kolesarskih servisov. Ena



Slika 4.2: Diagram UML primerov uporabe aplikacije

izmed funkcionalnosti aplikacije je iskanje kolesarskih informacijskih točk. Poleg tega pa aplikacija omogoča tudi prikaz zakladov, ki so jih uporabniki dodali na spletnem portalu `geocaching.com` in so na območju Slovenije.

Na sliki 4.2 so prikazani prej našteti primeri uporabe aplikacije na diagramu `UML`. Na diagramu so trije akterji: uporabnik aplikacije ter spletna portala `geocaching.com` in `slovenia.info`. Portala sta prikazana, ker sta vir podatkov za iskanje kolesarskih informacijskih točk in zakladov `Geocaching`. Poleg akterjev pa je prikazan tudi podsistem `Google App Engine`. Na njem se nahaja spletni strežnik, ki se uporablja za shranjevanje in dostop do podatkov.

4.3 Razvoj mobilne aplikacije

4.3.1 Izdelava uporabniškega vmesnika

Vmesnik aplikacije smo poskusili narediti čim preglednejši in enostavnejši za uporabo. Na začetnem zaslonu aplikacije je prikazanih šest velikih ikon, ki so vstopne točke za posamezne aktivnosti. Vsaka ikona je tudi podnaslovljena s tekstom, da uporabnik lažje razpozna namen ikone. Začetni zaslon aplikacije prikazuje slika 4.3.

Slika 4.3: Začetni zaslon aplikacije

Vmesnike posameznih aktivnosti aplikacije smo zgradili prek datotek `XML`. V njih so navedeni vsi elementi vmesnika in njihove lastnosti. Elementi, ki smo jih uporabili pri izdelavi uporabniških vmesnikov, so:

- **Button:** element za izvajanje akcij ob dotiku;
- **TextView:** omogoča prikaz teksta na uporabniškem vmesniku. Uporabljamo ga vedno, kadar želimo dodati statičen tekst na uporabniški vmesnik;

- **EditText:** element, ki omogoča vnos v tekstovno polje;
- **ImageView:** omogoča prikaz slik;
- **LinearLayout:** določa postavitev elementov na zaslonu. Ti so lahko razporejeni linearno, vertikalno ali horizontalno;
- **RelativeLayout:** namenjen je za razporeditev elementov na uporabniški vmesnik na podlagi pozicij drugih elementov;
- **TableLayout:** elemente na zaslonu razporedi v tabelarično obliko;
- **TableRow:** predstavlja vrstico elementov v `TableLayout`;
- **ImageButton:** omogoča enako funkcionalnost kot `Button`. Poleg tega mu lahko nastavimo sliko za ozadje;
- **TabHost:** uporablja se kot osnovni element za dodajanje zavihkov;
- **TabWidget:** prikazuje seznam oznak zavihkov;
- **FrameLayout:** uporablja se za izoliranje prostora na zaslonu za en sam element;
- **Fragment:** uporablja se za prikaz fragmenta. V našem primeru smo kot fragment prikazali zemljevid.

4.3.2 Pridobivanje podatkov

Osnovni podatki o kolesarskih aktivnostih, ki jih aplikacija posreduje uporabniku, so ime poti, datum vožnje, prevožena razdalja, čas vožnje, čas začetka, čas konca, povprečna hitrost in vzpon. Za pridobivanje teh podatkov smo naredili storitev, ki beleži lokacijo uporabnika. Uporabnik zažene storitev tako, da pritisne na gumb "nova aktivnost". Ko je storitev zagnana, periodično na dve sekundi poizveduje o lokaciji uporabnika. Če se je uporabnik v dveh sekundah premaknil za vsaj en meter, si storitev lokacijo shrani.

Storitev razširja razred `Service`. Ta razred zahteva implementacijo metode `onBind()`, ki pa v našem primeru vrne vrednost `null`, ker ne podpiramo povezavanja na storitev po tem, ko je bila zagnana. Ker se storitev v osnovi izvaja v glavni niti aplikacije, smo jo spremenili tako, da se izvaja v ozadju. S tem smo preprečili preobremenitev aplikacije. To smo storili tako, da smo izvajanje storitve izvedli z razredom `HandlerThread`, ki zažene novo nit.

```
HandlerThread thread = new HandlerThread("ServiceStartArguments",
                Process.THREAD_PRIORITY_BACKGROUND);
thread.start();
```

Opravilo, ki ga izvaja nit, smo določili prek razreda `Handler`. Ta v metodi `handleMessage()` zažene poslušalca, ki zaznava spremembo pozicije uporabnika. Za pridobivanje lokacije storitev uporablja komponento `LocationManager`. Pridobimo jo s klicem metode `getSystemService()`, ki od mobilne naprave pridobi storitve za upravljanje z lokacijo.

```
LocationManager mLocationManager = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);


mLocationManager.requestLocationUpdates(LocationManager.
    GPS_PROVIDER, TWO_SECONDS, ONE_METER, listener);
```

`LocationManager` pridobiva lokacijo iz `GPS-a` in spletnega ponudnika mobilne naprave. Če ima uporabnik katerega izmed teh dveh ponudnikov onemogočena, se lokacija pridobi od tistega ponudnika, ki je omogočen. V primeru, da pridobimo lokacijo od obeh ponudnikov, primerjamo lokaciji in izberemo natančnejšo

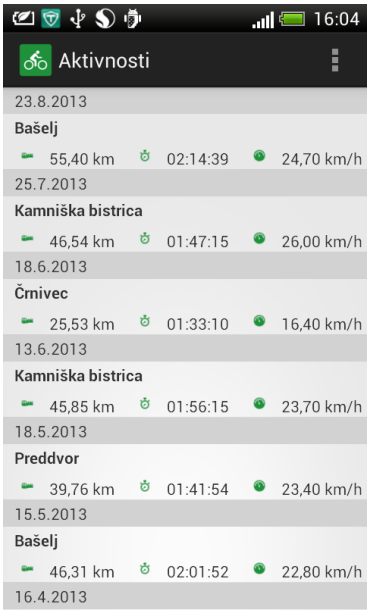
Med izvajanjem aktivnosti ima uporabnik tudi možnost spremljanja porabljenega časa, prevožene razdalje in trenutne povprečne hitrosti.

