Univerza v Mariboru
Fakulteta za elektrotehniko,
računalništvo in informatiko

Miha Ravber

KOORDINIRAN PRENOS GPS PODATKOV MOBILNE NAPRAVE DO STREŽNIKA

Diplomsko delo

Maribor, september 2012

KOORDINIRAN PRENOS GPS PODATKOV MOBILNE NAPRAVE DO STREŽNIKA

Diplomsko delo

Študent:	Miha Ravber
Študijski program:	VS ŠP Računalništvo in informacijske tehnologije
Mentor:	prof. dr. Janez Brest
Somentor:	doc. dr. Borko Bošković

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

Številka: BRIT.26
Datum in kraj: 31. 05. 2015, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 1/2010)

SKLEP O DIPLOMSKEM DELU

1. **Mihi Ravberju,** študentu visokošolskega strokovnega študijskega programa RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE, se dovoljuje izdelati diplomsko delo pri predmetu Integracijsko programiranje.

2. **MENTOR:** **red. prof. dr. Janez Brest**
   SOMENTOR: **asist. dr. Borko Bošković**

3. **Naslov diplomskega dela:**
   KOORDINIRAN PRENOS GPS PODATKOV MOBILNE NAPRAVE DO STREŽNIKA

4. **Naslov diplomskega dela v angleškem jeziku:**
   CONTROLLED GPS DATA TRANSFER FROM MOBILE DEVICES TO SERVER

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela". Skladno s 7. členom *Pravilnika o postopku priprave in zagovora diplomskega dela na dodiplomskem študiju,* je bilo odobreno podaljšanje roka za oddajo diplomskega dela do 06. 05. 2013. Diplomsko delo študent odda v treh izvodih (dva vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:

Obvestiti:
- kandidata,
- mentorja,
- somentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju prof. dr. Janezu Brestu za pomoč in vodenje pri opravljanju diplomskega dela. Prav tako se zahvaljujem somentorju doc. dr. Borku Boškoviću.

Posebna zahvala velja staršem, ki so mi omogočili študij.

KOORDINIRAN PRENOS GPS PODATKOV MOBILNE NAPRAVE DO STREŽNIKA

Ključne besede: operacijski sistem Android, mobilna aplikacija, prenos podatkov GPS, model odjemalec-strežnik.

UDK: 004.43:004.738.5(043.2)

Povzetek

Diplomska naloga opisuje razvoj Android aplikacije in strežnika ter vso uporabljeno tehnologijo. Predstavljen je koordiniran prenos GPS (ang. Global Positioning System) podatkov iz mobilne naprave do strežnika. Aplikacija pridobi GPS podatke naprave in jih pošlje strežniku, ki te podatke prejema. Strežnik bo tudi krmilil hitrost pošiljanja GPS podatkov s telefona na strežnik. Implementiran sistem tako omogoča uporabo GPS podatkov v različne namene.

CONTROLLED GPS DATA TRANSFER FROM MOBILE DEVICES TO SERVER

Key words: Android operating system, mobile application, GPS data transfer, client-server model.

UDK: 004.43:004.738.5(043.2)

Abstract

The thesis describes the development of an Android application and server, and all the technology used. Described is a coordinated transfer of GPS (Global Positioning System) data from mobile devices to the server. The application obtains GPS data from the device and sends it to a server that receives this data. The server also controls the speed at which the GPS data will be sent from the device to the server. The implemented system enables the use of GPS data for different purposes.

VSEBINA

SEZNAM SLIK

SEZNAM TABEL

UPORABLJENE KRATICE

ADB – Android Debug Bridge

ADT – Android Development Tools

AJAX – Asynchronous JavaScript and XML

AOSP – Android Open Source Project

API – Application Programming Interface

ASP – Active Server Pages

ASF – Apache Software Foundation

AVD – Android Virtual Device

CDT – C/C++ Development Tooling

CPU – Central Processing Unit

CSS – Cascading Style Sheets

DDMS – Dalvik Debug Monitor Server

DHTML – Dynamic HTML

DOM – Document Object Model

GPS – Global Positioning System

GSM – Global System for Mobile Communications

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

IDE – Integrated Development Environment

IP – Internet Protocol

IPC – Inter-process Communication

JDT – Java Development Tools

JIT – Just-in-time

JSON – JavaScript Object Notation

JSP – Java Server Pages

JVM – Java Virtual Machine

Java EE – Java Enterprise Edition

LAMP – Linux, Apache, MySQL and Perl/PHP/Python

LCD – Liquid Crystal Display

MAC – Media Access Control

MD5 – Message-Digest algorithm 5

OHA – Open Handset Alliance

OS – Operating System

PDT – PHP Development Tools

PHP – Hypertext Preprocessor

SD – Secure Digital

SDK – Software Development Kit

SGL – Skia Graphics Library

SMS – Short Message Service

SSL – Secure Sockets Layer

SQL – Structured Query Language

UI – User Interface

URL – Uniform Resource Locator

USB – Universal Serial Bus

UUID – Universally Unique Identifier

VM – Virtual Machine

WAR – Web ARchive

WiFi – Wireless Fidelity

XML – Extensible Markup Language

1 UVOD

Sedaj so zelo razburljivi časi za razvijalce mobilnih naprav, saj so le-te zmeraj bolj popularne. Vedno več potrošnikov se odloča za nakup pametnega telefona. Te telefone odlikujejo komponente, kot so pospeškometer, GPS, zmogljiv procesor ter zaslon, občutljiv na dotik [1]. Velika značilnost pametnih telefonov je tudi zmožnost upravljanja več aplikacij hkrati, kar omogoča operacijski sistem. Med popularne operacijske sisteme spada Android, ki je ugledal luč sveta leta 2007 in je eden izmed najhitreje rastočih platform. Delež mobilnih telefonov z nameščenim OS (ang. *Operating System*) Android v letu 2011 je okoli 14,5% in še raste. Tako je Android v letu 2011 prvič prehitel Applov iOS, vendar je Nokia še vedno vodilna. Eden od razlogov za popularnost OS je velika izbira različnih aplikacij, ki jih lahko najdemo na Androidovi »tržnici«. Februarja 2010 je bilo zabeleženih prek 150.000 aplikacij za Android. Tako smo se tudi mi odločili izdelati aplikacijo za ta operacijski sistem [13].

Glavni namen diplomske naloge je spoznati operacijski sistem Android in izdelati aplikacijo. Aplikacija bo pridobila točne koordinate mobilne naprave in jih poslala na strežnik. To nam bo omogočalo sledenje naprave oz. uporabnika in uporabo v različnih drugih aplikacijah. Na strežniku bo spletna stran, ki bo prebrala podatke iz podatkovne baze in jih prikazala v tabeli. Omogočala bo tudi nastavitev časovnega intervala zajemanja koordinat položaja mobilne naprave. Časovni interval bo imel velik vpliv na porabo baterije. Pri manjšem intervalu bomo dobili pozicijo naprave večkrat in s tem se bo tudi povečala poraba baterije, saj mora aplikacija v vsakem intervalu na novo pridobiti pozicijo ter jo poslati na strežnik. Pri razvoju aplikacije moramo upoštevati dejstvo, da morda ni signala ali internetne povezljivosti. Prav tako ne smemo zanemariti, da lahko koordinate odstopajo od dejanskih zaradi slabega signala.

1.1 Struktura diplomskega dela

V drugem poglavju je predstavljena tehnologija, uporabljena pri izdelavi aplikacije in strežnika. Podrobneje je opisan operacijski sistem in njegova arhitektura. Tretje poglavje je razdeljeno na dve podpoglavji, in sicer na razvoj Android aplikacije in strežnika. Obe poglavji podrobno opisujeta posamezne dele aplikacije in način njihovega delovanja.

V četrtem poglavju je podano preizkušanje aplikacije na emulatorju in mobilni napravi ter obremenitveno preizkušanje strežnika.

2 OPIS UPORABLJENE TEHNOLOGIJE

2.1 OPERACIJSKI SISTEM ANDROID

Android je operacijski sistem, ki vključuje programsko opremo, namenjeno mobilnim napravam ter vmesne in ključne aplikacije [12]. Google Inc. je kupil začetnega razvijalca programske opreme Android Inc. v letu 2005. Androidov mobilni operacijski sistem temelji na jedru Linux. Google in ostali člani Open Handset Alliance (OHA) so sodelovali pri razvoju Androida in njegovi izdaji. Android Open Source Project (AOSP) je zadolžen za vzdrževanje in nadaljnji razvoj Androida [13].

Androidov paket odprtokodne programske opreme je sestavljen iz aplikacij programskega jezika Java. Aplikacije tečejo na osnovi objektno orientiranega aplikacijskega ogrodja, vključno z jedrnimi knjižnicami Jave na virtualnem stroju Dalvik. Dalvik vsebuje prevajalnik JIT (ang. *Just-in-time*), to je metoda za izboljšanje izvajanja programov. Paket vsebuje tudi knjižnice napisane v programskem jeziku C: OpenCore medijsko ogrodje, podatkovno bazo SQLite, API (ang. *Application Programming Interface*) za OpenGL ES 2.0 3D grafiko, načrtovalno orodje WebKit, grafično orodje SGL (ang. *Skia Graphics Library*), SSL (ang. *Secure Sockets Layer*), in Bionic libc. Operacijski sistem, vključno z jedrom Linux, je sestavljen iz približno 12 milijonov vrstic kode, vključno s 3 milijoni vrstic XML (ang. *Extensible Markup Language*), 2,8 milijona vrstic programskega jezika C, 2,1 milijona vrstic programskega jezika Java, in 1,75 milijona vrstic programskega jezika C++ [10].

2.1.1 ARHITEKTURA

2.1.1.1 Aplikacijski nivo

Vse aplikacije so zgrajene na aplikacijskem nivoju z uporabo istih API. Android vključuje nekaj osnovnih aplikacij, kot so odjemalec za elektronsko pošto, koledar, brskalnik, zemljevid, aplikacija za klicanje itd. Vse te aplikacije so napisane v programskem jeziku Java [6], [7].

2.1.1.2 Aplikacijsko ogrodje

Nudi razrede, potrebne za razvoj aplikacij, in abstrakcijo med dostopom do strojne opreme. Glavna knjižnica vključuje telefonijo, ponudnika vsebin (podatkov), sredstva, lokacije in UI (ang. *User Interface*). Arhitektura je oblikovana za poenostavitev ponovne uporabe programov. Aplikacijsko ogrodje si lahko predstavljamo kot nabor osnovnih orodij, s katerimi lahko razvijalec gradi bolj kompleksna orodja [1], [7].

2.1.1.3 Knjižnice

Ta nivo je sestavljen iz knjižnic, napisanih v programskih jezikih C in C++, ki jih uporabljajo različne komponente sistema Android. Te knjižnice povedo napravi, kako ravnati z različnimi vrstami podatkov in so razvijalcem dostopne preko Androidovega aplikacijskega ogrodja. Nekatere knjižnice, ki jih vključuje so: medijske knjižnice, knjižnica za grafiko, 3D knjižnice, SQLite, knjižnico spletnega brskalnika itd. [7].