Uporabnik konča aktivnost s pritiskom na gumb "končaj aktivnost". Storitev, ki beleži lokacijo uporabnika, se zaključi in shrani podatke v datotekah končnic `gpx` in `dat`. Datoteka `gpx` vsebuje vse shranjene lokacije, ki jih je pridobila storitev. Format `gpx` je univerzalen format za zapis koordinat `GPS`.

Uporabnik lahko kasneje to datoteko izvozi in jo uvozi v katerega izmed drugih programov za analizo športnih aktivnosti. Ob izdelavi datoteke `gpx` pa se izračunajo tudi podatki, ki se zapišejo v datoteko končnice `dat`. Ta vsebuje podatke o imenu vožnje, prevoženi razdalji, času, času začetka, času konca, povprečni hitrosti in vzponu.

4.3.3 Pregled aktivnosti

Aplikacija omogoča pregled vseh shranjenih voženj. Uporabnik lahko do teh podatkov dostopa prek menija aktivnosti. Tu so vse aktivnosti prikazane v obliki seznama. Seznam v odvisnosti od števila podatkov dobi tudi drsnike. Tako je omogočen pregled vseh aktivnosti prek enega zaslona. Izgled seznama je prikazan na sliki 4.4.



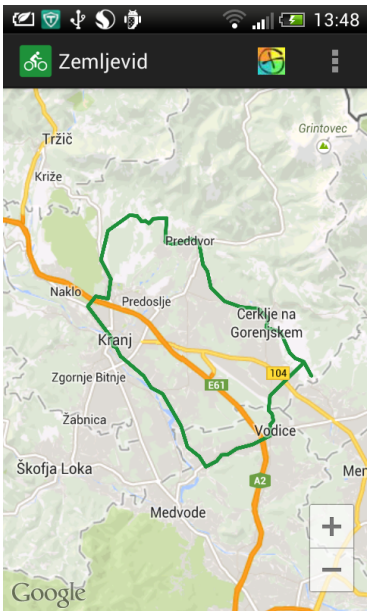
Slika 4.4: Izgled aktivnosti, ki prikazuje shranjene vožnje

To smo naredili na način, da smo ustvarili novo aktivnost, ki razširja razred `ListActivity`. Tej aktivnost nastavimo podatke tako, da kličemo metodo `setListAdapter()`, ki ji podamo razred, ki razširja razred `ListAdapter`.

```
public class RoutesActivity extends ListActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      ...

      CyclingRoute[] routes = getSavedRoutes();
      setListAdapter(new RoutesArrayAdapter(this, R.layout.
         routes_rowlayout, routes));
   }
}
```

Naredili smo še razred `RoutesArrayAdapter`, ki razširja razred `ArrayAdapter`. V tem razredu nastavimo izgled posameznega elementa seznama. To storimo tako, da najprej ustvarimo datoteko `XML`, kjer določimo elemente in njihovo postavitev. Znotraj tega razreda poiščemo elemente z metodo `findViewById()` in jim v metodi `getView()` nastavimo vrednosti, ki smo jih dobili ob klicu razreda.

```
public class RoutesArrayAdapter extends ArrayAdapter<CyclingRoute>
    {
   ...

   @Override
   public View getView(int position, View convertView, ViewGroup
       parent) {
      LayoutInflater inflater = (LayoutInflater) context.
         getSystemService(Context.LAYOUT_INFLATER_SERVICE);
      View rowView = inflater.inflate(layoutId, parent, false);

      TextView textViewName = (TextView) rowView.findViewById(R.id.
         txtRowRouteName);
      CyclingRoute route = objects[position];
```



Slika 4.5: Izgled aktivnosti, ki prikazuje vožnjo na zemljevidu

```
        textViewName.setText(route.getName());
        ...

        return rowView;
    }
}
```

Pregled ob kliku na posamezno aktivnost omogoča tudi prikaz podrobnosti. Prikazani so naslednji podatki: ime poti, datum, razdalja, čas, čas začetka, čas konca, povprečna hitrost in vzpon. Lahko pa vožnjo prikažemo na zemljevidu. Primer prikaza je na sliki 4.5.

Zemljevid smo v aplikaciji prikazali prek aktivnosti `FragmentActvity`. Ta se nahaja v paketu `android.support.v4.app`. V osnovi je podrazred razreda `Activity` in vsebuje nekaj dodatnih metod za zagotavljanje kompatibilnosti s starejšimi verzijami `Androida`. Za prikaz zemljevida `Google Maps` moramo poklicati metodo `getSupportFragmentManager()`, ki iz datoteke `XML` pridobi fragment razreda `com.google.android.gms.maps.SupportMapFragment`.

```
GoogleMap mMap = ((SupportMapFragment) getSupportFragmentManager().
    findFragmentById(R.id.map))
```

Ko smo pridobili zemljevid, smo prevoženo pot narisali z razredom `PolygonOptions`. Temu razredu podamo geografske koordinate, ki jih želimo narisati prek objektov tipa `LatLng`. Nastavimo lahko še druge parametre, na primer barvo in debelino črte.

```
LatLng startPosition = null;
PolygonOptions routeDriven = new PolygonOptions();

for(int i=0; i<arrayLatLng.length; i++) {
   if(i == 0)
      startPosition = (LatLng) arrayLatLng[i];
   routeDriven.add((LatLng) arrayLatLng[i]);
}

routeDriven.strokeColor(Color.parseColor("#FF1E913A"));
routeDriven.strokeWidth(5);

Polygon polygon = mMap.addPolygon(routeDriven);
```

4.3.4 Kolesarske informacijske točke

Po Sloveniji deluje petintrideset kolesarskih informacijskih točk. Na teh točkah lahko pridobimo zemljevide kolesarskih poti izvemo, kje se lahko nastanimo in kje lahko kupimo rezervne dele za kolo ali ga popravimo. Med drugim lahko najamemo vodnika ali rezerviramo vodeno turo. Podatke o imenu in lokaciji kolesarskih informacijskih točk smo pridobili s spletne strani `www.slovenia.info`. Te podatke smo prebrali na spletni strani in iz njih ustvarili stavke `SQL insert`. Te stavke smo potem izvedli na podatkovni bazi `Google Cloud SQL`.