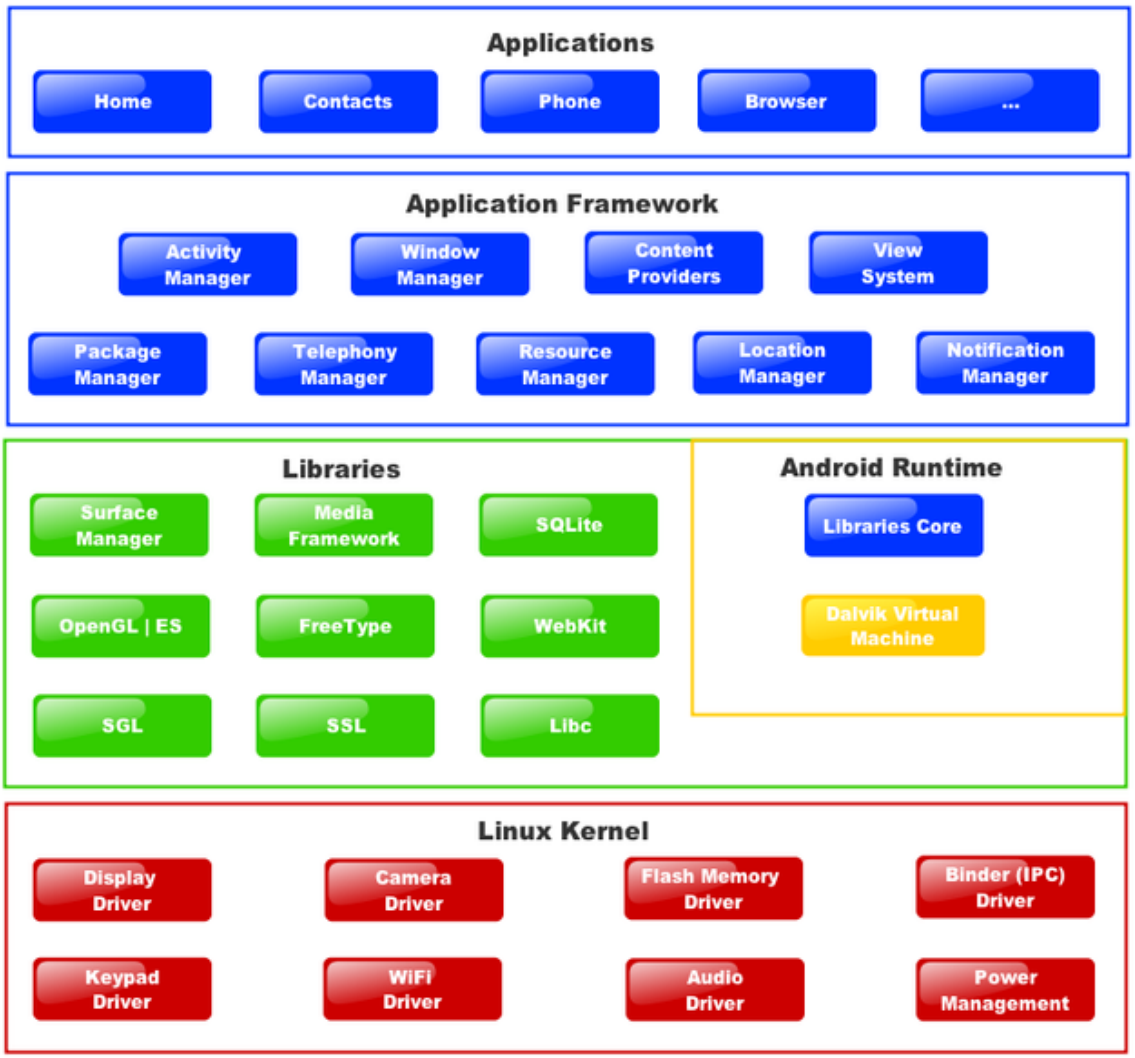
2.1.1.4 Izvajalno okolje Android

Android vključuje nabor osnovnih knjižnic, ki zagotavljajo večino funkcij na voljo v osnovnih knjižnicah jezika Java. Vsaka aplikacija izvaja svoj proces, s svojim primerkom virtualnega stroja Dalvik. Le-ta je bil napisan tako, da lahko naprava učinkovito izvaja več VM (ang. *Virtual Machine*) hkrati. Dalvik izvaja datoteke v formatu Dalvik Executable (.Dex), ki je optimiziran za minimalno porabo delovnega pomnilnika [7].

2.1.1.5 Jedro Linux

Ta nivo vsebuje programe za upravljanje s pomnilnikom, varnostne nastavitve, programsko opremo in več gonilnikov za strojno opremo, dostop do datotečnega sistema, omrežje in medprocesno komunikacijo. Jedro deluje tudi kot abstraktna plast med strojno opremo in ostalo programsko opremo [7].

Na sliki 2.1 je prikazana celotna arhitektura operacijskega sistema Android.



Slika 2.1: Arhitektura operacijskega sistema Android [6].

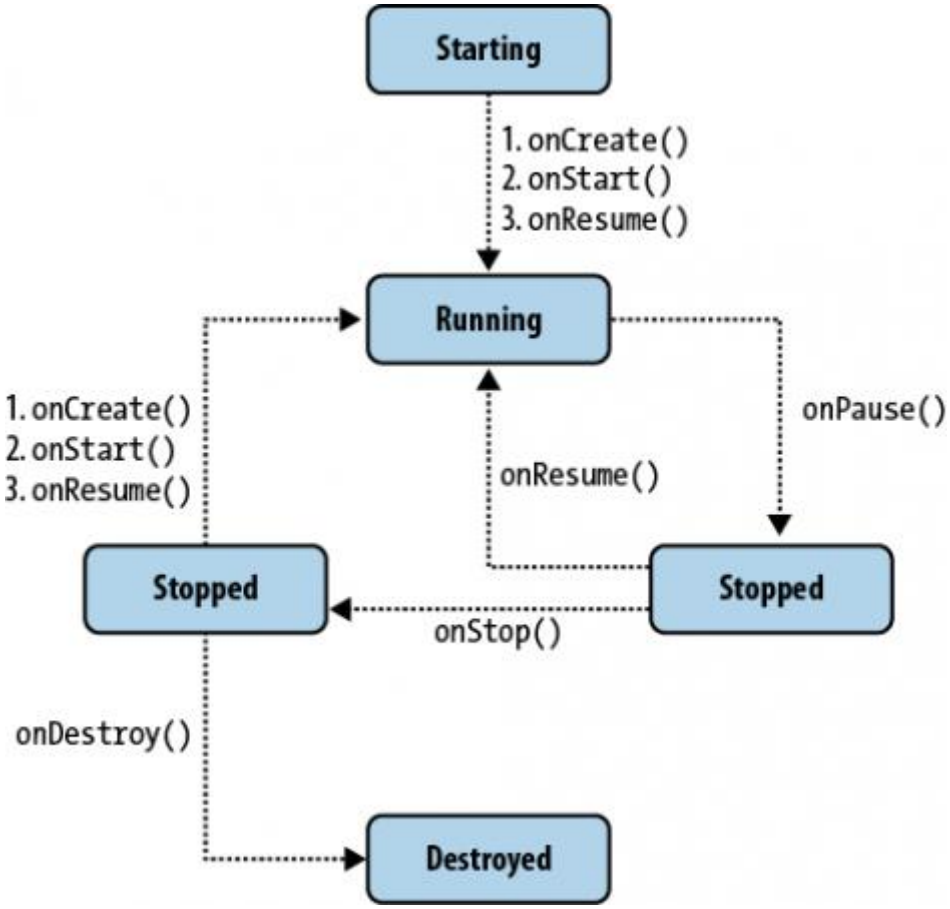
2.1.1.6 Aktivnosti

Aktivnost je komponenta aplikacije, ki ima vmesnik za interakcijo z uporabnikom. Med te dejavnosti spada: klicanje, fotografiranje, pošiljanje e-pošte, ogled zemljevida itd. Vsaki aktivnosti je podano okno, v katerem se izriše določen uporabniški vmesnik. Okno običajno zapolni zaslon, lahko pa je tudi manjše, nad ostalimi okni.

Aplikacija je običajno sestavljena iz več aktivnosti, ki so med seboj povezane. Značilno je, da je ena aktivnost v uporabi, opredeljena kot "glavna" aktivnost. Ta se prikaže, ko zaženemo aplikacijo. Vsaka aktivnost lahko nato zažene druge aktivnosti za izvajanje različnih akcij. Vsakič, ko se začne nova aktivnost, je prejšnja aktivnost ustavljena, vendar pa sistem ohranja aktivnost na skladu (back stack). Ko se začne nova aktivnost, je dana na

sklad in je v ospredju. Ko je uporabnik končal s trenutno aktivnostjo ter pritisne tipko nazaj, jo OS pobere iz sklada (in uniči) in predhodna aktivnost se nadaljuje [14].

Na sliki 2.2 je prikazan življenjski cikel aktivnosti.



Slika 2.2 : Življenjski cikel aktivnosti [15].

2.1.1.7 Servisi

Servis je sestavni del aplikacije, ki lahko izvaja dolgotrajne operacije v ozadju in nima uporabniškega vmesnika. Aplikacijska komponenta lahko požene servis in se bo še naprej izvajala v ozadju, tudi če uporabnik preklopi na drug program. Poleg tega se lahko del veže na servis za komunikacijo in celo opravljajo medprocesne komunikacije (IPC). Na primer, servis lahko opravlja omrežne transakcije, igra glasbo, izvaja vhodno/izhodne operacije datotek, ali pa poskrbi za interakcijo s ponudnikom vsebine [16].

2.1.1.8 Ponudniki vsebin

Ponudniki vsebin shranjujejo in pridobivajo podatke, kateri so dostopni vsem aplikacijam in so edini način za izmenjavo podatkov med aplikacijami; ne obstaja skupni skladiščni prostor, do katerega bi lahko dostopali vsi Android paketi.

Android vsebuje več ponudnikov vsebin za splošne vrste podatkov (avdio, video, slike, osebne podatke, itd.). Nekatere izmed njih lahko vidimo navedene v android.provider paketu. Iz ponudnikov lahko tudi dobimo podatke, ki jih vsebujejo (čeprav je za nekatere potrebno pridobiti ustrezno dovoljenje za branje podatkov).

Če želimo, da bi bili naši podatki na voljo javnosti, imamo dve možnosti: lahko ustvarimo svojega ponudnika vsebine (podrazred ContentProvider), lahko pa dodamo podatke obstoječemu ponudniku, če obstaja en, ki nadzira iste vrste podatkov in imamo vse pravice za pisanje vanj [28].

2.1.1.9 Sprejemniki

Sprejemniki so objekti, izpeljani iz vmesnika BroadcastReceiver in so namenjeni za sprejemanje dogodkov, kot so vklop telefona, prejeti klic, novo sporočilo, odklepanje telefona. Ko se dogodek sproži, sprejemnik običajno reagira tako, da požene aktivnost, servis ali katero drugo komponento. Vmesnik BroadcastReceiver ima samo eno metodo: onReceive().