Za dostop do kolesarskih informacijskih točk iz mobilne naprave smo na-

redili API, ki se prek Googlove tehnologije Cloud Endpoint poveže na bazo Cloud SQL in iz nje pridobi podatke ter jih vrne v obliki JSON. Opis implementacije servleta je v poglavju spletni strežnik.

Pridobivanje podatkov prek tehnologije Cloud Endpoints

Za povezavo na App Engine prek tehnologije Cloud Endpoints moramo najprej zgenerirati knjižnice Endpoint. Prek vtičnika Google za okolje Eclipse lahko to storimo prek klika na meni Generate Cloud Endpoint Client Library. Projektu moramo v mapo libs dodati še naslednje datoteke:

- google-api-client-1.12.0-beta.jar;
- google-api-client-android-1.12.0-beta.jar;
- google-http-client-1.12.0-beta.jar;
- google-http-client-android-1.12.0-beta.jar;
- google-http-client-gson-1.12.0-beta.jar;
- google-oauth-client-1.12.0-beta.jar;
- gson-2.1.jar;
- guava-jdk5-13.0.jar;
- jsr305-1.3.9.jar.

Za dostop do Endpoint API-ja moramo ustvariti storitveni objekt. Prek tega objekta lahko potem dostopamo do vseh metod, ki jih API omogoča. Primer dostopa do kolesarskih informacijskih točk prek Endpoint API-ja:

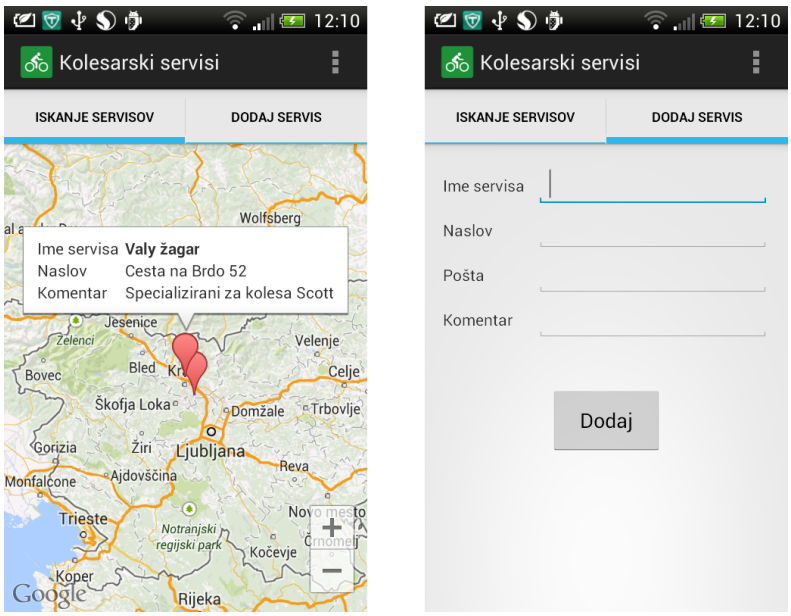
```
public class CyclingInfoPointEndpointsTask extends AsyncTask<
    Context, Integer, List<CyclingInfoPoint>> {
   protected List<CyclingInfoPoint> doInBackground(Context... maps)
       {
      List<CyclingInfoPoint> cyclingInfoPoints = new ArrayList<
         CyclingInfoPoint>();
```

```
      Cyclinginfopointendpoint.Builder endpointBuilder = new
          Cyclinginfopointendpoint.Builder(
            AndroidHttp.newCompatibleTransport(), new JacksonFactory
                (),
            new HttpRequestInitializer() {
               public void initialize(HttpRequest httpRequest) {
               }
            });

      Cyclinginfopointendpoint endpoint = CloudEndpointUtils.
          updateBuilder(
            endpointBuilder).build();

      try {
         CyclingInfoPointListContainer listContainer = endpoint.
             listCyclingInfoPoint().execute();
         cyclingInfoPoints = listContainer.getItems();
      } catch (IOException e) {
         e.printStackTrace();
      }
      return cyclingInfoPoints;
   }
}
```

Objekt z imenom `endpoint` je v tem primeru storitveni objekt. Prek njega kličemo metodo `listCyclingInfoPoint()`, ki nam vrne seznam vseh kolesarskih informacijskih točk iz podatkovne baze. Ker seznama ne moremo neposredno prenašati prek `Endpoints API-ja`, smo naredili še vmesni razred `CyclingInfoPointListContainer`, ki vsebuje atribut seznam ter metodi `getter` in `setter` za dostop do seznama.

4.3.5 Kolesarski servisi

Aplikacija omogoča tudi dodajanje in iskanje kolesarskih servisov. Namen tega je, da si uporabniki med sabo posredujejo informacije, katere kolesarske servise obiskujejo. Če kolesarimo v bolj oddaljen kraj in se nam med izletom poškoduje kakšen del kolesa, nam aplikacija omogoči pregled servisov v bližini. Uporabnik, ki doda kolesarski servis, lahko dopiše tudi komentar, s katerim drugim uporabnikom posreduje dodatne informacije o kolesarskem servisu. Izgled uporabniškega vmesnika prikazuje slika 4.6.



Slika 4.6: Uporabniški vmesnik aktivnosti "kolesarski servisi"

Podatki, ki jih vnesejo uporabniki, se shranijo v podatkovno bazo `Google Cloud SQL` na strežniku `App Engine`. Za prenos podatkov v podatkovno bazo smo implementirali `servlet HTTP`. Uporabnik vnese ime servisa, naslov, pošto in komentar. Ti podatki se shranijo v podatkovno bazo. Dodatno pa iz podatka o naslovu in pošti izračunamo tudi geografske koordinate servisa. Za to opravilo smo uporabili `Google Geocoding API`. Ta `API` omogoča pretvorbo naslova lokacije v geografske koordinate. Podatke posredujemo

API-ju prek parametrov GET. Obvezen parameter output določa tip vrnjenih podatkov. Izbiramo lahko med JSON in XML. Druga obvezna parametra sta še naslov lokacije (angl. address) in ali smo pridobili lokacijo prek senzorja kakšne naprave (angl. sensor). [18] Mi smo se odločili za format JSON. Parameter senzor pa smo pri vseh klicih nastavili na false, ker uporabniki ročno vnašajo naslove kolesarskih servisov.

Za natančnejše določanje naslova lokacije, ki ga uporabimo pri klicih Geocoding API-ja, smo implementirali tudi pridobivanje poštnih številk krajev po Sloveniji. Iz spletne strani Pošte Slovenije smo si shranili podatke o številkah in imenih pošt. Te podatke smo vstavili v podatkovno bazo Google Cloud SQL prek programa, ki smo ga napisali v programskem jeziku Java. Ko uporabnik vnese kraj pošte, se kot klic Geocoding API uporabi tudi pridobljena številka o pošti.

Servlet poleg dodajanja kolesarskih servisov omogoča tudi pridobitev podatkov o vseh shranjenih kolesarskih servisih. Klic servleta za pridobitev vseh kolesarskih servisov izvedemo tako, da podamo parameter GET, in sicer list. Primer klica: http://kolesar-android.appspot.com/services?list. Pri tem klicu se iz podatkovne baze preberejo podatki o vseh shranjenih kolesarskih servisih in posredujejo v obliki JSON. Vrnjene podatke v mobilni aplikacij preberemo in prikažemo na zemljevidu.

Za branje podatkov o kolesarskih servisih smo naredili razred, ki razširja abstraktni razred AsyncTask:

```
private class ListCyclingServicesTask extends AsyncTask<Void, Void,
    String> {
  @Override
  protected String doInBackground(Void... params) {
     ...
  }
}
```

To je potrebno zato, ker ni dovoljeno izvajati omrežnih zahtevkov v glavni niti aplikacije. Ta razred se najprej poveže na spletni strežnik in izvede

poizvedbo `GET` za pridobitev vseh kolesarskih servisov. Omrežni zahtevek smo naredili prek objektov `URL, URLConnection in HttpURLConnection` iz knjižnice `java.net`.

```
private InputStream getHttpConnection(String urlString)
      throws IOException {
   InputStream stream = null;
   URL url = new URL(urlString);
   URLConnection connection = url.openConnection();

   try {
      HttpURLConnection httpConnection = (HttpURLConnection)
          connection;
      httpConnection.setRequestMethod("GET");
      httpConnection.connect();

      if (httpConnection.getResponseCode() == HttpURLConnection.
          HTTP_OK) {
         stream = httpConnection.getInputStream();
      }
   } catch (Exception ex) {
      ex.printStackTrace();
   }

   return stream;
}
```

V nadaljevanju smo pridobili vsebino iz `InputStreama` prek razredov `BufferedReader` in `InputStreamReader`. Tako pridobljen objekt `JSON` smo pretvorili v objekt `Java`. Za pretvorbo smo uporabili knjižnico `json-simple`. To je knjižnica, ki omogoča enostavno in hitro delo z objekti `JSON` v `Javi`. Najprej je treba inicializirati objekt `JSONParser`, nato pa lahko prek tega objekta s klicem metode `parse()` pridobimo objekt tipa `Object`. Glede na podatke, ki jih imamo, pa ta objekt pretvorimo v objekt `JSONObject` ali

JSONArray.

```
JSONParser parser = new JSONParser();
JSONObject service;
ServiceCycling cyclingService;
ArrayList<ServiceCycling> arrayServices = new ArrayList<
    ServiceCycling>();

try {
   Object obj = parser.parse(response);
   JSONArray array = (JSONArray)obj;

   for(int i=0; i<array.size(); i++) {
      service = (JSONObject)array.get(i);
      cyclingService = getCyclingServiceFromJson(service);
      arrayServices.add(cyclingService);
   }
} catch(org.json.simple.parser.ParseException ex) {
   Log.d("napaka", "Prislo je do napake pri pridobivanju kolesarskih
        servisov");
}
```

Oba objekta prek metode `get()` omogočata dostop do podelementov objekta. Pri `JSONObject` so to atributi objekta, pri `JSONArray` pa objekti `JSONObject`. Tako smo za kolesarske servise pridobili podatke iz `JSONObject` in prek metod `setter` nastavili atribute našemu javanskemu razredu `ServiceCycling`.

```
private ServiceCycling getCyclingServiceFromJson(JSONObject service
    ) {
   ServiceCycling resService = new ServiceCycling();

   resService.setName(service.get("name").toString());
   resService.setAddress(service.get("address").toString());
   resService.setComment(service.get("comment").toString());
   resService.setLatitude(Double.parseDouble(service.get("latitude")
```

```
        .toString()));
    resService.setLongitude(Double.parseDouble(service.get("longitude
        ").toString()));
    return resService;
}
```

4.3.6 Geocaching

Povezavo mobilne aplikacije s spletno skupnostjo `Geocaching` smo naredili prek spletnega strežnika `App Engine`. Najprej smo pridobili vse zabojnike v Sloveniji s spletne strani `Geocaching.com`. Podatke o zabojnikih smo pridobili s pomočjo poizvedb na strani `Geocaching.com`. Ker ena poizvedba vrne maksimalno tisoč rezultatov, smo morali izvesti več poizvedb. V vsaki smo določili drug razpon datumov, ob katerih so bili zabojniki postavljeni. Rezultate poizvedb smo dobili v obliki datotek končnice `gpx`. Datoteka vsebuje podatke o zabojniku, kot so pozicija `GPS`, ime, datum postavitve, ime postavitelja, povezavo `URL` do zaklada na strani `Geocaching.com`, tip zaklada, zahtevnost, teren, opis, namig, tip zabojnika in vse do sedaj zabeležene najdbe zabojnika.

Za prenos podatkov iz datotek `gpx` v podatkovno bazo `Google Cloud SQL` na spletnem strežniku `App Engine` smo napisali program v programskem jeziku `Java`. Program prebere podatke iz datoteke `gpx`, jih pretvori v primerno obliko za vstavljanje v podatkovno bazo in nato vstavi.