Sprejemnik je aktiven le toliko časa, kolikor ga potrebuje proces onReceive(). Ko se metoda zaključi, je sprejemnik odstranjen iz pomnilnika in ga ni možno ponovno uporabiti. To pomeni, da so sprejemniki nekoliko omejeni v tem, kaj lahko storijo, predvsem da bi se izognili nečemu, kar vključuje vse vrste povratnih klicev. Sprejemniki se, na primer, ne morejo povezati s servisom, in ne morejo odpreti pogovornega okna [35].

2.1.1.10 Manifest

Temelj za katerokoli aplikacijo je datoteka manifest: AndroidManifest.xml v korenu projekta. Tu lahko določimo, kaj je v naši aplikaciji - aktivnosti, servisi itn. V tej datoteki določimo tudi, kako se te komponente pritrdijo v sistemu, na primer, lahko navedemo katero aktivnost (ali aktivnosti) je treba prikazati na glavnem meniju naprave (zagon).

Ko ustvarimo našo aplikacijo, se nam generira manifest. Za enostavno uporabo, ki ponuja eno samo aktivnost in nič drugega, bo samodejno ustvarjen manifest verjetno zadoščal. Zahteval bo morda nekaj manjših sprememb [2].

2.1.2 Emulator

Android SDK (ang. *Software Development Kit*) vključuje emulator mobilne naprave - virtualno mobilno napravo, ki deluje na računalniku (slika 2.3). Emulator omogoča, da razvijamo in preizkušamo Android aplikacijo brez mobilne naprave.

Emulator zažene okno na računalniku, ki izgleda kot Android telefon. Prvi zagon emulatorja je počasen, celo na zelo zmogljivih računalnikih. S konfiguriranjem emulatorja poskušamo posnemati številne vidike dejanske Android naprave. Vendar nekatere funkcije (kot so senzorji in avdio/video) niso enake. Emulator je treba smatrati, kot uporaben način za potrditev osnovne funkcionalnosti za naprave, ki niso na voljo uporabniku [4].

Emulator omogoča enako interakcijo kot dejanska mobilna naprava. Razlika je v tem, da uporabljamo kazalec miške za dotik na zaslon. Nekatere tipke na tipkovnici emulatorja se sklicujejo na gumbe mobilne naprave [29].

Slika 2.3: Android emulator.

2.1.2.1 Android Debug Bridge

Android Debug Bridge (ADB) je vsestransko orodje ukazne vrstice, ki omogoča komunikacijo z emulatorjem ali s povezano Android napravo. To je program tipa odjemalec-strežnik, ki vključuje dve komponenti:

- odjemalca, ki deluje na razvojnem stroju. Lahko se sklicujemo na odjemalca iz lupine z izdajo ukazov ADB. Ostala orodja Android, kot so vtičnik ADT (ang. *Android Development Tools*) in DDMS (ang. *Dalvik Debug Monitor Server*) tudi ustvarjajo odjemalce ADB, in
- strežnik, ki teče kot proces v ozadju, na razvojnem stroju. Strežnik upravlja komunikacije med odjemalcem in ADB daemonom (proces v ozadju), ki teče na emulatorju ali napravi [18].

2.1.3 Digitalni podpis

Sistem Android zahteva, da so vse nameščene aplikacije digitalno podpisane s potrdilom, katerega zasebni ključ je v lasti razvijalca aplikacije. Sistem Android uporablja potrdilo kot sredstvo za prepoznavanje avtorja aplikacije in pri vzpostavljanju odnosov zaupanja med aplikacijami. Potrdilo se ne uporablja za nadzor nameščanja aplikacij. Overiteljem ni potrebno podpisati potrdila, povsem dopustno je, in tipično za Android aplikacije, da uporabljajo samopodpisana potrdila [30].

2.2 Razvojno okolje Eclipse

Za razvoj Android aplikacije smo uporabili klasični Eclipse, medtem ko smo strežniški del razvijali v Eclipse IDE (ang. *Integrated Development Environment*) za *Java* EE (ang. *Enterprise Edition*) razvijalce. Ker Eclipse sam po sebi ne vključuje vsa orodja za razvoj Android aplikacij, smo morali namestiti vtičnik ADT in orodja SDK.

Eclipse je večjezično razvojno okolje programske opreme, ki vključuje IDE in razširljiv sistem za vtičnike. Večina programa je napisana v programskem jeziku Java. Lahko se uporablja za razvoj aplikacij v Javi, in s pomočjo različnih vtičnikov tudi v drugih programskih jezikih, vključno z Ada, C, C++, COBOL, Haskell, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy, Android in Scheme. Prav tako se lahko uporablja za razvoj paketov programa Mathematica. Razvojno okolje Eclipse vključuje JDT (ang. *Java Development Tools*) za Javo, Eclipse CDT (ang. *C/C++ Development Tooling*) in Eclipse PDT (ang. *PHP Development Tools*) [37].

2.3 Programsko okolje Java

Java je programski jezik, ki ga je razvil James Gosling pri podjetju Sun Microsystems (podružnica Oracle Corporation) in je bil izdan v letu 1995. Ogromen delež sintakse jezika izhaja iz jezikov C in C++, vendar ima preprostejši objektni model in manj objektov v zbirnem jeziku. Java aplikacije so običajno sestavljene tako, da bytecode (oblika navodil, ki jih izvaja virtualni stroj Java) lahko deluje na kateremkoli virtualnem stroju Java (JVM), ne glede na arhitekturo računalnika. Java je splošno namenski, sočasni, objektno orientiran

jezik, ki je posebej zasnovan, da ima čim manj odvisnosti izvajanja kot je le mogoče. Predvideno je, da naj razvijalci aplikacij, "pišejo enkrat, zaženejo vsepovsod". Java je trenutno eden izmed najbolj priljubljenih programskih jezikov v uporabi in je široko uporabljen na vseh področjih [8].

2.3.1 JavaServer Pages (JSP)

JSP je tehnologija, ki pomaga razvijalcem programske opreme razvijati dinamične spletne strani s pomočjo programskega jezika Java. JSP je bil izdan leta 1999 kot odgovor podjetja Sun na ASP (ang. *Active Server Pages*) in PHP (ang. *Hypertext Preprocessor*).

Za namestitev in zagon se uporablja strežnik Apache Tomcat. To je odprtokodna programska oprema, ki jo je razvilo podjetje Apache Software Foundation (ASF). Tomcat implementira Java servlete in JSP specifikacije ter zagotavlja "čisto Java" HTTP (ang. *Hypertext Transfer Protocol*) spletno okolje strežnika [31].

Arhitekturo JSP lahko obravnavamo kot visoko nivojsko abstrakcijo servletov. Strani JSP se naložijo na strežnik in se upravljajo s posebnim nameščenim strukturiranim paketom, imenovanim Java EE spletne aplikacije. Te aplikacije so zapakirane v datoteko s končnico .war ali .ear [9].

2.3.2 Servlet

Servlet je Java razred v Java EE, ki je v skladu s servlet API protokolom, s katerim se lahko aplikacija odziva na zahteve. Servleti niso vezani na določen protokol odjemalec-strežnik, vendar pa se najpogosteje uporabljajo s protokolom HTTP. Zato se beseda "servlet" pogosto uporablja v pomenu "HTTP servlet". Tako lahko razvijalec programske opreme uporablja servlet za dodajanje dinamične vsebine na spletni strežnik s platformo Java. Vsebina je običajno v HTML (ang. *HyperText Markup Language*), vendar so lahko podatki tudi drugačnega tipa, kot je format XML. Servleti lahko ohranijo stanje v sejnih spremenljivkah preko mnogih transakcij strežnika s HTTP piškotki, ali s prepisovanjem URL (ang. *Uniform Resource Locator*) naslova.

Servlet je objekt, ki prejme zahtevo in ustvari odgovor na to zahtevo. Osnovni paket servleta definira objekte Java, da predstavljajo zahteve in odgovore. Servlet se lahko zapakira v datoteko WAR (ang. *Web ARchive*) kot spletno aplikacijo.

Servleti se lahko generirajo avtomatsko iz JSP s prevajalnikom. Običajno vstavijo HTML v notranjost programske kode Java, JSP pa ravno obratno.

Servleti podpirajo modela zahteva/odgovor in odjemalec/strežnik ter metode HTTP, GET in POST [11].

2.3.2.1 Metodi GET in POST

Ko odjemalec vzpostavi povezavo s strežnikom in izvede HTTP zahtevo, je zahteva lahko različnih tipov, ki se imenujejo metode. Najbolj pogosto uporabljeni metodi sta GET in POST. Metoda GET je zasnovana za pridobivanje informacij (dokument, grafikon, ali rezultat poizvedb). Metoda POST se uporablja za objavo informacije (številka kreditne kartice, nekaj novih podatkov za grafikon ali informacij, ki se shranijo v bazo podatkov).

Čeprav je metoda GET namenjena za branje podatkov, lahko vključuje kot del zahteve nekaj dodatnih informacij. Te informacije opisujejo, kaj metoda želi dobiti. Prenašajo se kot zaporedje znakov, ki se dodajo URL naslovu. Dodajanje dodatne informacije v URL naslov omogoča, da lahko stran shranimo v zaznamke ali pošljemo po e-pošti kot vsak drugi spletni naslov. Ker je dolžina URL naslova omejena, je omejena tudi količina podatkov, ki jo metoda GET prenaša.

Metoda POST uporablja drugačno tehniko za pošiljanje podatkov na strežnik, ker je v nekaterih primerih potrebno poslati megabajte podatkov. POST metoda posreduje vse podatke (število znakov je omejeno na strežniku). Podatki so poslani neposredno preko vtičnice kot del zahteve HTTP. Izmenjava je nevidna za odjemalca. URL naslov se ne spremeni, zato POST metode ni mogoče ustvariti kot zaznamek ali poslati URL po e-pošti, v nekaterih primerih je celo ni možno ponovno naložiti [22], [21].

2.4 JSON

JSON (ang. *JavaScript Object Notation*) je odprti standard za izmenjavo podatkov. Izhaja iz skriptnega jezika JavaScript za predstavljanje enostavnih podatkovnih struktur in asociativnih polij, imenovanih objekti. Temelji na podmnožici programskega jezika JavaScript. JSON je tekstoven format, ki je v celoti neodvisen od jezika, z razpoznavalniki na voljo za večino jezikov. Format JSON se pogosto uporablja za serializacijo in prenos

strukturiranih podatkov prek omrežne povezave. Uporablja se predvsem za prenos podatkov med strežnikom in spletnimi aplikacijami, ki služi kot alternativa XML [19].

JSON temelji na dveh strukturah:

- zbirka parov ime/vrednost. V različnih jezikih je realizirana kot objekt, zapis, struktura, slovar, razpršena tabela ali asociativno polje; in
- urejen seznam vrednosti. V večini jezikov je realiziran kot polje, vektor, seznam ali zaporedje.

To sta univerzalni podatkovni strukturi. Praktično ju vsi moderni programski jeziki poznajo v taki ali drugačni obliki. Smiselno je, da tudi jezik za izmenjavo podatkov, ki je neodvisen od programskega jezika, temelji na teh dveh strukturah [20].

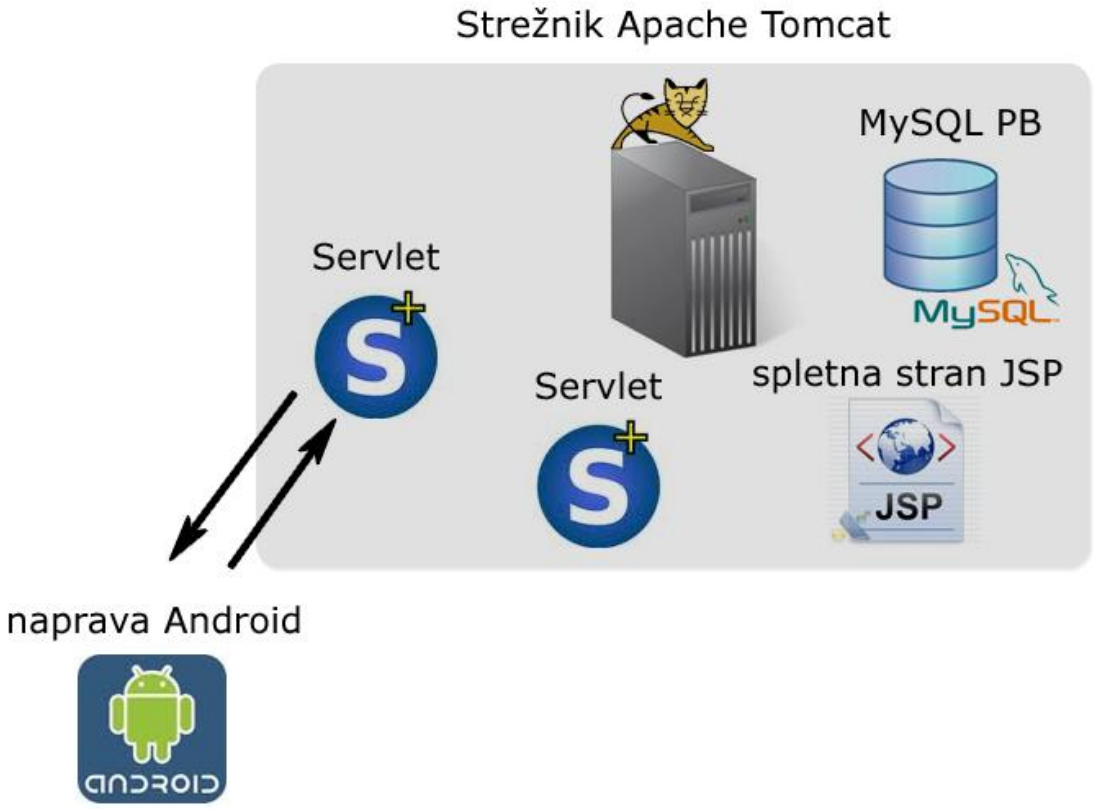
2.5 AJAX

AJAX (ang. *Asynchronous JavaScript and XML*) je skupina medsebojno povezanih tehnologij za spletni razvoj, ki se uporabljajo za ustvarjanje interaktivnih spletnih aplikacij. S tehnologijo AJAX lahko spletne aplikacije pošiljajo in pridobivajo podatke iz strežnika asinhrono v ozadju, ne da bi vplivale na prikaz in obnašanje naloženih strani. Podatki so običajno pridobljeni z uporabo objekta XMLHttpRequest. Kljub imenu AJAX, uporaba formata XML ni potrebna, prav tako ne rabi biti zahteva asinhrona.

Kot DHTML (ang. *Dynamic HTML*) in LAMP (Linux, Apache, MySQL in Perl/PHP/Python), AJAX ni ena tehnologija, temveč skupina tehnologij. AJAX uporablja kombinacijo HTML in CSS (ang. *Cascading Style Sheets*) za označevanje in slog informacije. DOM (ang. *Document Object Model*) dostopamo z JavaScript za dinamični prikaz in omogoča uporabniku interakcijo s prikazanimi informacijami. JavaScript in objekt XMLHttpRequest priskrbita metodo za asinhrono izmenjavo podatkov med brskalnikom in strežnikom, da bi se izognili popolnemu ponovnemu nalaganju strani [23].

3 PROGRAMIRANJE APLIKACIJE IN STREŽNIKA

V tem poglavju je opisan razvoj aplikacije in strežnika. Na spodnji sliki 3.1 je prikazana shema naše aplikacije in strežnika za lažje razumevanje delovanja.



Slika 3.1: Shema aplikacije in strežnika.

Kot je razvidno iz sheme, naprava Android komunicira s servletom na strežniku Tomcat. Strežnik shranjuje podatke v podatkovni bazi MySQL in jih prikazuje s pomočjo spletnih strani. Na strežniku se nahajajo servleti, ki skrbijo za branje in pisanje podatkov v podatkovni bazi.

3.1 Android aplikacija

Za delovanje naše aplikacije potrebujemo napravo z nameščenim operacijskim sistemom Android. Da bomo lahko pridobili lokacijo naprave, mora le-ta imeti vgrajeno anteno GPS. Ko aplikacija pridobi lokacijo, jo pošlje na strežnik. Za pošiljanje podatkov mora imeti naprava povezavo z internetom.

3.1.1 Dovoljenja aplikacije

Androidov varnostni model omejuje dostop do določenih storitev in funkcionalnosti. Vsaka aplikacija mora navesti, katera dovoljenja potrebuje. V času namestitve se uporabniku prikažejo dovoljenja aplikacije, preden se zaveže, da jo namesti [1]. Privzeto nima aplikacija nobenega dovoljenja, ki bi negativno vplivalo na uporabniške izkušnje ali kakršnekoli podatke na napravi. Za uporabo zaščitenih funkcij naprave, moramo vključiti v naš AndroidManifest.xml eno ali več oznak <uses-permission> [32]. Na sliki 3.2 so razvidna dovoljenja, ki jih potrebuje naša aplikacija.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

Slika 3.2: Dovoljenja zapisana v datoteki AndroidManifest.xml.

ACESS_FINE_LOCATION – Omogoča aplikaciji dostop do lokacije (GPS). To dovoljenje je ključno za našo aplikacijo, saj z njim lahko dostopamo do antene GPS preko razreda Location Manager, s tem pa tudi do geografske širine in dolžine.

INTERNET – Omogoča aplikaciji odpiranje omrežnih vtičnic. S tem dovoljenjem se bomo lahko povezali na internet, kar je potrebno za komunikacijo s strežnikom.

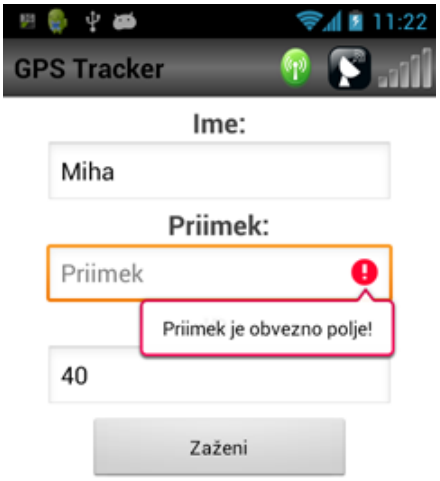
ACESS_NETWORK_STATE – Omogoča aplikaciji dostop do informacij o omrežjih. Dostop do informacij omrežja potrebujemo, da preverimo, če je naprava povezana z internetom. V primeru, da ni, obvestimo uporabnika s sporočilom na zaslonu.

WAKE_LOCK – Omogoča nadzor stanja napajanja naprave (CPU in zaslon). Pri izklopu zaslona gre procesor v spanje, da naprava prihrani električno energijo. Vendar tega ne želimo in potrebujemo dovoljenje za upravitelja napajanja. Naša aplikacija bo tekla tudi po

izklopu zaslona, vse dokler ne kličemo metode za sprostitev CPU (ang. *Central Processing Unit*) [24].

3.1.2 Uporabniški vmesnik

Aplikacija ima uporabniški vmesnik za vnos podatkov uporabnika, kateri bodo kasneje omogočali lažjo identifikacijo podatkov. Na vmesniku so tudi prikazani nekateri podatki, ki jih lahko razberemo s pomočjo sistema GPS. Začetni zaslon ima tri polja: ime, priimek in identifikacijska številka. Prvi dve polji sta nastavljeni tako, da se vnos črk prične z veliko začetnico. Polje za identifikacijo sprejme samo števila, tako uporabnik ne more vnašati črk in ostalih znakov. Preden lahko aplikacijo poženemo, morajo biti vsa polja izpolnjena. Ko so vsi podatki vneseni, se omogoči gumb zaženi in ga lahko pritisnemo. Če uporabnik pritisne gumb zaženi in vsa polja niso izpolnjena, se bo v praznem polju prikazal klicaj z opombo (slika 3.3). Aplikacija bo ob uspešnem zagonu prebrala podatke in jih poslala v glavni servis. Te podatke si bo tudi shranila v podatkovno bazo SQLite. Če so podatki že zapisani v bazi, jih aplikacija zapiše v ustrezno polje. Tako uporabniku ni potrebno ponovno vnašati podatkov ob vsakem zagonu. Ko je servis v pogonu, so na vmesniku prikazani podatki o hitrosti, višini, frekvenci zajemanja podatkov ter zemljepisna širina in dolžina. Po želji lahko uporabnik na opcijskem meniju vklopi prikaz zemljevida. Za prikaz zemljevida smo uporabili knjižnico Google Maps Android *API*. Da smo lahko uporabili to knjižnico, smo jo morali vključiti v manifest. Preden lahko uporabimo podatke *Google* Maps, se moramo registrirati za storitev Maps tako, da se strinjamo s pogoji storitve in z oskrbo MD5 prstnega odtisa potrdil, ki jih bomo uporabljali za podpisovanje naše aplikacije. Za vsak prstni odtis registriranega certifikata, nam storitev Maps ponuja API ključ - alfanumerični niz, ki enolično identificira nas in naše potrdilo. Nato shranimo API ključ v objekte MapView tako, da ko zahtevamo podatke od Google Maps, lahko strežnik določi, da smo s storitvijo registrirani [27]. Na zemljevidu se izrisuje ikona, ki prikazuje položaj naprave. Vmesnik ima tudi naslovno vrstico, ki prikazuje natančnost GPS signala in ali je naprava povezana z internetom. Natančnost signala je ponazorjena s petimi črticami, ki se obarvajo zeleno. Več kot je zelenih črtic, bolj natančen je signal. Ikona, ki prikazuje povezavo z internetom, ima dve barvi. Rdeča barva pomeni, da naprava nima povezave z internetom, medtem ko zelena pomeni, da jo ima. Če pritisnemo na ikono, nas popelje do nastavitev, kjer lahko vklopimo WiFi ali mobilni internet.