Za povezavo zunanjih programov na `Cloud SQL` moramo najprej vsaj enkrat dostopiti do baze prek konzolne vrstice. Ob prvem dostopu se shrani žeton `OAuth 2.0`, ki ga aplikacija potrebuje tudi za kasnejše dostope do baze. Za povezavo na bazo potrebujemo še gonilnik, ki ga dobimo tako, da v projekt uvozimo še datoteko `google_sql.jar`. Datoteka se nahaja v mapi, kjer je nameščena konzolna vrstica za dostop do baze.

Datoteke `gpx` so v obliki `XML`. Del takega `XML-ja`:

```
<wpt lat="46.41" lon="13.72485">
```

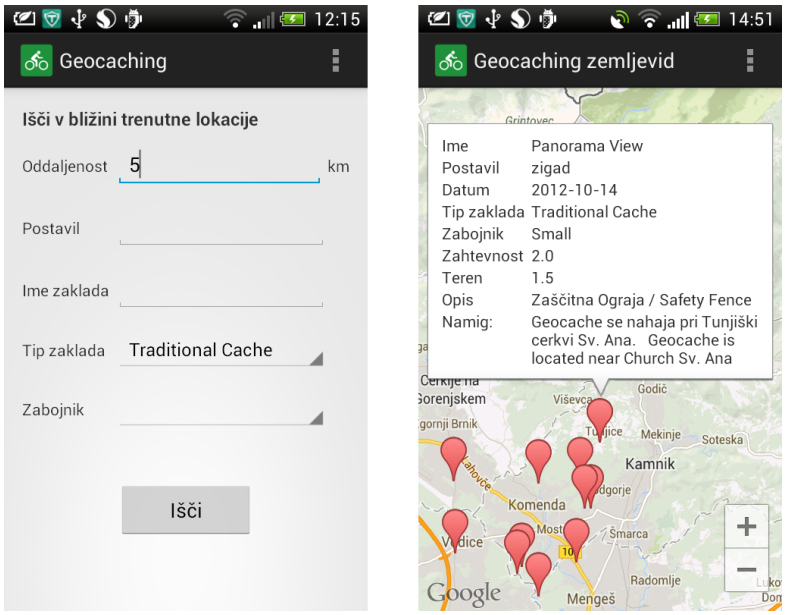
```
    <time>2001-08-28T07:00:00Z</time>
    <name>GC1C7E</name>
    <desc>Triglav National Park / Izviru Soce by Pythagoras,
       Traditional Cache (2/1.5)</desc>
    <url>http://www.geocaching.com/seek/cache_details.aspx?guid=
       b7304bdf-025d-4fad-8102-07242220179d</url>
    <urlname>Triglav National Park / Izviru Soce</urlname>
    <sym>Geocache</sym>
    <type>Geocache|Traditional Cache</type>
</wpt>
```

Za branje teh datotek smo uporabili razred `DocumentBuilder` iz paketa `javax.xml.parsers`. Ta pretvori datoteko v objekt, ki vsebuje podatke v drevesni strukturi. Takemu modelu pravimo `DOM - Document Object Model`. Prek tega objekta lahko potem iščemo elemente po drevesni strukturi. Primer branja datoteke `gpx`:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance
    ();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(file);

doc.getDocumentElement().normalize();
NodeList geoCaches = doc.getElementsByTagName("wpt");
```

Prikaz zakladov v mobilni aplikaciji

Uporabnik lahko v mobilni aplikaciji dostopa do zakladov `Geocaching` na dva načina. Prvi je iskanje zakladov v trenutni bližini, drugi pa pregled zakladov na že prevoženih kolesarskih poteh. Za prikaz zakladov v trenutni bližini lahko uporabnik določi oddaljenost, na kateri želi poiskati zaklad, ime postavitelja, ime zaklada, tip zaklada in tip zabojnika. Glede na iskane parametre mu aplikacija na zemljevidu prikaže zaklade in dodatne informacije o njih. Iz slike 4.7 sta razvidna iskanje in prikaz zakladov.



Slika 4.7: Iskanje in prikaz zakladov Geocaching v trenutni okolici

Druga možnost je prikaz zakladov na že shranjenih aktivnostih. Ob pregledu aktivnosti na zemljevidu je v akcijski vrstici ikona `Geocaching`. Ob pritisku na to ikono, se za vsako dvestoto točko prevožene poti izrišejo bližnji zakladi v okolici dveh kilometrov. Podatke pridobimo prek `servleta HTTP` v obliki `JSON` in jih s pomočjo knjižnice `json-simple` pretvorimo v objekte `Java`, ki jih uporabimo za prikaz na zemljevidu.

4.4 Spletni strežnik

4.4.1 Servleti HTTP

Povezava servletov HTTP s spletno aplikacijo

Povezavo `servleta HTTP` s spletno aplikacijo naredimo v datoteki `web.xml`, ki je del projekta spletne aplikacije. Za vsak `servlet` je treba nastaviti še ime in podati pot do razreda, kjer je implementiran. Poleg tega je treba nastaviti še povezavo `URL` ali vzorec `URL`, prek katerega dostopamo do `servleta`. To

```
<servlet>
  <servlet-name>GeocachingServlet</servlet-name>
  <servlet-class>com.androidkolesar.GeocachingServlet</servlet-
      class>
</servlet>

<servlet-mapping>
  <servlet-name>GeocachingServlet</servlet-name>
  <url-pattern>/geocaching/*</url-pattern>
</servlet-mapping>
```

Slika 4.8: Konfiguracija servleta Geocaching

naredimo v datoteki `web.xml` prek značk `servlet` in `servlet-mapping`. Na sliki 4.8 je primer take konfiguracije za `servlet Geocaching`, ki smo ga razvili v okviru diplomskega dela. Tu smo določili ime `servleta`, in sicer `GeocachingServlet`, in podali polno pot do razreda `GeocachingServlet`. Določili smo še vzorec `URL`, ki določa pot do `servleta` iz vseh naslovov `URL`, ki se začnejo z `geocaching`.

Servlet Geocaching

Mobilna aplikacija dostopa do podatkov o zabojnikih `Geocaching` prek `servleta` `HTTP`. `Servlet` smo sprogramirali v programskem jeziku `Java`. Omogoča iskanje zabojnikov `Geocaching` prek klica `servleta` s parametri `GET`. Podamo mu lahko parametre:

lat

koordinata latitude

lon

koordinata longitude

distance

oddaljenost od koordinat latitude in longitude, do katere še iščemo zabojnike

placedby

postavljavec zabojnika Geocaching

cachename

ime zabojnika Geocaching

cachetype

tip zabojnika Geocaching (Traditional Cache, Multi-cache, Unknown Cache, Webcam Cache, Earthcache, Letterbox Hybrid, Wherigo Cache, Event Cache)

container

tip zabojnika (Regular, Micro, Other, Small, Not chosen, Large)

Primer klica servleta in vrnjenega rezultata je prikazan spodaj. `Servlet` v odgovor na tak klic vrne podatke v obliki `JSON`. Vrne tabelo vseh točk, ki ustrezajo iskanim kriterijem.