Slika 3.3: Slika vmesnika z nepopolno vnesenimi podatki.

Pri izdelavi vmesnika smo morali upoštevati, da ima naprava lahko pospeškometer, kar sproži spremembo konfiguracije pri zasuku zaslona (naprave). Če se konfiguracija naprave (kot je definirano v razredu Resources.Configuration) spremeni, potem se vse, kar prikazuje uporabniški vmesnik, posodobi. Tako se ob vsaki spremembi, kar je prikazano na zaslonu ujema s konfiguracijo. Ker je aktivnost osnovni mehanizem za interakcijo z uporabnikom, vključuje posebno podporo za obravnavo spremembe konfiguracije.

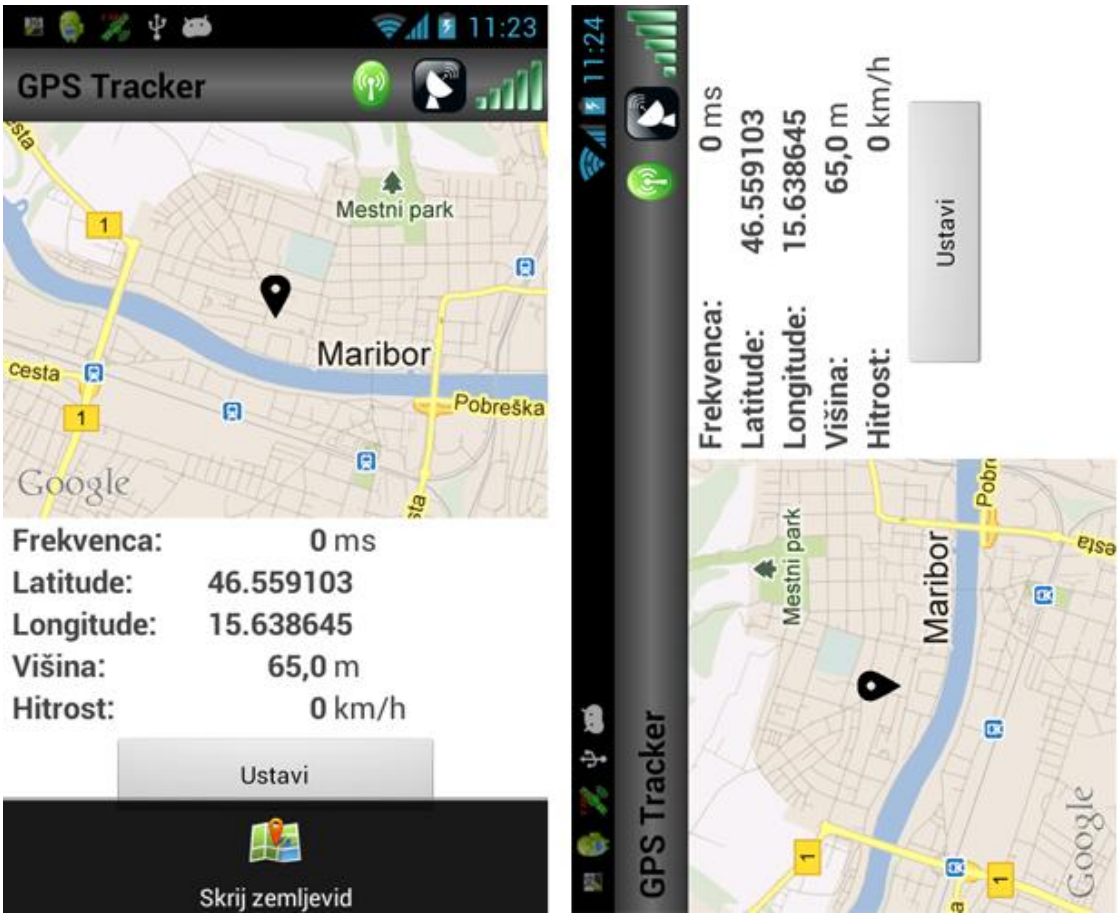
Če ne navedemo drugače, bo sprememba konfiguracije (kot so sprememba v orientaciji zaslona, jezika, vhodne naprave, itd.) uničila trenutno aktivnost. Aktivnost bo tako šla normalno skozi proces življenjskega cikla aktivnosti onPause(), onStop(), in onDestroy().

To se izvede zato, ker lahko spremenimo katerikoli vir aplikacije, vključno z načrti, ki temeljijo na katerikoli konfiguracijiski vrednosti. Tako je edini varen način za obdelavo spremembe konfiguracije, da ponovno naloži vsa sredstva. Aktivnosti vedo, kako shraniti svoje stanje in kako se ponovno ustvariti iz tega stanja. Zato je priporočeno, da se aktivnost sama po sebi ponovno požene z novo konfiguracijo [33].

V našem primeru smo obvozili ponovni zagon aktivnosti pri spremembi konfiguracije za orientacijo zaslona. To smo izvedli z atributom configChanges v datoteki manifest. Za vse vrste sprememb konfiguracije, ki smo jih določili, bodo klicale metodo onConfigurationChanged trenutne aktivnosti. Tako se aktivnosti ne bodo ponovno zagnale. Slika 3.4 prikazuje izgled naše aplikacije v ležečem in pokončnem načinu.

Pospeškometer tudi omogoča, da lahko Android aplikacija deluje v ležečem ali pokončnem načinu. To je odvisno od tega, kako uporabnik nagne zaslon naprave. Za podporo različnih orientacij so elementi vmesnika postavljeni tako, da aplikacija izgleda enako v pokončnem ali ležečem načinu. Ker smo metodo onConfigurationChanged prepisali, smo lahko izboljšali prikaz vmesnika v ležečem načinu.

Zasloni mobilnih naprav so običajno majhni, kar lahko povzroči, da so elementi vmesnika izven fizičnega zaslona naprave. Problem smo rešili z uporabo drsenja, tako da je le del informacij viden naenkrat, ostalo je na voljo s pomikanjem navzgor ali navzdol. Elementi vmesnika so v zabojniku ScrollView. Nad elementi, ki so v tem zabojniku, je omogočeno drsenje.



Slika 3.4: Slika vmesnika, ki prikazuje zemljevid in podatke v ležečem in pokončnem načinu.

3.1.3 Delovanje aplikacije

Glavni del naše aplikacije je servis, ki se zažene ob pritisku na gumb v uporabniškem vmesniku. Kot smo že omenili, mora uporabnik vnesti podatke za identifikacijo, vendar to ne zadošča. Potrebujemo še unikatni ključ za identifikacijo naprave, ki ga ne bo mogoče spremeniti. Na izbiro imamo več opcij. Lahko uporabimo naslov MAC (ang. *Media Access Control*), ki ga lahko pridobimo iz strojne opreme WiFi (ang. *Wireless Fidelity*) ali Bluetooth. Vendar ta naslov ni priporočljiv kot identifikator, saj nimajo vse naprave potrebne strojne opreme. Upoštevati moramo tudi, da je lahko WiFi izklopljen. Tedaj strojna oprema morda ne bo vrnila naslova MAC. Ob prvem zagonu naprave se generira naključen unikaten ključ *ANDROID_ID*. Vendar ima ta identifikator nekaj slabosti: identifikator ni 100% zanesljiv na Androidih pred različico 2.2 ("Froyo"). Prav tako je prišlo do vsaj enega široko opaznega programskega hrošča, kjer je veliko število primerkov imelo enak *ANDROID_ID*. Ker želimo, da se ključ odstrani po izbrisu aplikacije, ne moremo uporabiti serijske številke naprave. Zato smo se odločili uporabiti razred UUID (ang. *Universally Unique Identifier*). UUID je nespremenljiva predstavitev s 128-bitnem univerzalnim unikatnim identifikatorjem. Identifikator je shranjen v datoteki, tako nam je zmeraj dostopen. Datoteka z identifikatorjem se odstrani vključno z aplikacijo. Slika 3.5 prikazuje programsko kodo, ki generira UUID in ga shrani v datoteko [17].

```
public class ID {
    private static String sID = null;
    private static final String KEY = "KEY";

    public synchronized static String id(Context context) {
        if (sID == null) {
            File installation = new File(context.getFilesDir(), KEY);
            try {
                if (!installation.exists())
                    writeInstallationFile(installation);
                sID = readInstallationFile(installation);
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
        return sID;
    }

    private static String readInstallationFile(File installation) throws IOException {
        RandomAccessFile f = new RandomAccessFile(installation, "r");
        byte[] bytes = new byte[(int) f.length()];
        f.readFully(bytes);
        f.close();
        return new String(bytes);
    }

    private static void writeInstallationFile(File installation) throws IOException {
        FileOutputStream out = new FileOutputStream(installation);
        String id = UUID.randomUUID().toString();
        out.write(id.getBytes());
        out.close();
    }
}
```

Slika 3.5: Odsek programske kode, ki generira in shrani UUID.

3.1.3.1 Pridobivanje lokacije naprave

Za pridobivanje lokacije smo uporabili razred Loacation Manager. Ta razred omogoča dostop do sistemskih storitev za lokacijo. Te storitve omogočajo aplikaciji za pridobitev redne posodobitve geografskih koordinat naprave, ali pa zagon procesa, ko je naprava v bližini neke geografske lokacije [34]. Za delovanje moramo nastaviti nekaj parametrov. Za tip ponudnika lahko izbiramo med GPS in ponudnikom omrežja, ki skuša oceniti lokacijo naprave s pomočjo dostopnosti WiFi točke in telefonskega oddajnika. Ker je GPS veliko natančnejši, je bila to naša opcija že od samega začetka. Naslednji parameter je minimalna razdalja med intervali za obvestila. Nastavili smo ga na en meter, tako da ne bomo posodabljali koordinat, če naprava ne spreminja lokacije, prav tako bomo prihranili na energiji baterije. Ker parameter minimalni čas med intervalni ni dovolj natančen in se uporablja le kot namig, smo ga postavili na 0. Kot alternativo parametra minimalnega časa smo uporabili štoparico, ki bo po intervalu omogočila pošiljanje podatkov. Časovni interval bo na strežniku odstopal zaradi različnih časovnih zakasnitev. Ko so vsa polja

vnesena, zaženemo zahtevo, ki bo posodabljala lokacijo. Vsaka sprememba lokacije kliče metodo onLocationChanged. V njej lahko preberemo geografsko dolžino in širino. Kombinacija teh dveh komponent določa položaj katerekoli lokacije na planetu, vendar pa ne upošteva nadmorske višine oz. globine. V tej metodi smo prebrali tudi višino, hitrost in natančnost. Natančnost je podana v metrih, ki pove za koliko odstopajo koordinate od dejanske lokacije.