```
http://kolesar-android.appspot.com/geocaching?lat=46.2219316&lon
    =14.5248098&distance=2000&cachetype=Traditional%20Cache
```


```
[{
  "idTocke":0,
  "idZemljevida":1,
  "latitude":46.2134,
  "longitude":14.54035,
  "cas":"2012-01-12 08:00:00",
  "ime":"Planinski raj",
  "url":"http://www.geocaching.com/seek/cache_details.aspx?guid
     \u003d3d9b3464-085d-41ac-acae-0e82f520fd87",
  "postavil":"mirobori",
```

```
  "tipZaklada":"Traditional Cache",
  "zahtevnost":1.5,
  "teren":1.5,
  "opis":"",
  "namig":"Dez ga ne zmoci/Not wetted by rain",
  "najden":5,
  "zabojnik":"Small"
}]
```

Podatki, ki jih `servlet` vrne, se pridobijo s podatkovne baze na strežniku `App Engine`. Najprej v metodi `doGet()` pridobimo parametre, s katerimi je bil `servlet` klican. Primer pridobitve parametra:

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    {
   String placedBy = req.getParameter("placedby");
   ...
}
```

Nato sestavimo poizvedbo `SQL`, ki vsebuje prebrane parametre. Poizvedba ne upošteva razdalj med točkami, zato se v zanki sprehodimo do vseh vrnjenih točk poizvedbe in izračunamo razdaljo od začetne do trenutne točke ter neustrezne točke odstranimo. Iz podatkov ustreznih točk pa smo naredili seznam objektov v `Javi`. Te objekte smo potem s knjižnico `GSON` pretvorili v format `JSON`, ki ga vrnemo kot rezultat.

4.4.2 Google Cloud Endpoints

API za kolesarske informacijske točke

Podatke o kolesarskih informacijskih točkah smo pridobili s spletne strani `www.slovenia.info` in jih vnesli v podatkovno bazo `Google Cloud SQL`. Za dostop do teh podatkov smo naredili `API` s pomočjo tehnologije `Google Cloud Endpoints`. Pri razvoju `API-ja` smo najprej naredili entitetni razred

z imenom `CyclingInfoPoint` in mu določili razredne atribute ter metode `getter` in `setter`. Dodali smo mu tudi oznaki `Entity` in `Id`. Del razreda `CyclingInfoPoint`:

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class CyclingInfoPoint {

   @Id
   private int idPoint;
   private int idMap;
   private int tourOperator;
   private double latitude;
   private double longitude;
   private String name;

   public int getIdPoint() {
      return idPoint;
   }
   public void setIdPoint(int idPoint) {
      this.idPoint = idPoint;
   }
}
```

Nato smo v integriranem razvojnem okolju `Eclipse` s pomočjo dodatka `Google` zgenerirali razred `Cloud Endpoint` iz našega entitnega razreda `CyclingInfoPoint`. V tem razredu se zgenerirajo privzete metode `API` za dostop do podatkov o kolesarskih informacijskih točkah. To so metode `list()`, `get()`, `insert()`, `update()` in `remove()`. Te privzete metode shranjujejo in pridobivajo podatke iz `Googlove` podatkovne shrambe `Datastore`. Ker uporabljamo podatkovno bazo `Google Cloud SQL`, smo napisali metodi `list()` in `get()` za

to bazo. Metodi smo opremili z notacijami, ki omogočajo dostop do kolesarskih informacijskih točk v podatkovni bazi `Google Cloud SQL`. Notacija `ApiMethod` pomeni, da je metoda del `API`-ja. Z atributom `name` pa določimo ime metode. Vsaki metodi `API` moramo poleg tega dodati še atributa `path` in `httpMethod`. Atribut `path` določa pot v URL-ju, prek katere dostopamo do metode. Privzeti metodi `get` se pot določi na ime razreda. V našem primeru je zato pot `cyclinginfopoint`. Atribut `httpMethod` določa, katero metodo `HTTP` uporabi strežnik za dostop do metode. Metoda `get` privzeto uporablja metodo `HTTP GET`. Metoda sprejme tudi en parameter. To je ID kolesarske informacijske točke. Metodi ga podamo tako, da v pot `URL` vnesemo še poševnico in želeni ID. Del kode metode `get`:

```
@ApiMethod(name = "getCyclingInfoPoint")
public CyclingInfoPoint getCyclingInfoPoint(@Named("id") int id) {
   CyclingInfoPoint cyclinginfopoint = new CyclingInfoPoint();

   // Povezava na Google Cloud SQL bazo in pridobitev podatkov o
      tocki z iskanim id-jem

   return cyclinginfopoint;
}
```

Ko smo imeli oba razreda implementirana, smo prek dodatka `Google` iz okolja `Eclipse` zagnali izdelavo knjižnice `Endpoint`. Prek te knjižnice lahko potem dostopamo do metod `API-ja`. `API` nam vrača podatke v obliki `JSON`. Primer klica in vrnjenih podatkov `API-ja`:

```
https://kolesar-android.appspot.com/\_ah/api/
    cyclinginfopointendpoint/v1/cyclinginfopoint/1
```