3.1.3.2 Komunikacija s strežnikom

V metodi onLocationChanged najprej preverimo, če je vzpostavljena povezava in če se je že končal časovni interval. Če pogoja ustrezata, pripravimo podatke za pošiljanje na strežnik. Aplikacija podatke najprej pretvori v format JSON z uporabo Googlove odprtokodne knjižnice Gson, ki se lahko uporablja za pretvorbo Java objektov v JSON. Prav tako se lahko uporablja za pretvorbo nizov JSON v enakovreden Java objekt. Ko so podatki v formatu JSON, jih pošljemo servletu z metodo POST. Če dobimo od strežnika ustrezen odgovor, so bili podatki uspešno poslani. Nato pošljemo servletu zahtevo po velikosti časovnega intervala z metodo GET. Uporabnik ne more spremeniti IP naslova strežnika preko vmesnika. Vsakič, ko želimo zamenjati naslov, moramo to nastaviti v izvorni kodi.

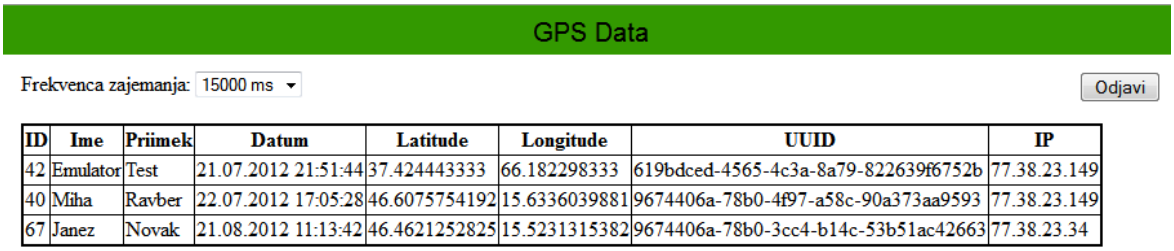
3.2 Strežnik

Strežnik ima dva tipa odjemalcev, in sicer uporabnika, ki bo dostopal do strani JSP preko spletnega brskalnika ter aplikacijo na napravah Android, ki bo komunicirala preko servletov. Pri zahtevah obeh odjemalcev mora strežnik dostopati do podatkovne baze, kar je časovno zahtevna operacija. Komponente na strežniku so prenosljive, ker je tehnologija neodvisna od strežnika in platforme.

3.2.1 Grafični vmesnik

Grafični vmesnik je izdelan v tehnologiji JSP. Dostop do spletnega vmesnika smo omejili, zato se mora uporabnik pri vstopu najprej prijaviti. Po prijavi se prikaže glavni vmesnik s tabelo (slika 3.6), ki se posodobi vsakih 1000 milisekund. Ker smo želeli, da se posodablja le prikaz podatkov v tabeli brez osveževanja celotne strani, smo uporabili tehnologijo AJAX. Tako vsakih 1000 milisekund funkcija pošlje zahtevo servletu. Servlet prebere

zadnjo lokacijo vsakega uporabnika iz baze in kot odgovor generira HTML tabelo. Upoštevati moramo, da ta operacija postane časovno zahtevna pri velikem številu podatkov. Na strani najdemo tudi polje za izbiranje frekvence zajemanja podatkov. Ko izberemo frekvenco zajemanja podatkov, se ta zapiše v bazo.



ID	Ime	Priimek	Datum	Latitude	Longitude	UUID	IP
42	Emulator	Test	21.07.2012 21:51:44	37.424443333	66.182298333	619bdced-4565-4c3a-8a79-822639f6752b	77.38.23.149
40	Miha	Ravber	22.07.2012 17:05:28	46.6075754192	15.6336039881	9674406a-78b0-4f97-a58c-90a373aa9593	77.38.23.149
67	Janez	Novak	21.08.2012 11:13:42	46.4621252825	15.5231315382	9674406a-78b0-3cc4-b14c-53b51ac42663	77.38.23.34

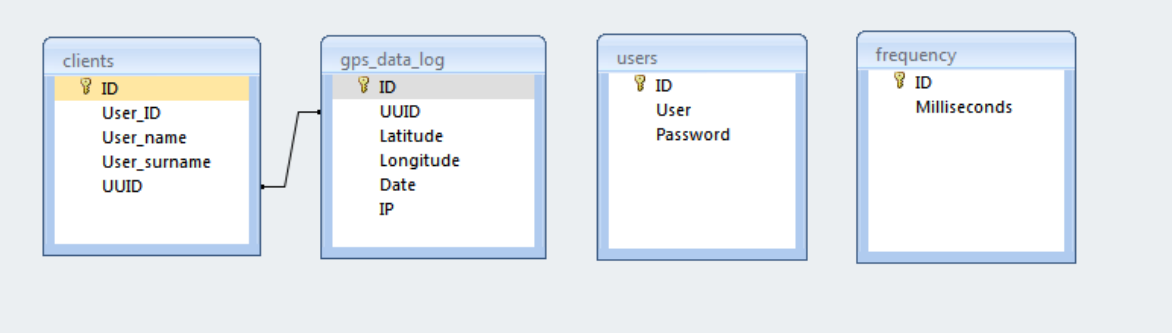
Slika 3.6: Spletna stran s podatki uporabnikov.

3.2.2 Servleti

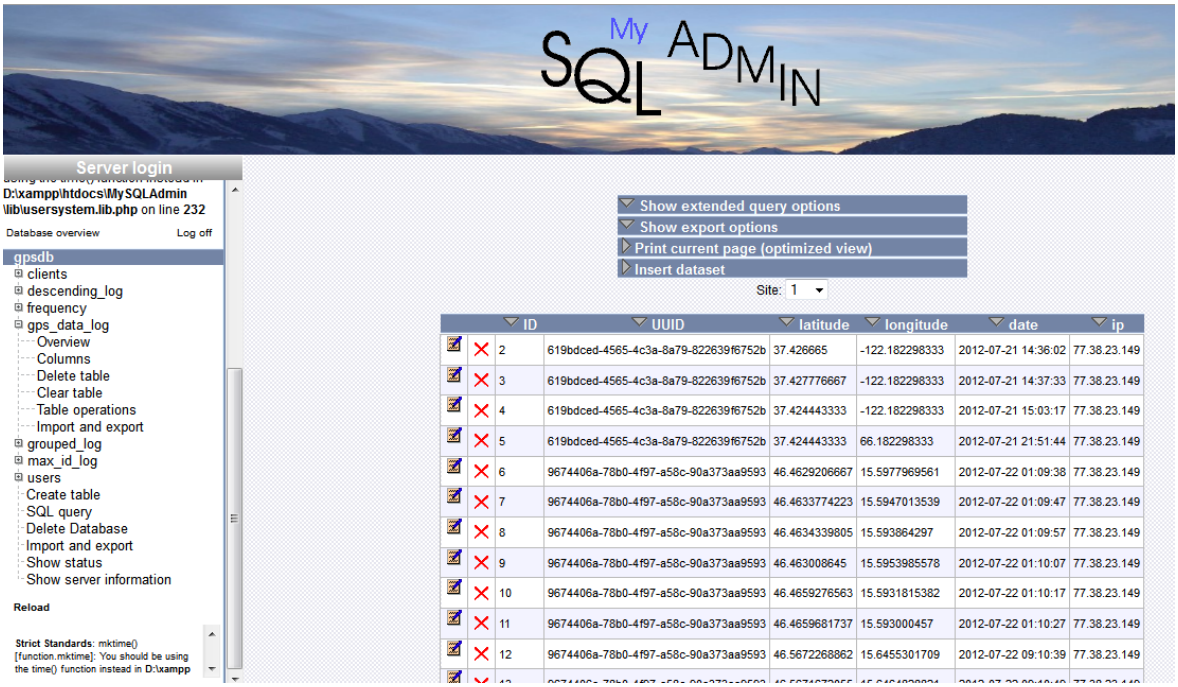
Ključni del strežnika so servleti. Glavni servlet je zadolžen za komunikacijo z napravami Android. V metodi doGet() se iz baze prebere frekvenca zajemanja podatkov in se vrne kot odgovor. V metodi doPost() se iz zahteve razpozna IP (ang. *Internet Protocol*) naslov odjemalca in JSON. IP naslov potrebujemo za natančnejše identificiranje uporabnika. Ko iz kode JSON razberemo vse podatke, jih shranimo v podatkovno bazo. Če so se operacije izvedle uspešno, se to sporoči odjemalcu s pomočjo odgovora HTTP.

3.2.3 Podatkovna baza

V podatkovni bazi imamo štiri tabele (slika 3.7). V tabeli users so shranjeni uporabniki, ki se lahko prijavijo v spletno strani in vidijo podatke. Tabela frequency hrani frekvenco zajemanja podatkov v milisekundah. Podatke o uporabnikih hranimo v tabeli clients. Najpomembnejša tabela je gps_data_log, saj v njej hranimo koordinate uporabnikov. Tabela ima kot tuji ključ UUID uporabnika. Tako vemo, kateremu uporabniku pripadajo koordinate in ostali podatki. Ker želimo na spletni strani prikazati samo zadnji položaj uporabnika, smo uporabili ustrezni SQL (ang. *Structured Query Language*) stavek. Ker bo s strežnikom komuniciralo več naprav, smo metode, ki dostopajo do baze, sinhronizirali. Za izdelavo podatkovne baze smo uporabili orodje MySql Administrator (slika 3.8).



Slika 3.7: E-R diagram podatkovne baze.



Slika 3.8: Vmesnik odjemalca MySql Admin.

4 PREIZKUŠANJE

4.1 Preizkušanje aplikacije mobilne naprave

4.1.1 Preizkušanje na emulatorju

Aplikacija je bila preizkušena na emulatorju, saj lahko programsko kodo zaženemo z enim klikom na gumb. Napake smo najpogosteje odpravljali s koračnim razhroščevanjem med izvajanjem aplikacije. Emulator lahko konfiguriramo z uporabo AVD (ang. *Android Virtual Device*). Najpomembnejša lastnost naše virtualne naprave je podpora GPS, ki pa je že privzeto vklopljena. Emulator ne more samodejno pridobiti GPS lokacijo, zato smo za preizkušanje GPS prejemnika v nadzoru emulatorja vnašali lažne geografske koordinate in jih pošiljali virtualni napravi. V upravitelju AVD lahko tudi izbiramo med različnimi verzijami operacijskega sistema Android in stopnjami API. Stopnja API igra ključno vlogo pri zagotavljanju najboljše možne izkušnje za uporabnike in razvijalce aplikacij:

- omogoča platformi Android opisati največjo revizijo okvirja API, ki ga podpira;
- omogoča aplikacijam opisati revizijo okvirja API, ki ga potrebujejo;
- omogoča preverjanje namestitev aplikacij tako, da nezdružljive različice aplikacije niso nameščene [25].

Ko ustvarjamo nov AVD, lahko določimo sledeče nastavitve strojne opreme, ki jo AVD emulira in so prikazane v tabeli 1.

Tabela 1: Nastavitve AVD [26].

Značilnost	**Opis**
Velikost delovnega pomnilnika	Količina fizičnega pomnilnika na napravi, v MB. Privzeta vrednost je "96".
Podpora za zaslon na dotik	Ali je na napravi zaslon na dotik ali ne. Privzeta vrednost je "da".
Podpora sledilne kroglice	Ali obstaja sledilna kroglica na napravi. Privzeta vrednost je "da".
Podpora za tipkovnico	Ali ima naprava *QWERTY* tipkovnico. Privzeta vrednost je "da".
Podpora za DPad	Ali ima naprava DPad tipke. Privzeta vrednost je "da".
Podpora za GSM (ang. *System for Mobile Communications*) modem	Ali je GSM modem v napravi. Privzeta vrednost je "da".
Podpora za kamero	Ali ima naprava kamero. Privzeta vrednost je "ne".
Največje število pikslov kamere horizontalno	Privzeta vrednost je "640".
Največje število pikslov kamere vertikalno	Privzeta vrednost je "480".
GPS podpora	Ali je GPS v napravi. Privzeta vrednost je "da".
Podpora za baterijo	Ali lahko naprava deluje na baterijo. Privzeta vrednost je "da".
Merilnik pospeška (pospeškometer)	Ali obstaja pospeškometer v napravi. Privzeta vrednost je "da".
Podpora za snemanje zvoka	Ali lahko naprava snema zvok. Privzeta vrednost je "da".

Podpora za predvajanje zvoka	Ali lahko naprava predvaja zvok. Privzeta vrednost je "da".
Podpora za SD (ang. *Secure Digital*) kartico	Ali naprava podpira vstavitev/odstranitev virtualne kartice SD. Privzeta vrednost je "da".
Podpora za particijo predpomnilnika	Ali uporabljamo predpomnilniško particijo na napravi. Privzeta vrednost je "da".
Velikost predpomnilniške particije	Privzeta vrednost je "66MB".
Abstrahirana gostota LCD (ang. *Liquid Crystal Display*) zaslona	Nastavi gostoto splošnih značilnosti, ki jih uporablja zaslon AVD. Privzeta vrednost je "160".

4.1.1.1 Omejitve emulatorja

Emulator je priročno orodje, vendar ima kar nekaj omejitev:

- emulator ni naprava. Simulira splošno vedenje naprave in ne specifično strojno izvedbo;
- podatki senzorjev, kot so informacije o satelitski lokaciji, bateriji, nastavitvah napajanja in omrežne povezave, so vsi simulirani z računalnikom;
- periferija, kot je strojna oprema, kamere, ni popolnoma funkcionalna;
- telefonski pogovori ne morejo biti izvedeni ali prejeti, temveč so simulirani. Prav tako so simulirana SMS (ang. *Short Message Service*) sporočila in ne uporabljajo pravega omrežja; in
- ni podpore za USB (ang. *Universal Serial Bus*) in Bluetooth.

Uporaba emulatorja Android ni nadomestek za testiranje na pravi ciljni napravi [5].

4.1.1.2 Kompatibilnost z novejšimi različicam

Aplikacije Android so na splošno združljive (naprej) z vsako novo različico platforme Android, kajti skoraj vse spremembe okvira API se seštevajo. Aplikacija Android, razvita z uporabo katerekoli različice API (kot je določeno s svojo stopnjo API), je združljiva z novejšimi različicami platforme Android in višjim stopnjam API. Aplikacija bi morala

delovati na vseh novejših različicah platforme Android, razen v posameznih primerih, kjer aplikacija uporablja del API, ki se kasneje odstrani iz nekega razloga [26].

4.1.1.3 Kompatibilnost s starejšimi različicami

Android aplikacije niso nujno združljive (nazaj) z različicami platforme Android, starejše od tiste, za katero so bile razvite.

Vsaka nova različice platforme Android lahko vključuje nov okvir stopenj API, kot so tiste, ki dajejo aplikacijam dostop do novih zmogljivosti platforme ali nadomestitev obstoječih delov API. Nove stopnje API so dostopne aplikacijam, ko tečejo na novi platformi in, kot že omenjeno, tudi takrat, ko tečejo na novejših različicah platforme, kot je določeno s stopnjo API. Starejše različice platforme ne vključujejo novih stopenj API. Aplikacije, ki uporabljajo novi API, ne morejo teči na teh platformah [25].

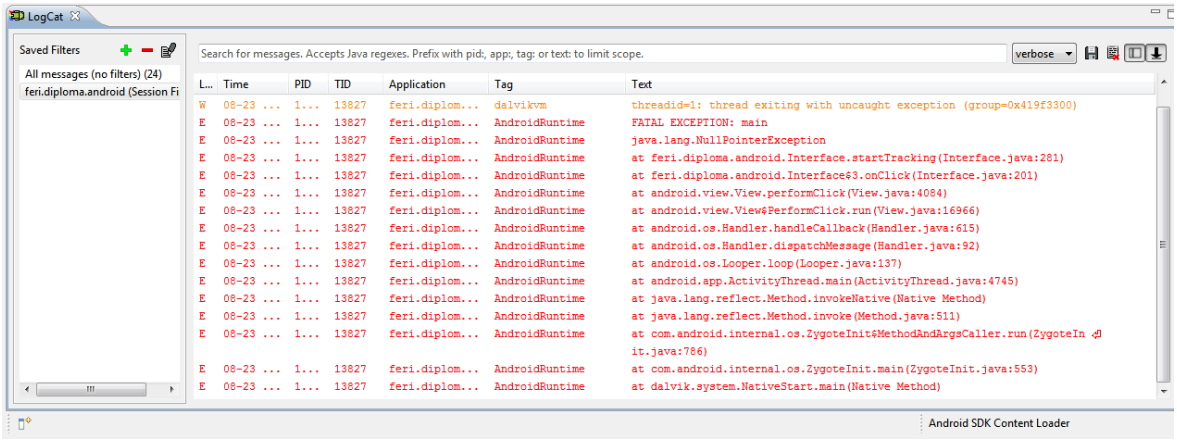
4.1.2 Preizkušanje na napravi Android

Ker z emulatorjem ne moremo ustvariti realnih okoliščin, je glavno preizkušanje potekalo na telefonu Samsung galaxy GT-I5510 s prednameščenim operacijskim sistemom Android 2.2 Froyo. Prav tako je bila aplikacija preizkušena na telefonu Nexus S. Nexus S ima nameščen Jelly bean 4.1.1, to je najnovejša različica operacijskega sistema Android.

Pri preizkušanju nam je bila v veliko pomoč perspektiva DDMS (ang. *Dalvik Debug Monitor Server*). Perspektiva DDMS se lahko uporablja za spremljanje procesov aplikacije, kakor tudi za interakcijo z emulatorjem. Lahko simuliramo govorne klice, pošiljamo SMS sporočila na emulator in posodabljamo lokacijo [5], [3].

Ob zagonu aplikacije najprej opazimo, da je potrebno počakati na sprejemnik GPS. Ta namreč ne more zbirati podatkov GPS, dokler nima signala od zadostnega števila satelitov. Aplikacija, kot pričakovano, porabi ogromno električne energije. K temu največ pripomore sprejemnik GPS, ki pa je ključen za delovanje aplikacije.

Pri preizkušanju aplikacije na mobilni napravi je prihajalo do napak, ki jih emulator ni javil. Da smo te napake lažje locirali, smo na zaslon izpisovali različne podatke. Ti podatki so vključevali vrednost spremenljivk in v katerem delu programske kode se aplikacija nahaja. Večino napak smo odpravili s pomočjo DDMS dnevnika (slika 4.1). V dnevniku se izpiše vrsta napake, kaj jo je sprožilo ter mesto napake. Za delovanje dnevnika mora biti naprava priključena na računalnik z USB vmesnikom.

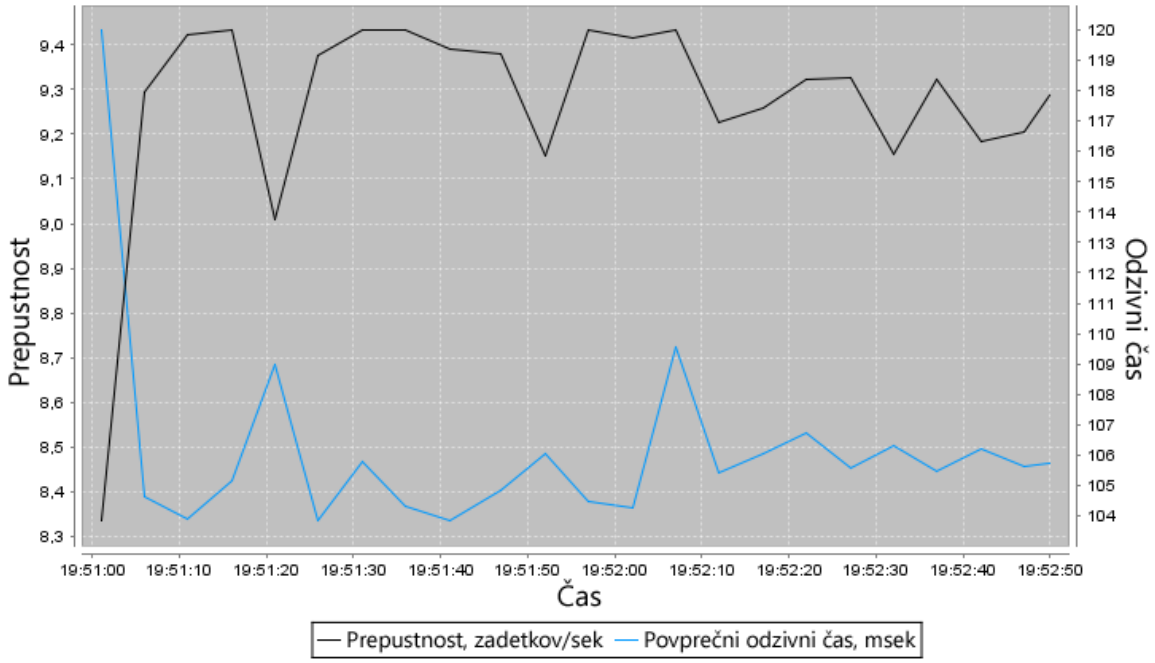


Slika 4.1: Napaka prikazana v DDMS dnevniku.

Pri preizkušanju smo si pomagali z aplikacijo Fake GPS Location. Aplikacija pošilja napravi lažne GPS podatke o lokaciji. V aplikaciji lahko nastavimo frekvenco pošiljanja koordinat in celo simuliramo gibanje. Za delovanje aplikacije moramo v nastavitvah naprave omogočiti lažno lokacijo. Po preizkušanju lahko trdimo, da je aplikacija dosegla pričakovane rezultate.