```
{
 "idPoint": 1,
 "idMap": 1,
 "tourOperator": 6117,
```

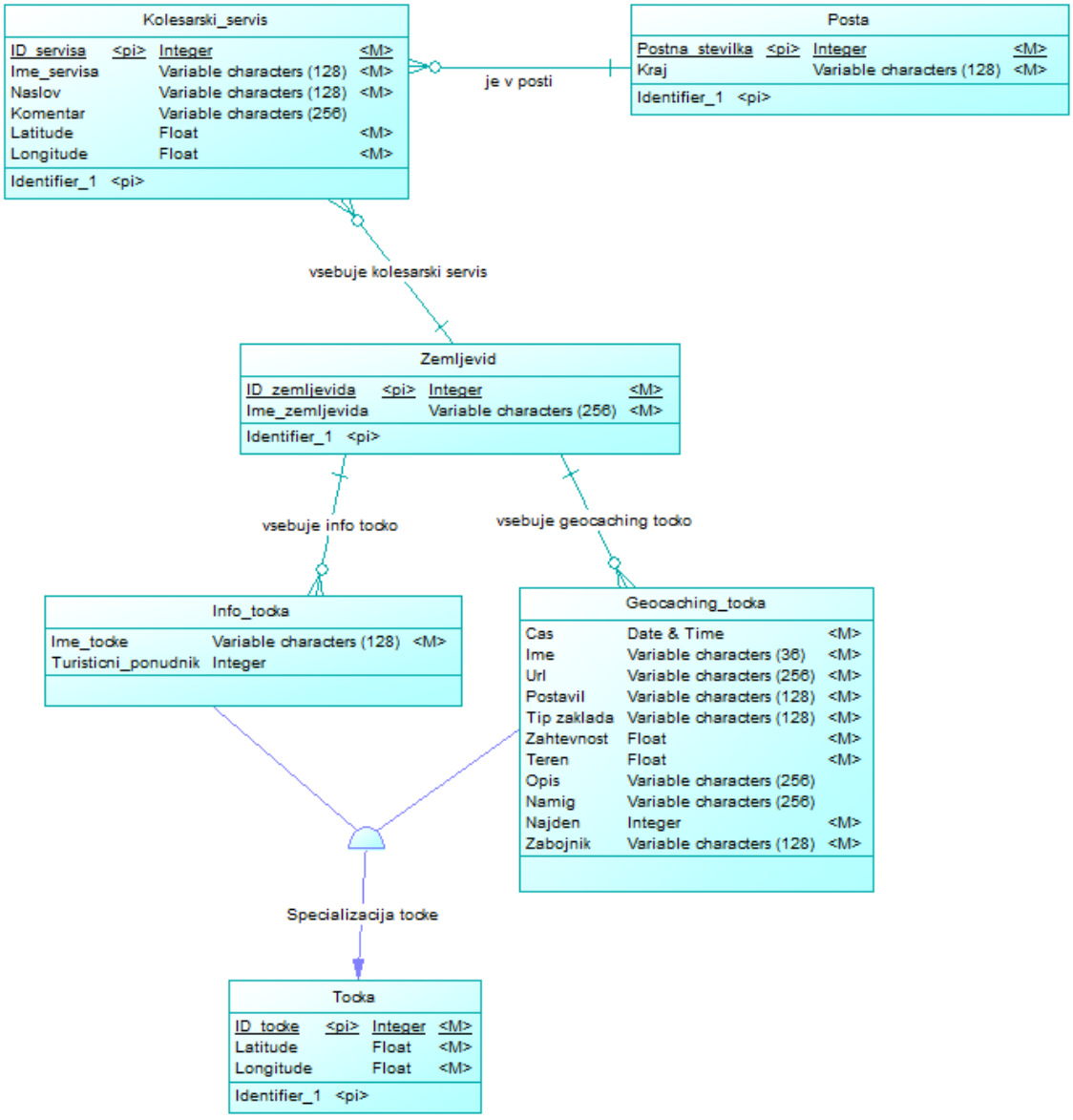
```
"latitude": 46.05099868774414,
"longitude": 14.51039981842041,
"name": "Slovenski turisticni informacijski center",
"kind": "cyclinginfopointendpoint#resourcesItem",
"etag": "\"Wewn3h3tlyn8nce8OgCz9ZTSLtE/YGzHcmjDMg7eEZP-ew9
   ogpFZiUU\""
}
```


4.4.3 Podatkovna baza

Podatkovna baza, ki jo uporabljamo na spletnem strežniku, je `Google Cloud SQL`. To je relacijska podatkovna baza, ki uporablja sistem za upravljanje podatkovne baze `MySQL`. Za naš informacijski sistem smo najprej načrtovali zgradbo baze. To smo naredili prek konceptualnega modela, ki je prikazan na sliki 4.9. Osrednja entiteta modela je `Zemljevid`. Za vsako področje, na katerem bi delovala aplikacija, se ustvari ta entiteta in se ji določi ime. V našem primeru smo naredili entiteto z imenom Slovenija. Vsaka entiteta `Zemljevid` pa lahko vsebuje še nič ali več entitet tipa `Info_tocka`, `Geocaching_tocka` in `Kolesarski_servis`. Entiteti `Info_tocka` in `Geocaching_tocka` sta specializaciji entitete `Tocka`, ker imata skupna atributa, to sta koordinati `latitude` in `longitude`. Uporabljata se za shranjevanje točk `Geocaching` s spletne strani `geocaching.com` in kolesarskih informacijskih točk s strani `slovenia.info`. Entiteta `Kolesarski_servis` pa služi za shranjevanje kolesarskih servisov, ki jih uporabniki dodajo v aplikaciji. Povezana je še z entiteto `Pošta`. Ta podatek se uporablja pri pretvorbi naslova kolesarskega servisa v geografske koordinate.

Ustvarjanje podatkovne baze

Bazo `Google Cloud SQL` ustvarimo prek konzole `Google API`, ki je dostopna na naslovu `https://code.google.com/apis/console`. Tu naredimo nov projekt ali pa uporabimo že obstoječega. Ko je projekt pripravljen, v



Slika 4.9: Konceptualni model podatkovne baze

meniju izberemo `Google Cloud SQL` in ustvarimo novo instanco. Tu se je treba odločiti za tip instance, in sicer glede na to, za kaj jo bomo uporabljali. Aplikacije, ki jim dovolimo dostop do instance, določimo v meniju `Authorized applications`. Tu navedemo enolične identifikatorje aplikacij `Google App Engine`. [23]

Povezava na podatkovno bazo

Za povezavo na bazo `Cloud SQL` moramo uvoziti pravilni gonilnik. Gonilnik se nahaja v paketu `com.google.appengine.api.rdms.AppEngineDriver`. Ko imamo gonilnik, se prek njega lahko povežemo na bazo tako, da navedemo naslov do instance, ki smo jo prej ustvarili. Primer povezave na bazo `Cloud SQL`:

```
import com.google.appengine.api.rdbms.AppEngineDriver;

Connection conn = null;

try {
   DriverManager.registerDriver(new AppEngineDriver());
   conn = DriverManager.getConnection("jdbc:google:rdbms://android-
       kolesar:android-kolesar/kolesarski_zemljevid?
       characterEncoding=UTF-8");
} catch(SQLException ex) {
   ex.printStackTrace();
} finally {
   if(conn != null) {
      try {
         conn.close();
      } catch (SQLException ignore) { }
   }
}
```

Poglavje 5

Sklepne ugotovitve

V diplomskem delu smo razvili mobilno aplikacijo za platformo Android. Aplikacija omogoča beleženje in pregled kolesarskih poti ter pregled kolesarskih informacijskih točk, kolesarskih servisov in zakladov `Geocaching`. Za pridobivanje podatkov na mobilno napravo smo razvili spletni strežnik na platformi `Google App Engine`. Tu so bili razviti `servleti`, ki omogočajo dostop do shranjenih podatkov v podatkovni bazi `Google Cloud SQL`.

Pri razvoju aplikacije smo si najprej ogledali platformo `Android`. Nadgradili smo znanje, ki smo ga imeli o razvoju v tem okolju. Spoznali smo nove komponente, ki jih do sedaj še nismo uporabili. Razvoj aktivnosti za `Android` je potekal tekoče, saj smo na tem področju že imeli izkušnje, ki smo jih lahko uporabili pri izdelavi te aplikacije. Več časa smo morali posvetiti povezavi aplikacije s spletnim strežnikom, saj do sedaj še nismo delali s platformo `Google App Engine`. Pri izdelavi tega dela smo si pomagali z dokumentacijo in testnimi primeri, ki so na voljo na spletni strani podjetja `Google`. Dokumentacija nam je bila v veliko pomoč, saj na primerih pokaže, kako se dela z opisano tehnologijo. Tudi povezave s skupnostjo `Geocaching` ni bilo težko razviti, ker so podatki o zakladih shranjeni v lepo strukturirani obliki.

Aplikacijo bi se dalo v prihodnosti še izboljšati. Uporabniški vmesnik aplikacije bi se dalo izboljšati tako, da bi izboljšali izgled komponent in na-

redili vmesnik še enostavnejši za uporabo. Prav tako bi se dalo izboljšati prikaz podatkov o aktivnostih. Dodali bi lahko različne grafe, ki bi uporabniku pregledneje prikazali shranjene podatke. Ker je bila aplikacija razvita tako, da podatke shranjuje na spletni strežnik, bi jo lahko podprli tudi s spletno stranjo. Tako bi lahko uporabniki pregledovali podatke tudi prek računalnika.

Slike

Literatura

[1] M. Gargenta. Learning Android. 0'Reilly, Sebastopol, 2011

[2] D. Sanderson. Programming App Engine. O'Reilly, Sebastopol, 2013

[3] K. Jamsa. Cloud Computing. Jones & Bartlett Learning, Burlington, 2013

[4] B. Sosinsky. Cloud computing bible. Wiley Publishing Inc., Indianapolis, 2011

[5] W. Lee. Beginning Android Application Development. Wiley Publishing Inc., Indianapolis, 2011

[6] Wikipedija. Operacijski sistem Android. Dostopno na: http://en.wikipedia.org/wiki/Android_(operating_system). April 2013

[7] Osnove Android aplikacij. Dostopno na: http://developer.android.com/guide/components/fundamentals.html. April 2013

[8] Wikipedija. Razvoj programske opreme Android. Dostopno na: http://en.wikipedia.org/wiki/Android_software_development. Maj 2013

[9] Aktivnosti Android aplikacije. Dostopno na: http://developer.android.com/guide/components/activities.html. Maj 2013

[10] Storitve. Dostopno na:
http://developer.android.com/guide/components/services.html. Maj 2013

[11] Google API. Dostopno na:
https://developers.google.com/android/add-ons/google-apis. Maj 2013

[12] Google Maps API. Dostopno na:
https://developers.google.com/maps/documentation/android/intro. Maj 2013

[13] Kolesarske informacijske točke Dostopno na:
http://www.slovenia.info/si/Kolesarske-informacijske-točke/search-selected.htm?kolesarske_informacijske_tocke. Junij 2013

[14] Google App Engine. Dostopno na:
http://en.wikipedia.org/wiki/Google_App_Engine. Maj 2013

[15] Google Cloud Endpoints. Dostopno na:
https://developers.google.com/appengine/docs/java/endpoints. Julij 2013

[16] Adding and Annotating Entities for Endpoints. Dostopno na:
https://developers.google.com/eclipse/docs/endpoints-addentities. Julij 2013

[17] Geocaching. Dostopno na:
http://en.wikipedia.org/wiki/Geocaching. Maj 2013

[18] Google Geocoding API. Dostopno na:
https://developers.google.com/maps/documentation/geocoding. Avgust 2013

[19] Java Servlet. Dostopno na:
http://en.wikipedia.org/wiki/Java_Servlet. Avgust 2013

[20] Oracle - Java Servlet. Dostopno na:
http://docs.oracle.com/javaee/6/api/javax/servlet/Servlet.html. Avgust 2013

[21] Google Cloud SQL. Dostopno na:
https://developers.google.com/cloud-sql/. Julij 2013

[22] O Google Cloud SQL. Dostopno na:
https://developers.google.com/cloud-sql/docs/introduction. Julij 2013

[23] Povezava na Google Cloud SQL. Dostopno na:
https://developers.google.com/appengine/docs/java/cloud-sql/. Julij 2013

[24] Cloud computing. Dostopno na:
http://en.wikipedia.org/wiki/Cloud_computing. Avgust 2013