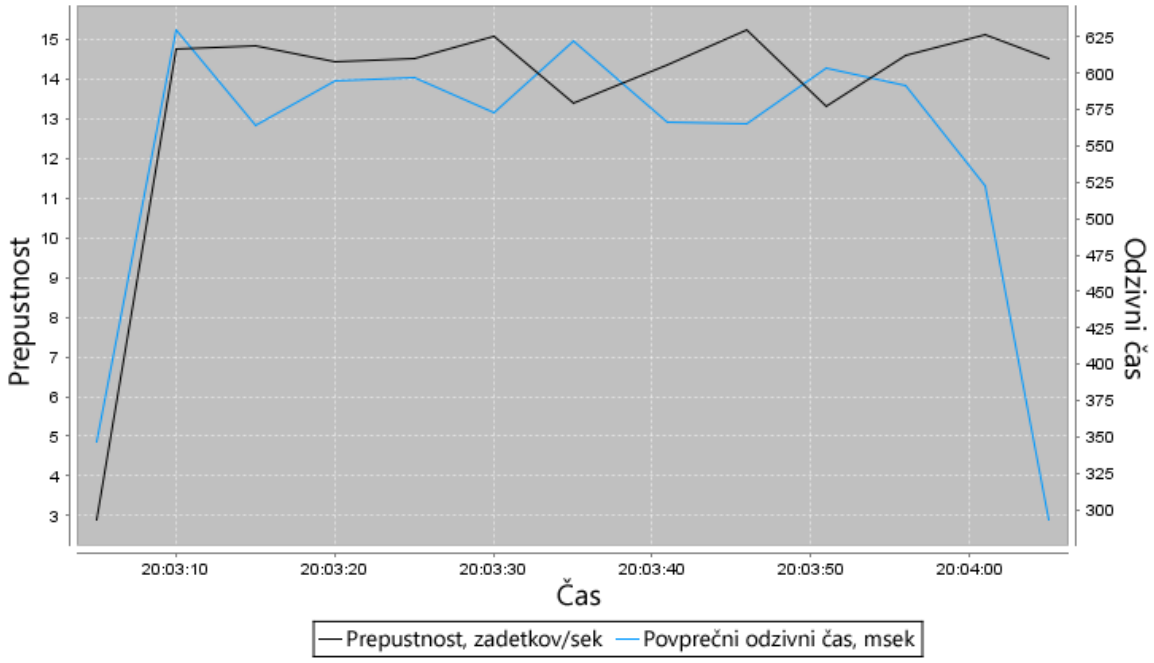
4.2 Preizkušanje strežnika

Napake na strežniku smo locirali s pomočjo konzole, kjer se izpiše sporočilo o napaki. Napake so se najpogosteje pojavljale pri pisanju in branju iz podatkovne baze zaradi neustreznih podatkov. Za preizkušanje strežnika smo uporabili odprtokodno aplikacijo Apache JMeter. Namizna aplikacija Apache JMeter je odprtokodna programska oprema, napisana v programskem jeziku Java, zasnovana za nalaganje testno funkcionalnega vedenja in merjenje učinkovitosti [36]. S pomočjo te aplikacije smo lahko preizkusili prepustnost in odzivni čas strežnika. V aplikaciji lahko nastavimo število odjemalcev in število zahtev posameznega odjemalca. Strežnik se normalno odziva pri enem odjemalcu, čeprav pošlje 1000 zahtev v dveh minutah. Na sliki 4.2 je predstavljen graf, ki prikazuje prepustnost (črna črta) in povprečen odzivni čas (modra črta). Razberemo lahko, da je prepustnost visoka, medtem ko je odzivni čas majhen.



Slika 4.2: Graf prikazuje odzivnost strežnika za enega uporabnika, ki pošlje 1000 zahtev v dveh minutah.

Ko smo povečali število odjemalcev na 10, smo opazili, da se pojavi ozko grlo, čeprav pošlje vsak odjemalec 100 zahtev. Na sliki 4.3 lahko vidimo, da je prepustnost visoka, vendar je visok tudi odzivni čas, kar pomeni, da strežnik potrebuje več časa za odgovor. Prišli smo do ugotovitve, da je strežnik primeren za manjše število uporabnikov.



Slika 4.3: Graf, ki prikazuje ozko grlo.

5 SKLEP

V diplomskem delu je predstavljen razvoj mobilne aplikacije za operacijski sistem Android. Aplikacija pridobi podatke GPS in jih pošlje na strežnik. Na strežniku lahko krmilimo frekvenco pošiljanja teh podatkov. Podatke shranimo v podatkovno bazo, kjer so dostopni za nadaljnjo obdelavo.

Največ težav so povzročale napake, ki so se pojavile med izvajanjem aplikacije na mobilni napravi. Te napake je težje locirati, kot pa napake, ki se pojavijo na emulatorju. Omejitev aplikacije je, da potrebuje dostop do interneta in signal GPS. Brez teh pogojev aplikacija ne deluje. Ker mora naprava imeti vklopljen GPS, je poraba električne energije zelo visoka in zato aplikacija ni primerna za daljšo uporabo brez napajanja.

V nadaljnjem razvoju bi lahko zadnje koordinate mobilne naprave prikazali na zemljevidu. Tako bi lažje videli kako točne so koordinate in kje se naprava nahaja. Zemljevid bi bil dostopen na spletni strani. Za izboljšanje odzivnosti strežnika, bi lahko namesto servletov uporabili spletne storitve.

6 VIRI, LITERATURA

[1] Reto Meier, *Professional Android 2 Application Development*, Wrox, 2010.

[2] Mark Murphy, *Beginning Android*, Apress, 233 Spring Street, New York, 2009.

[3] Mark Murphy, *Beginning Android 2*, Apress, 233 Spring Street, New York, 2010.

[4] James Steele, Nelson To, *The Android Developer's Cookbook*, Addison-Wesley Professional, 2010.

[5] Lauren Darcey, Shane Conder, *Sams Teach Yourself Android Application Development in 24 Hours*, Sams, 2010.

[6] Arhitektura Android, http://www.cellphoneanswers.info/android-architecture/, [13. 05. 2011].

[7] Arhitektura Android, http://blog.zeustek.com/2010/11/11/android-architecture/ [13. 05. 2011].

[8] Wikipedija: programski jezik Java, http://en.wikipedia.org/wiki/Java_(programming_language), [14. 05. 2011].

[9] Wikipedija: Java Server Pages (JSP), http://en.wikipedia.org/wiki/JavaServer_Pages, [15. 05. 2011].

[10] Operacijski sistem Android, http://www.videoconverterfactory.com/glossary/android.html, [15. 05. 2011].

[11] Wikipedija: Java Servlet, http://en.wikipedia.org/wiki/Java_Servlet, [16. 05. 2011].

[12] Operacijski sistem Android, http://developer.android.com/guide/basics/what-is-android.html, [18. 05. 2011].

[13] Wikipedija: Operacijski sistem Android, http://en.wikipedia.org/wiki/Android_(operating_system), [18. 05. 2011].

[14] Aktivnost, http://developer.android.com/guide/topics/fundamentals/activities.html, [19. 05. 2011].

[15] Življenjski cikel aktivnosti, http://answers.oreilly.com/topic/2692-android-programming-understanding-the-activity-life-cycle/, [21. 05. 2011].

[16] Servis, http://developer.android.com/guide/topics/fundamentals/services.html, [22. 05. 2011].

[17] Identifikacija Android naprave, http://android-developers.blogspot.com/2011/03/identifying-app-installations.html, [24. 05. 2011].

[18] Android Debug Bridge (ADB), http://developer.android.com/guide/developing/tools/adb.html, [27. 05. 2011].

[19] Wikipedia: JavaScript Object Notation (JSON), http://en.wikipedia.org/wiki/JSON, [08. 05. 2011].

[20] JavaScript Object Notation (JSON), http://www.json.org/json-sl.html, [02. 06. 2011].

[21] Wikipedija: HTTP metoda POST, http://en.wikipedia.org/wiki/POST_(HTTP), [07. 06. 2011].

[22] HTTP metoda GET, http://docstore.mik.ua/orelly/java-ent/servlet/ch02_01.htm, [10. 06. 2011].

[23] Wikipedija: AJAX, http://en.wikipedia.org/wiki/Ajax_(programming), [13. 06. 2011].

[24] Android opis dovoljenja, http://developer.android.com/reference/android/Manifest.permission.html, [15. 06. 2011].

[25] API stopnje, http://developer.android.com/guide/appendix/api-levels.html, [20. 07. 2011].

[26] Upravljanje AVD z AVD upraviteljem, http://developer.android.com/guide/developing/devices/managing-avds.html,

[18. 07. 2011].

[27] API ključ, https://developers.google.com/maps/documentation/android/#apikey, [25 07 2012].

[28] Ponudniki vsebin, http://developer.android.com/guide/topics/providers/content-providers.html, [22. 8. 2012].

[29] Emulator, http://developer.android.com/tools/devices/emulator.html, [22. 8. 2012].

[30] Digitalni podpis, http://developer.android.com/guide/publishing/app-signing.html, [22. 8. 2012].

[31] Apache Tomcat, http://en.wikipedia.org/wiki/Apache_Tomcat, [22. 8. 2012].

[32] Android dovoljenja, http://developer.android.com/guide/topics/security/permissions.html, [22. 8. 2012].

[33] Sprememba konfiguracije, http://developer.android.com/reference/android/app/Activity.html, [22. 8. 2012].

[34] Location Manager, http://developer.android.com/reference/android/location/LocationManager.html, [22. 8. 2012].

[35] Sprejemnik, http://developer.android.com/reference/android/content/BroadcastReceiver.html, [23. 8. 2012].

[36] Apache JMeter, http://jmeter.apache.org/, [24. 8. 2012].

[37] Eclipse, http://en.wikipedia.org/wiki/Eclipse_(software), [30. 8. 2012].

7 PRILOGE

7.1 Naslov študenta

Miha Ravber
Na Fari 24
2391 Prevalje

7.2 Kratek življenjepis

Rojen: 25. 01. 1990 v Slovenj Gradcu
Šolanje:

- Srednja šola Ravne na Koroškem
- Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE DIPLOMSKEGA DELA IN OBJAVI

Ime in priimek diplomanta-tke: Miha Ravber

Vpisna številka: E1009677

Študijski program: Računalništvo in informacijske tehnologije

Naslov diplomskega dela: Koordiniran prenos GPS podatkov mobilne naprave do strežnika

Mentor: prof. dr. Janez Brest

Somentor: doc. dr. Borko Bošković

Podpisani-a Miha Ravber izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Diplomsko delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 16/2007) dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija diplomskega dela je istovetna elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum diplomiranja, naslov diplomskega dela) na spletnih straneh in v publikacijah UM.

Kraj in datum: Maribor, 11.9.2012

Podpis diplomanta-tke: Ravber

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Miha Ravber,

z vpisno številko E1009677,

sem avtor/-ica diplomskega dela z naslovom:

Koordiniran prenos GPS podatkov mobilne naprave do strežnika

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  prof. dr. Janez Brest

  in somentorstvom (naziv, ime in priimek)

  doc. dr. Borko Bošković

- 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 DKUM.

V Mariboru, dne 11.9.2012 Podpis avtorja/-ice: Ravber

Univerza v Mariboru
Fakulteta za elektrotehniko, racunalnistvo in informatiko

IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Podpisani mentor prof. dr. Janez Brest izjavljam, da je
(ime in priimek mentorja)
študent Miha Ravber izdelal diplomsko
(ime in priimek študenta-tke)

delo z naslovom: Koordiniran prenos GPS podatkov mobilne naprave do strežnika

(naslov diplomskega dela)

v skladu z odobreno temo diplomskega dela, Navodili o pripravi diplomskega dela in mojimi navodili.

Datum in kraj: 11. 9. 2012

Podpis mentorja: