UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Bojanec

RAZVOJ APLIKACIJE VODIČ PO LJUBLJANI NA MOBILNI PLATFORMI ANDROID

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01650/2010
Datum: 05.04.2010

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

Kandidat: **MATEJ BOJANEC**

Naslov: **RAZVOJ APLIKACIJE VODIČ PO LJUBLJANI NA MOBILNI PLATFORMI ANDROID**

DEVELOPMENT OF THE APPLICATION LJUBLJANA GUIDE ON THE MOBILE PLATFORM ANDROID

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Kandidat naj v svoji diplomski nalogi predstavi mobilno platformo Android. Na konkretnem primeru aplikacije naj predstavi razvojno okolje, ki se uporablja za razvoj mobilnih aplikacij. Predstavljeni naj bodo tudi vsi programski konstrukti in strojna oprema telefonov, ki je bila uporabljena za razvoj dotične aplikacije. Natančneje naj opiše posamezne korake razvoja aplikacije od same ideje pa do faze testiranja ter končne objave takšne aplikacije na medmrežju drugim uporabnikom. Za zaključek naj razvito aplikacijo predstavi tudi s stališča uporabniške izkušnje.

Mentor:
prof. dr. Franc Solina

Fakulteta za računalništvo in informatiko 2 UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Franc Solina

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Bojanec

RAZVOJ APLIKACIJE VODIČ PO LJUBLJANI NA MOBILNI PLATFORMI ANDROID

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: dr. Franc Solina

Ljubljana, 2010

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Matej Bojanec
z vpisno številko 63030137

sem avtor diplomskega dela z naslovom:
RAZVOJ APLIKACIJE VODIČ PO LJUBLJANI NA MOBILNI PLATFORMI ANDROID

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  dr. Franc Solina

- 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 25.9.2010 Podpis avtorja: _____

ZAHVALA

Najprej bi se rad zahvalil staršem za omogočen študij in izkazano podporo. Posebej bi se rad zahvalil somentorju dr. Borutu Batagelju za mentorstvo in nasvete pri izdelavi diplomske naloge. Zahvala pa gre tudi sošolcem in prijateljem, s katerimi smo preživeli nepozabna študentska leta.

KAZALO

SEZNAM UPORABLJENIH KRATIC IN SIMBOLOV

2D	Two Dimensional (2-dimenzionalni prostor)
3D	Three Dimensional (3-dimenzionalni prostor)
API	Application Programming Interface (programski vmesnik)
GB	Gigabyte (oznaka za kapaciteto pomnilnika)
GHz	Gigahertz (enota za frekvneco ure procesorja)
GPS	Global Positioning System (sistem globalnega določanja položaja)
HTML	Hyper Text Markup Language (jezik za označevanje nadbesedila)
IDE	Integrated Development Environment (razvojno okolje)
IP	Internet Protocol address (enolična številka v internet omrežju)
JDK	Java Development Kit (skupek javanskih razvojnih orodij)
MD5	Message-Digest algorithm 5 (kodirna funkcija)
OS	Operating System (operacijski sistem)
SD	Secure Digital (vrsta pomnilniških kartic)
SDK	Software Development Kit (orodje za razvoj programske opreme)
SMS	Short Message Sservice (storitev kratkih sporočil)
USB	Universal Serial Bus (univerzalno serijsko vodilo)
WLAN	Wireless Local Area Network (brezžično lokalno omrežje)
XML	eXtensible Markup Language (razširljiv označevalni jezik)

POVZETEK

V diplomskem delu je obravnavano področje razvoja aplikacij na mobilni platformi Android. V uvodnih poglavjih je predstavljen operacijski sistem Android ter razvojno okolje. Predstavljena bo trenutno zadnja različica operacijskega sistema 2.2 (Froyo) in njemu pripadajoči Android SDK (angl. Software Development Kit). Poleg tega je predstavljena tudi strojna oprema mobilnih telefonov, ki imajo nameščen Android OS (operacijski sistem) in je uporabljena v aplikaciji Vodič po Ljubljani.

Osrednji del govori o procesu razvoja aplikacij na primeru aplikacije Vodič po Ljubljani in možnosti uporabe mobilne naprave pri ogledu in spoznavanju mesta. Spoznali bomo celoten potek razvoja mobilne aplikacije, testiranje in končne objave aplikacije na Android Marketu.

Na koncu so obdelani še nastali problemi in možnosti za izboljšavo tovrstnih aplikacij.

Ključne besede:

Android, mobilna aplikacija, razvoj, testiranje, operacijski sistem, GPS, Android Market

ABSTRACT

Thesis deals with development of applications on the mobile platform Android. The introduction chapter presents the Android operating system and development environment. The latest version of the current operating system 2.2 (Froyo) and its associated Android Software Development Kit is presented. In addition, there is a presentation of mobile phone hardware that have the Android OS (operating system) installed and use the application Vodič po Ljubljani

The central part discusses the process of developing application in case of Vodič po Ljubljani and the possibility of using mobile devices for viewing and exploring the city. The entire course of development of mobile application, testing and final publication of applications at the Android Market is described.

Finally, encountered problems and opportunities for improvement in such applications are presented.

Keywords:

Android, mobile application, development, testing, operating system, GPS, Android Market

1 UVOD

Osebni računalnik in internet sta revolucionarno spremenila način povezovanja ljudi, njihove zabave in izmenjave informacij. Vendar pa najboljšo povezljivost kjerkoli in kadarkoli doseže mobilni telefon.

Konec leta 2009 je storitve mobilne telefonije uporabljalo že 4,6 milijarde naročnikov. Rast števila naročnikov se bo nadaljevala tudi v letu 2010 in po predvidevanjih ITU (angl. Intenational Telecomunication Union) naj bi dosegla 5 milijard [1]. To lahko pripišemo porasti pametnih telefonov v razvitih državah in mobilnih operaterjev v državah v razvoju. Zanimiv podatek je tudi število naročnikov na mobilne širokopasovne storitve, ki se bo povečalo iz 600 milijonov v letu 2009 na 1 milijardo v letu 2010. V naslednjih letih lahko pričakujemo, da bo več ljudi dostopalo do spleta prek prenosnikov, mobilnih telefonov in podobnih naprav kot pa prek namiznih računalnikov.

Poleg močne rasti števila uporabnikov se sunkovito razvija tudi strojna in programska oprema. Mobilni telefoni se danes po zmogljivostih lahko kosajo z računalniki izpred nekaj let. So kot računalniki v malem s procesorji prek 1 GHz, zasloni na dotik, GPS moduli in kamerami, ki pridejo do izraza v povezavi z mobilnim operacijskim sistemom in aplikacijami.

Največji tržni delež na trgu mobilnih operacijskih sistemov trenutno pripada Symbian OS. Sledijo mu RIM, iPhone OS, Android OS in Microsoft Windows Mobile, kar skupaj predstavlja 95% delež [2]. Po zadnjih podatkih za leto 2010 sta tržni delež povečala samo iPhone OS in Android OS. Poleg operacijskega sistema pa uporabnost mobilnega telefona povečajo aplikacije. Vsak od operacijskih sistemov ima svoj nabor aplikacij, vendar po številu aplikacij vodi in bo še nekaj časa iPhone OS s preko 150.000 aplikacijami. Med njimi lahko najdemo največ iger ter veliko uporabnih aplikacij za obdelavo slik in videa, povezovanja v socialne mreže, navigacije elektronske pošte in še mnogo več.

Diplomsko delo se bo omejilo samo na Android OS in razvoj aplikacij na dotični platformi. Predstavljen bo sam operacijski sistem in pripadajoče razvojno okolje Android SDK z Eclipse IDE (angl. Integrated Development Environment). V nadaljevanju bomo spoznali še strojno opremo mobilnih telefonov z operacijskim sistemom Android. Podrobneje bo predstavljen GPS modul in digitalni kompas, katera bosta imela glavno vlogo pri aplikaciji Vodič po Ljubljani. Zatem pa preidemo na razvoj same aplikacije začetnega ogrodja, uporabniškega vmesnika, logike do testiranja in produkcijske faze. Zaključno poglavje bo zajemalo

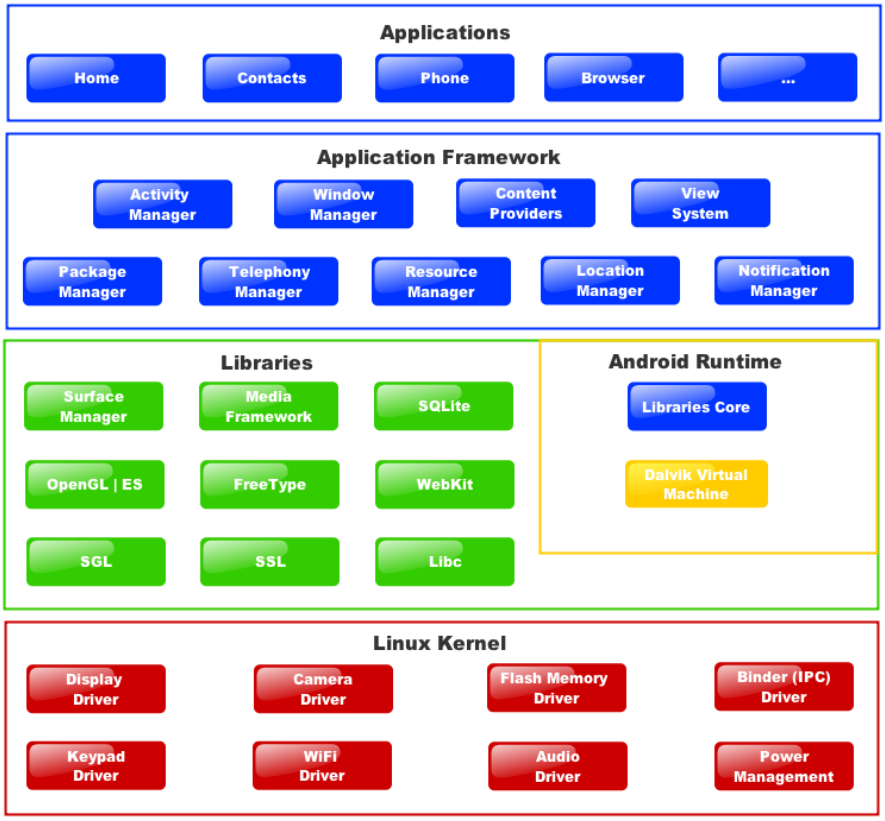
predstavitev zaključene aplikacije z ugotovitvami in možnosti nadaljnje nadgraditve same aplikacije.

2 MOBILNA PLATFORMA ANDROID

V tem poglavju bomo opredelili zgradbo in delovanje operacijskega sistema Android. Definirali bomo njegove zahteve in omejitve ter pregledali prednosti pred ostalimi mobilnimi operacijskimi sistemi. Drugi del poglavja bo govoril o razvojnem okolju Android SDK v povezavi z Eclipse IDE ter potrebnem znanju za razvoj aplikacij.

2.1 OPERACIJSKI SISTEM ANDROID

Izraz »Android« izhaja iz grške besede andr- kar pomeni 'človek' in pripone –oid, kar pomeni 'podoben', kar skupaj pomeni 'podoben človeku'. Koncept operacijskega sistema Android je zasnoval Andy Rubin kot odprtokodno mobilno platformo [3,4]. Taka zasnova omogoča široki množici programerjev sodelovanje pri razvoju samega sistema in aplikacij. Po mnenju nekaterih je tak način najboljši za razvoj najboljših možnih rešitev in aplikacij. Leta 2005 je Andy Rubin koncept predstavil podjetju Google, ki je Android tudi kupil. Leta 2008 je bila platforma tudi javno predstavljena in možen je bil dostop do izvorne kode.
Da bi razumeli Android, moramo spoznati njegovo zgradbo. Zgrajen je od spodaj navzgor kot sklad plasti, kjer vsaka opravlja svoje naloge (slika 1) [5].



Slika 1: Shema arhitekture operacijskega sistema Android

- **Linux jedro (angl. Linux Kernel)**

  Različica uporabljenega prilagojenega Linux jedra je 2.6, ki je zadolženo za upravljanje procesov, pomnilnika, omrežja, itd.

- **Izvajalnik kode Android (angl. Android Runtime)**

  Na tej plasti sta združeni 2 komponenti. Prva so skupek glavnih knjižnic, ki ponujajo večino funkcionalnosti programskega jezika Java. Druga pa je virtualni stroj Dalvik, ki deluje kot vmesnik med operacijskim sistemom in aplikacijami. Java je osnova za vse aplikacije, ki tečejo na Android-u. Ker operacijski sistem neposredno ne razume javanske programske kode, jo virtualni stroj Dalvik prevede v operacijskemu sistemu znano kodo. Pomemben podatek je, da vsaki delujoči aplikaciji pripada lasten virtualen stroj. S tem različne aplikacije ne morejo vplivati ena na drugo in napaka v programu vpliva samo na aplikacijo in ne na delovanje celotnega sistema.

- **Knjižnice**

  Vse Android knjižnice so vse napisane v programskem jeziku C in C++. Funkcionalnosti so razvijalcem izpostavljene preko Android aplikativnega ogrodja. V njih se nahaja podpora za 2D in 3D grafiko, podatkovno bazo, splet, avdio, video in še mnogo drugega.

- **Android aplikativno ogrodje**

  Ogrodje zagotavlja enak API (angl. Application programming interface) kot ga uporabljajo osnovne aplikacije tudi razvijalcem. Arhitektura je zastavljena tako, da zagotavlja enostavno ponovno uporabo komponent. Aplikacije lahko tudi ponudijo svoje komponente na uporabo drugim aplikacijam. Enako lahko tudi razvijalec nekatere komponente zamenja s svojimi.

- **Aplikacije**

  Sistem v paketu z Android-om ima prednaložene osnovne aplikacije kot so brskalnik, klient za elektronsko pošto, program za SMS sporočila, Google Maps, koledar, imenik in še veliko več. Vse te aplikacije so napisane v programskem jeziku Java. Dobro je omeniti, da je sistem Android večopravilen, kar nam omogoča uporabo več aplikacij

hkrati in tako lahko med pisanjem sporočila poslušamo glasbo. To je edina plast, katero bo uporabljal navaden uporabnik mobilnega telefona.

Android ima nekaj posebnih značilnosti, zaradi katerih izstopa glede na ostale mobilne tehnologije. Poglejmo si jih nekaj.

- **Odprta platforma**

  Android je odprtokodni projekt, kar omogočajo člani OHA (angl. Open Handset Alliance). Tak način omogoča raznolikost mnenj in pogledov na razvoj platforme za najboljše možne rešitve problemov in hkrati omogoča razvijalcem lažje razumevanje sistema.

- **Preprost dostop do ključne funkcionalnosti**

  Android omogoča bogat nabor knjižnic za dostop do ključnih funkcij, ki jih lahko razvijalci uporabijo v svojih aplikacijah. Android ogrodje nam omogoča dostop do funkcionalnosti preko Java razredov. Ni nam potrebno vedeti kako se pošilja SMS sporočilo, uporabimo samo potrebne knjižnice za pošiljanje SMS.

- **Razširljive in zamenljive aplikacije**

  Android aplikacije (tudi osnovne prednaložene aplikacije) lahko razširimo in obogatimo z novo inovativno funkcionalnostjo. Če gremo še korak naprej, lahko osnovne aplikacije zamenjamo s svojimi. Recimo, da naredimo svojo aplikacijo za upravljanje s kontakti in jo določimo kot privzeto aplikacijo za upravljanje s kontakti.

- **Povezljivost aplikacij**

  Android aplikacije se med sabo lahko povezujejo z izpostavljanjem funkcionalnosti ena drugi. Ena aplikacija lahko pripravlja podatke, medtem ko lahko druga uporabi pripravljene podatke. Tako lahko aplikacije dosegajo višje nivoje uporabnosti.

2.2 ANDROID SDK

Android programski paket za razvoj aplikacij (angl. software development kit ali SDK) združuje celovito množico razvijalskih orodij. V paketu dobimo razhroščevalnik, knjižnice, posnemovalnik mobilne naprave (v nadaljevanju emulator), dokumentacijo in vrsto primerov. Trenutno podprti operacijski sistemi so Linux, Mac OS in Windows. Za začetek razvoja potrebujemo še Apache Ant in Java programski paket za razvoj aplikacij (angl. java development kit ali JDK). Edino uradno podprto razvojno okolje (angl. integrated development environment ali IDE) je Eclipse od različice 3.4 naprej z dodanim vtičnikom Android Development Tools (ADT). Razvijalci lahko uporabijo tudi druge urejevalnike teksta za urejanje java in xml datotek, s katerimi operirajo prek ukazne vrstice. Preko ukazne vrstice se lahko tudi upravlja s priključenimi napravami.

2.2.1 ANDROID EMULATOR

Vsak Android SDK vsebuje emulator oz. virtualno mobilno napravo, ki teče na našem računalniku. Omogoča nam testiranje aplikacij brez uporabe priklopljene fizične naprave. Emulator posnema vse strojne in programske značilnosti tipične mobilne naprave z nekaj omejitvami. Okno emulatorja vsebuje celoten nabor možnih gumbov za izvajanje akcij z uporabo miške ali tipkovnice in ekran, kjer se prikazuje izvajanje aplikacije (slika 2).

Slika 2: Okno emulatorja z operacijskim sistemom Android 2.1

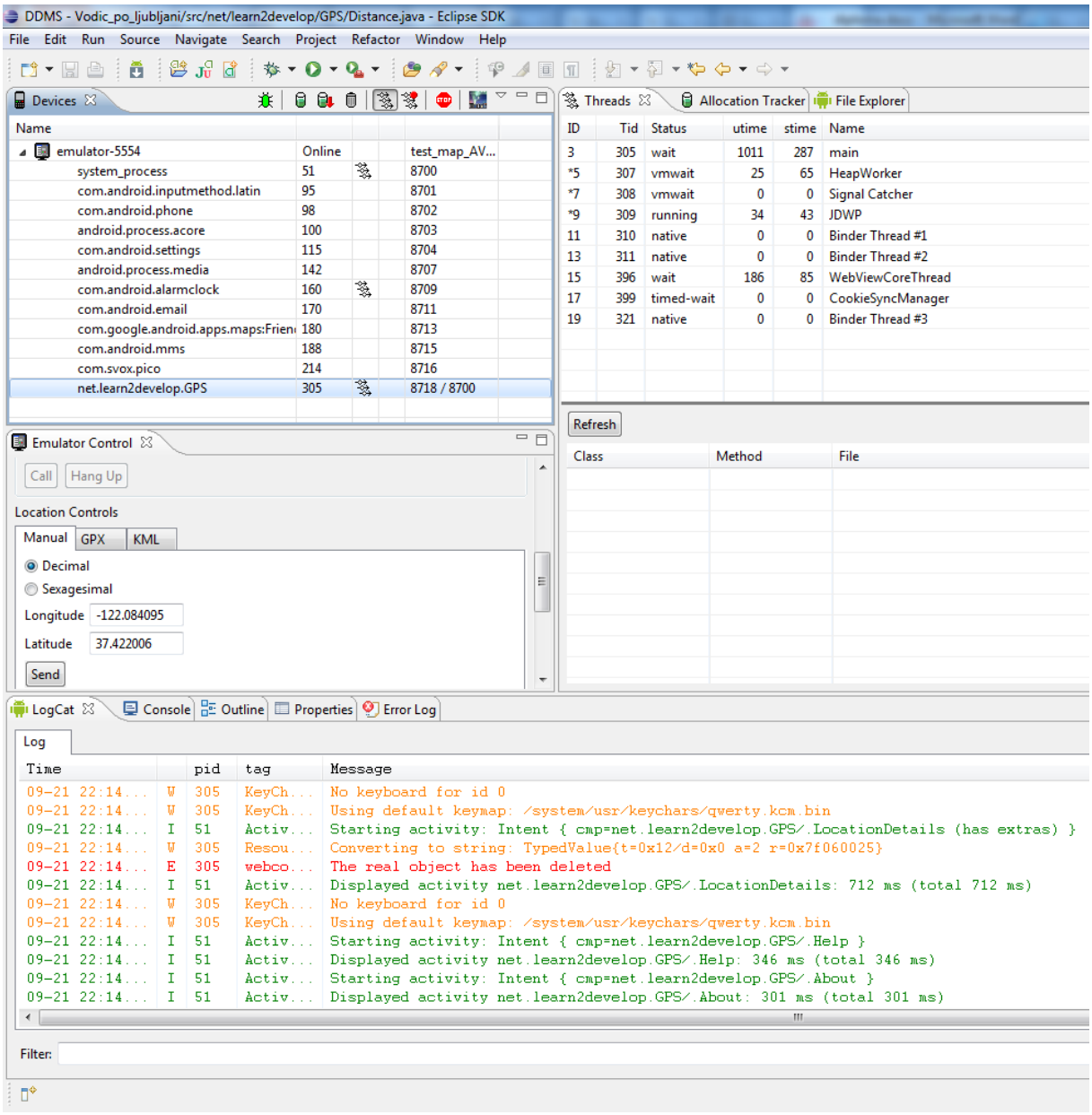
Pomemben del emulatorja je tudi konfiguracija virtualne naprave Android Virtual Device (v nadaljevanju AVD). AVD-ji nam omogočajo spreminjati želeno različico Android platforme, vrsto izgleda emulatorja in želeno konfiguracijo strojne opreme naše virtualne mobilne naprave. S tem lahko simuliramo različne fizične mobilne naprave in različice operacijskega sistema Android. Trenutno lahko izbiramo med različicami operacijskega sistema 1.5, 1.6, 2.0, 2.1 in 2.2 ter možnostmi strojne sestave emulatorja (podpora za GPS, kamero, SD kartico, pospeškometer, itd.). Ko je aplikacija nameščena na emulatorju, lahko uporablja storitve Android platforme za povezovanje z drugimi aplikacijami, predvajanja zvoka in videa, dostopa do omrežja, komunikacije z uporabnikom itd. Ker je emulator virtualna naprava, ki posnema fizične naprave, ima tako tudi svoje omejitve. Trenutno nima podpore za:

- opravljanje in sprejemanje dejanskih klicev (lahko jih le simuliramo)
- USB povezave
- zajem slike in videa
- priključene slušalke
- Bluetooth
- zaznavanje stanja baterije
- zaznavanje stanja povezanosti
- zaznavanje vstavljanja SD kartice

2.2.2 RAZHROŠČEVALNIK

Razhroščevalnik se imenuje Dalvik Debug Monitor Server (DDMS), ki ponuja storitev posredovanja vrat, zajem slike ekrana, informacije o procesih, dnevnike, simulacijo prejemanja klicev in SMS sporočil, navidezno spreminjanje lokacije in veliko več. DDMS deluje na emulatorju ali fizični mobilni napravi. Če sta priključeni obe napravi, se DDMS priključi na emulator.

DDMS deluje kot vmesnik med razvojnim orodjem in delujočo aplikacijo na napravi. Vsaka aplikacija teče v svojem procesu, kateri gosti svoj navidezni stroj. Vsak proces posluša DDMS na različnih vratih. Ko se navidezni stroj z aplikacijo zažene, DDMS pridobi identifikator procesa (PID) in se poveže z razhroščevalnikom navideznega stroja in tako lahko poteka komunikacija med DDMS in navideznim strojem. Za vsak navidezni stroj DDMS odpre druga vrata, povezovanje pa se začne na vratih 8600.



Slika 3: Glavno okno DDMS

Na levi strani uporabniškega vmesnika (slika 3) se nahaja seznam vseh priključenih naprav z vsemi delujočimi navideznimi stroji na posamezni napravi (Devices). Na seznamu izbran virtualni stroj je trenutno priključen na DDMS in ves promet teče k njemu (označen je z modro net.learn2develop.GPS).

Vsa ostala okna nam ponujajo obilico uporabnih informacij in orodij. Zelo uporabno orodje je Emulator Control, s katerim lahko simuliramo različna stanja priključene naprave.

- Status telefona – spreminjamo lahko različna stanja omrežja
- Akcija telefona – simuliramo lahko klice in SMS sporočila

- Nadzor položaja – pošiljamo lahko navidezne položaje napravi
  Na voljo imamo tri načine delovanja:
  - Ročno pošiljanje enega položaja
  - Uporaba datoteke GPX, ki opisuje začrtano pot
  - Uporaba datoteke KML, ki opisuje niz posamezne lokacije

2.2.3 ANDROID OGRODJE

Android aplikacije so napisane v programskem jeziku Java. Prevedena javanska koda s potrebnimi viri aplikacije se zapakira v datoteko s končnico .apk. Nastala datoteka je osnova za distribucijo in nameščanje aplikacije na mobilnih napravah. Vse, kar je zapakirano v eni apk datoteki, se smatra kot ena aplikacija.
Glavna značilnost Androida je uporaba delov aplikacije v drugih aplikacijah. V svoji aplikaciji lahko uporabimo prikaz imenika, ki je del že implementirane aplikacije. Del kode za prikaz imenika se ne vključi v našo aplikacijo, ampak aplikacija zažene le del aplikacije za imenik. Sistem mora biti sposoben zagnati aplikativni proces, kadarkoli je zahtevan kateri od delov aplikacije, in zanj inicializirati Java objekt. Za razliko od večine programov na drugih platformah Android aplikacije nimajo samo ene vstopne točke (nimajo *main()* metode). Zato so aplikacije sestavljene iz osnovnih komponent, katere lahko sistem zaganja po potrebi. Obstajajo 4 komponente: aktivnosti, storitve, sprejemniki in ponudniki vsebin.

Aktivnosti (angl. activities)
Aktivnost predstavlja vizualni uporabniški vmesnik. Aktivnost lahko prikazuje seznam možnosti, med katerimi uporabnik izbira, prikazuje slike ali nekaj, kar želimo, da uporabnik vidi. Aplikacija za sporočila ima eno aktivnost za prikazovanje seznama kontaktov za pošiljanje sporočil, druga aktivnost je namenjena pisanju sporočila, ostale aktivnosti pa za pregledovanje starih sporočil ali spreminjanje nastavitev. Čeprav delujejo te aktivnosti kot ena aplikacija, so med seboj neodvisne. Vsaka od aktivnosti je podrazred osnovnega razreda *Activity*.
Aplikacija je lahko sestavljena iz ene ali več aktivnosti. Koliko je teh aktivnosti in kaj so, je odvisno od zasnove same aplikacije. Običajno je ena aktivnost označena kot prva in se prikaže, ko uporabnik zažene aplikacijo. Za premik med aktivnostmi poskrbi trenutna aktivnost, ki zažene naslednjo. Vsaka aktivnost se izriše v svojem oknu. Običajno aktivnost

zaseda celoten ekran, vendar je lahko tudi manjša od ekrana ali se pojavi nad prejšnjimi okni. Aktivnosti se lahko razširijo tudi s pojavnimi okni ali okni z dodatnimi informacijami.

Za vizualno vsebino oken poskrbijo pogledi, ki izhajajo iz osnovnega razreda *View*. Vsak pogled opisuje določen pravokoten del ekrana. Pogledi so organizirani v hierarhični obliki, kjer vsak starš vsebuje razporeditev svojih otrok. Pogledi v listih drevesa pogledov zagotavljajo izris vsebine in skrbijo za interakcijo z uporabnikom na svojem delu ekrana. Pogled lahko izriše sliko in zažene opravilo, ko uporabnik pritisne na sliko. Veliko pogledov je že predpripravljenih z gumbi, tekstovnimi polji, potrditvenimi polji itd. in jih lahko razvijalec uporabi v svoji aplikaciji. Posamezno drevo pogledov se določi aktivnosti z metodo *Activity.setContentView().*

Storitve (angl. services)

Storitev ne zagotavlja uporabniškega vmesnika, vendar teče v ozadju. Storitev teče nedoločen čas in medtem lahko opravlja neko nalogo, na primer predvajanje glasbe med prebiranjem sporočil ali opravi nalogo za aktivnost, ki nekaj potrebuje. Vsaka storitev razširja osnovni razred *Services*.

Poglejmo si primer predvajanja seznama pesmi. Aplikacija za predvajanje glasbe ima eno ali več aktivnosti, na katerih bo uporabnik uredil seznam pesmi in zagnal predvajanje. Ko se uporabnik premakne v neko drugo aplikacijo, bi želel, da se izbrane pesmi izvajajo v ozadju. Zato lahko aktivnost zažene storitev, ki bo v ozadju predvajala izbrane pesmi. Tako dosežemo, da se bo glasba predvajala tudi, ko ne bomo več uporabljali aplikacije za predvajanje glasbe.

Možna je tudi povezava in komunikacija s storitvijo v teku (ali jo zagnati, če še ni). Komunikacija s storitvijo je možna preko vmesnika, ki ga storitev izpostavi. Za primer storitve za predvajanje glasbe bi bile to možnosti predvajaj, ustavi ali prevrti.

Sprejemniki (angl. broadcast receivers)

Sprejemnik je komponenta, ki samo sprejema in reagira na razpršena obvestila. Veliko obvestil izvira iz samega sistema. Možna so obvestila o zamenjavi časovnega pasa, o zamenjavi jezika, o izpraznjenosti baterije, ob prejemu novega SMS sporočila in še veliko drugih. Enako lahko aplikacija pošlje vsem drugim obvestilo o nekem dogodku. Če aplikacija prejme nove podatke, lahko ostale obvesti, da imajo te podatke na voljo. Aplikacija ima lahko

več takih sprejemnikov za vsa obvestila, ki bi bila zanjo pomembna. Vsi sprejemniki razširjajo osnovni razred *BroadcastReceiver*.

Sprejemniki ne prikazujejo uporabniškega vmesnika. Glede na pridobljeno informacijo lahko zaženejo aktivnost ali pa samo uporabijo *NotificationManager*, da opozorijo uporabnika. Opozarjanje uporabnika je možno na veliko načinov. Uporablja se vibriranje telefona, zvočna ali svetlobna opozorila. Najpogosteje pa se uporabnika obvesti z ikono v statusni vrstici, kjer lahko uporabnik prebere sporočilo.

Ponudniki vsebine (content provider)

Ponudnik vsebine poskrbi za določene podatke aplikacije, da so razpoložljivi drugim aplikacijam. Podatki so lahko shranjeni na datotečnem sistemu, v SQLite podatkovni bazi ali kakšni drugi obliki. Ponudnik vsebine razširja osnovni razred *ContentProvider*. S tem dobimo na voljo standarden nabor metod, ki omogoča ostalim aplikacijam pridobivanje in shranjevanje podatkov. Aplikacije ne kličejo neposredno metod ponudnika vsebin, vendar se uporablja razred *ContentResolver*, ki komunicira s ponudnikom.

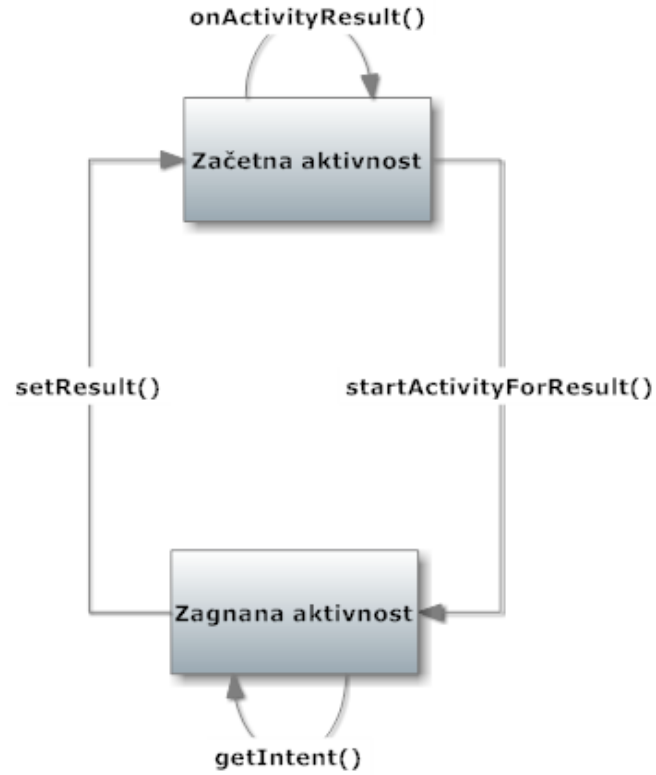
Proženje komponent

Ponudnik vsebine je aktiviran, ko ta dobi zahtevo od komponente *ContentResolver*. Ostale tri komponente pa se prožijo z asinhronimi prožilci (angl. *intents*). Prožilec je objekt tipa *Intent*, ki drži vsebino sporočila. Za aktivnosti in storitve naslovi zahtevano akcijo, določi enotni označevalnik vira (URI) za podatke, nad katerimi naj se izvede akcija, in ostale potrebne podatke. Na primer prožilec lahko proži aktivnost, ki prikaže neko sliko ali dovoli uporabniku vnesti tekst. Za sprejemnike prožilec pove, katera akcija je bila izvedena. Na primer prožilec napove sprejemniku, da je bilo sproženo zajemanje slike iz kamere.

Za proženje ima vsaka komponenta svoje metode.

- Za zagon aktivnosti (ali podajanje zahteve za dodatno naloge) se prožilec posreduje metodi *Content.startActivity()* (slika 4) ali *Content.startActivityForResult() (*slika 5). Zagnana aktivnost lahko pridobi inicialni prožilec s klicem metode *getIntent()*. Aktivnost pogosto zažene novo aktivnost. Če pričakuje odgovor od zagnane aktivnosti, potem kliče novo aktivnost z metodo *Content.startActivityForResult()*. Na primer, če zažene aktivnost za izbor slike, pričakuje nazaj izbrano sliko. Rezultat se vrne preko metode *setResult()* kot prožilec, ki ga prva aktivnost pridobi s klicem metode *onActivityResult()*.

Slika 4: Zagon aktivnosti s startActivity()



Slika 5: Zagon aktivnosti s startActivityForResult()

- Za zagon storitve (ali pošiljanje ukazov zagnani storitve) se prožilec posreduje metodi *Context.startService()*. Podobno kot pri aktivnostih se pri storitvah prožilec poda metodi *Context.bindService()* za vzpostavitve povezave med kličočo komponento in storitvijo. Storitev sprejme prožilec z metodo *onBind()*. Aktivnost na primer lahko vzpostavi povezavo s storitvijo za predvajanje glasbe, da omogoči uporabniku upravljanje predvajane glasbe. Aktivnost z *bindService()* vzpostavi povezavo s storitvijo, nato pa kliče posamezne metode, izpostavljene s strani storitve.

- Aplikacija lahko začne razpršeno oddajanje s posredovanje prožilca metodam kot so, *Context.sendBroadcast()*, *Context.sendOrderedBroadcast()* in *Context.sendStickyBroadcast()*. Vsi zainteresirani sprejemniki sprejemajo prožilec s klicem metode *onReceive()*.

Zaustavljanje komponent

Ponudnik vsebine je aktiven samo dokler odgovarja na zahtevo od *ContentResolver*-ja. Tudi sprejemnik je aktiven, dokler se odziva na sprejeto sporočilo. Obeh komponent ni potrebno

izrecno ustavljati. Aktivnosti vključujejo uporabniški vmesnik in živijo mnogo dlje. Življenjski cikel aktivnosti poteka, dokler se izvaja interakcija z uporabnikom. Tudi storitve lahko živijo dalj časa, zato potrebujemo mehanizem, da ustavimo aktivnosti in storitve.

- Aktivnost zaustavimo s klicem lastne metode *finish()*. Če pa aktivnost zažene drugo aktivnost z metodo *startActivityForResult()*, pa jo lahko tudi ustavi s *finishaActivity()*.

- Storitev lahko ustavimo s klicem njene metode *stopSelf()* ali s klicem metode *Context.stopService()*.

Življenjski cikel komponente

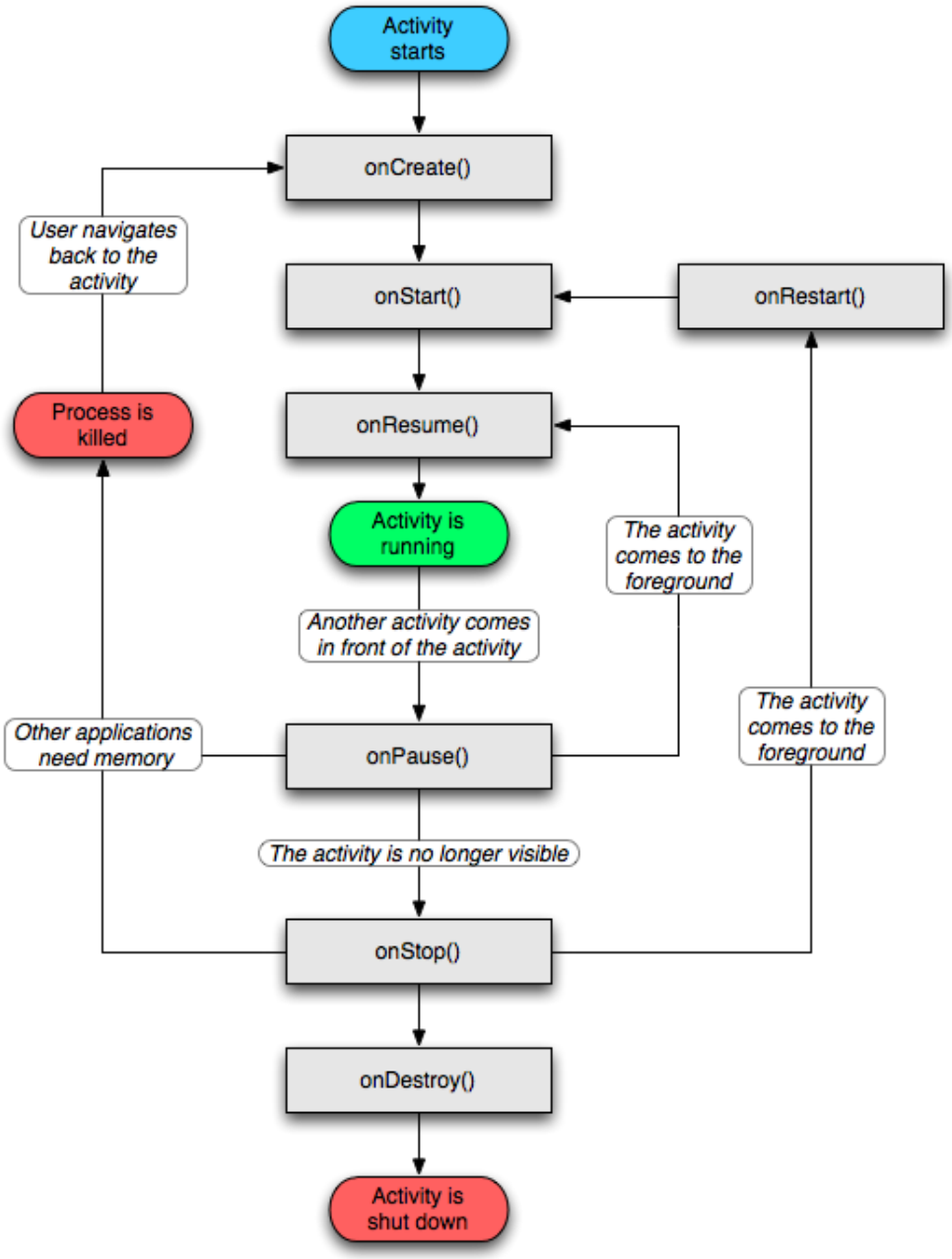
Komponente aplikacije imajo svoj življenjski cikel od zagona komponente s prožilcem do njenega uničenja. V tem času je lahko aktivna ali neaktivna in v primeru aktivnost tudi vidna ali nevidna uporabniku.

Aktivnosti

Aktivnost ima tri stanja.

- Aktivno stanje je, ko je aktivnost na ekranu in na vrhu sklada ter se trenutno uporablja za interakcijo z uporabnikom.
- Prekinjeno stanje je, ko aktivnost zgubi fokus, vendar je še vedno vidna. Nad njo lahko leži drugo okno. Aktivnost je še vedno živa, vendar jo lahko sistem uniči v primeru pomanjkanja pomnilnika.
- Ustavljeno stanje je, ko v aktivno stanje preide druga aktivnost. Še vedno drži svoje stanje, vendar jo lahko sistem uniči glede na potrebe pomnilnika.

Aktivnost v prekinjenem ali ustavljenem stanju lahko sistem pobriše iz pomnilnika s klicem metode *finish()* ali preprosto uniči njen proces. Ob ponovni zahtevi po tej aktivnosti se mora aktivnost ponovno zagnati in obnoviti prejšnje stanje. Prehajanje aktivnosti med stanji lahko ponazorimo z diagramom na sliki 6.



Slika 6: Diagram prehajanja stanj aktivnosti

Glede na diagram imamo tri možne cikle.

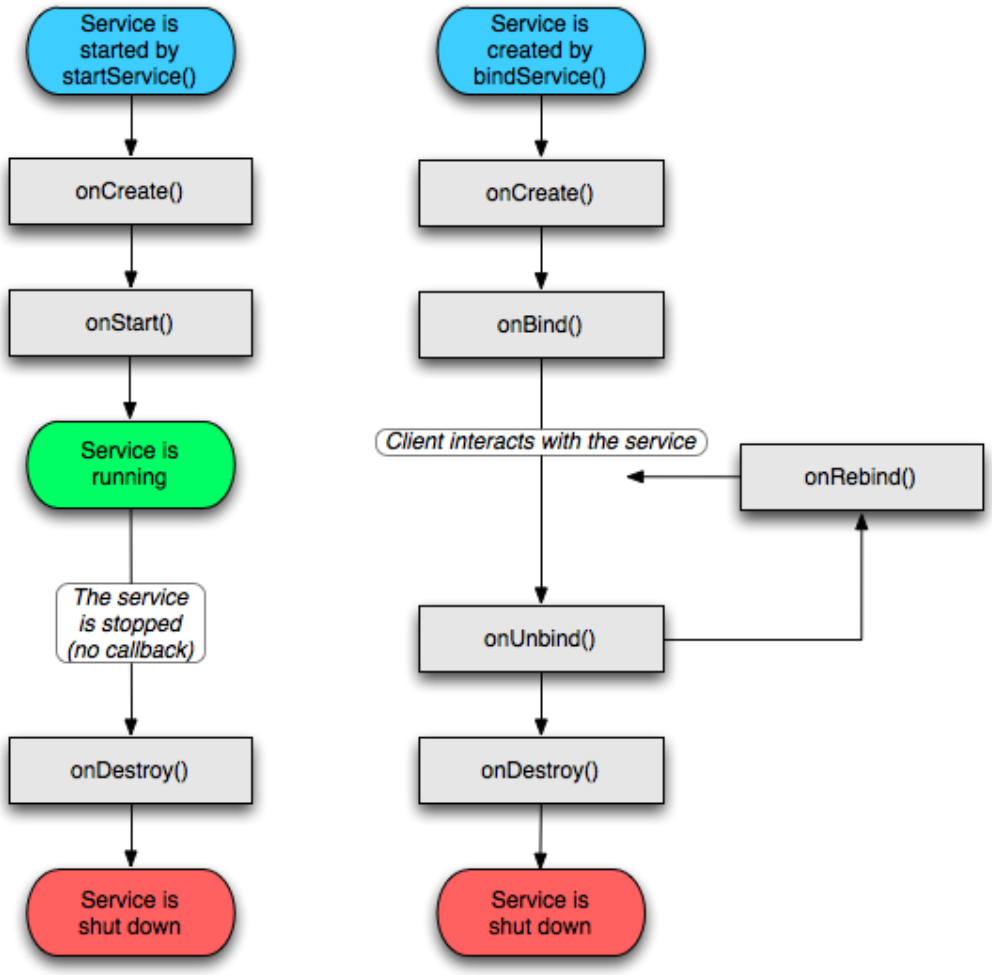
- Celoten življenjski krog se zgodi med *onCreate()* in *onDestroy()*. Vmes se zgodi vse od začetne inicializacije do sproščanja virov na koncu.
- Vidni cikel se zgodi med *onStart()* in *onStop()*. Med tem časom uporabnik aktivnost vidi na ekranu, vendar ni nujno, da je celoten čas v interakciji z uporabnikom.
- Prednostni cikel se zgodi med *onResume()* in *onPause()*. V tem času je aktivnost pred vsemi drugimi aktivnostmi in je v interakciji z uporabnikom.

Storitve

Storitev se lahko uporabi na dva načina.

- Lahko je zagnana z dovoljenjem, da teče, dokler je kdo ne ustavi ali se ustavi sama. V tem načinu se zažene s *Context.startService()* in ustavi s *Context.stopService()*. Sama se lahko ustavi s *Service.stopSelf()* ali *Service.stopSelfResult()*.
- Lahko pa je programsko krmiljena z izpostavljenim vmesnikom. Klienti vzpostavijo povezavo s storitvijo in preko povezave kličejo funkcije storitve. Povezava se vzpostavi s *Context.bindService()* in prekine s *Context.unbindService()*. Z isto storitvijo je lahko povezanih več klientov.

Zgornja načina se med sabo lahko prepletata. Klient se z *bindService()* lahko poveže tudi na storitev, ki je bila zagnana z metodo *startService()*. V tem primeru bo *stopService()* ustavil storitev šele, ko se bo zaključila zadnja povezava. Prehajanje med stanji lahko ponazorimo z diagramom na sliki 7.



Slika 7: Diagram prehajanja stanj storitve

Glede na diagram imamo dva možna cikla.

- Celoten življenjski krog se zgodi med *onCreate()* in *onDestroy()*. Vmes se zgodi vse od začetne inicializacije do sproščanja virov na koncu.
- Aktivni cikel se začne ob klicu *onStart()*. Metoda sprejme prožilec, ki ga je dobila metoda *onCreate()*.

Metodi *onCreate()* in *onDestroy()* sta klicani za vse storitve ne glede na način (*Context.startService()* ali *Context.bindService()*). Metoda *onStart()* pa je poklicana samo, ko je storitev zagnana s *Context.startService()*.

Sprejemniki

Sprejemnik ima samo eno metodo *onReceive()*. Ko pride, prožilec sistem zažene *onReceive()* in sprejemniku poda prožilec. Sprejemnik se smatra kot aktiven, dokler izvaja metodo *onReceive()*. Ko metoda *onReceive()* vrne rezultat, postane sprejemnik neaktiven.

Lokacije in zemljevidi

Aplikacije, ki vključujejo zemljevide in različne oblike lokacij, so za uporabnike zelo privlačne. Take aplikacije lahko gradimo s pomočjo paketa *android.location* in zunanje knjižnice Google Maps.

Lokacijske storitve

Android ponuja aplikacijam dostop do lokacijskih storitev naprave preko razreda v paketu *android.location*. Glavna komponenta paketa je sistemska storitev *LocationManager*, ki zagotavlja programski vmesnik za določitev lokacije ob pogoju, da naprave vsebujejo primerno strojno opremo.

Sistemsko storitev *LocationManager* ne naslavljamo direktno, ampak s klicem *getSystemService(Context.LOCATION_SERVICE)* zahtevamo primerek *LocationManager*-ja. Metoda nam vrne ročico, s katero lahko aplikacija naredi tri stvari.

- Poizveduje po seznamu vseh *LocationProvider*-jev, ki jih pozna *LocationManager* in njihove zadnje znane lokacije.
- Prijavi/odjavi se od periodičnih obvestil o trenutni lokaciji za specifičen *LocationProvider*.
- Prijavi/odjavi prožilec, da se sproži, ko je trenutna lokacija v bližini podane zemeljske dolžine in širine.

Zunanja knjižnica Google Maps

Google je aplikacijam za delo z zemljevidi pripravil zunanjo knjižnico, ki vsebuje paket *com.google.android.maps*. Razredi tega paketa nam omogočajo prenašanje, izris, pomnjenje delčkov zemljevida in veliko prikaznih možnosti ter kontrol.

Glavni razred paketa je *com.google.android.maps.MapView*, ki je podrazred *ViewGroup*. *MapView* prikazuje zemljevid s pridobljenimi podatki iz storitve Google Maps. Ko je *MapView* aktiven, lahko zemljevid premikamo in spreminjamo velikost prikaza. Pri spreminjanju zemljevida se avtomatsko prenašajo manjkajoči delčki zemljevida. Zagotovljeni so tudi vsi elementi uporabniškega vmesnika za upravljanje uporabnika z zemljevidi. *MapView* ponuja tudi lastne metoda za programsko upravljanje z zemljevidi in veliko možnosti za izris prekrivnih elementov preko zemljevida.

Zunanje knjižnica Google Maps ni del standardnih Android knjižnic, zato nekatere Android naprave te knjižnice ne vsebujejo. Tako tudi standarden Android SDK ne vsebuje te knjižnice, vendar pa jo Google ponuja kot dodatek.

Manifest datoteka

Sistem se mora pred zagonom aplikacije seznaniti s komponentami aplikacije. Aplikacija svoje komponente prijavi v manifest datoteki, ki je vključena v paket apk z vso kodo in ostalimi viri. Manifest datoteka je xml datoteka in se v vseh aplikacijah imenuje AndroidManifest.xml. Manifest poleg vseh komponent vsebuje tudi vse knjižnice, uporabljene poleg standardnih Android knjižnic, in vsa dovoljenja, ki jih aplikacija potrebuje. Glavna naloga je obveščanje sistema o vključenih komponentah. Primer spodaj sistemu prijavi aktivnost.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
  <application . . . >
    <activity android:name="com.example.project.FreneticActivity"
          android:icon="@drawable/small_pic.png"
          android:label="@string/freneticLabel"
          . . . >
    </activity>
    . . .
  </application>
</manifest>
```

Xml elementi za prijavo različnih komponent so:

- <*activity*> je za aktivnosti
- <*service*> za storitve
- <*receiver*> za sprejemnike
- <*provider*> za ponudnike vsebin

Aktivnosti, storitve in ponudniki vsebin, ki niso navedeni v manifest datoteki, sistem ne vidi in ne morejo biti nikoli zagnane. Sprejemniki pa so lahko dinamično ustvarjeni preko kode in prijavljeni sistemu prek *Context.registerReceiver()*.

3 STROJNA OPREMA

Danes že več kot 200 različnih naprav uporablja operacijski sistem Android. Največja skupina naprav so mobilni telefoni, med katerimi najdemo nekaj večjih proizvajalcev mobilnih telefonov kot so HTC, Motorola, Samsung in LG. Skoraj vsi telefoni z operacijskim sistemom Android so podobno opremljeni.

- Procesor do 1 GHz
- Delovni pomnilnik do 512 MB
- Podatkovne kartice do 32 GB
- Kamera do 8 MP
- Zaslon na dotik do 4,3"
- Pospeškometer, digitalni kompas
- GPS modul
- Bluetooth modul
- WLAN anteno

Na primeru aplikacije Vodič po Ljubljani prideta najbolj do izraza GPS modul in digitalni kompas.

3.1 GPS MODUL

GPS je sistem globalnega določanja položaja in časa kjerkoli in kadarkoli na Zemlji ali v njeni bližini [6,7]. Deljen je na 3 odseke:

- Vesoljski

  Vključuje GPS satelite, med katerimi je od marca 2008 31 delujočih in 2 odslužena. Potujejo po 6 tirnicah na višini 20200 km in vsak dan Zemljo obkrožijo dvakrat. S tako razporeditvijo iz vsake lokacije na Zemlji v vsakem trenutku vidimo okoli 8 satelitov.

- Nadzorni

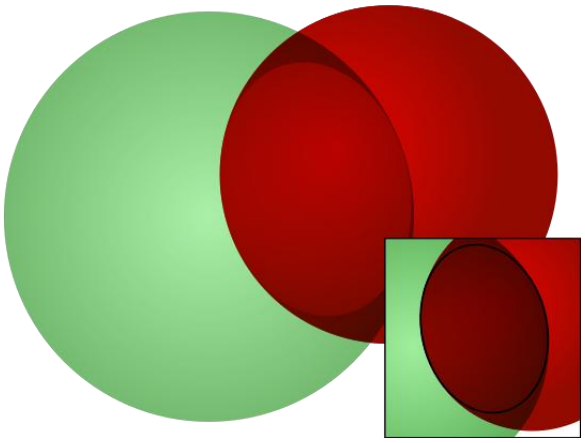
  Vključuje glavno zemeljsko postajo in njeno alternativno lokacijo ter 4 antene in 6 nadzornih postaj. Vse to je potrebno za nadzorovanje satelitov in usklajevanje njihovih ur.

- Uporabniški

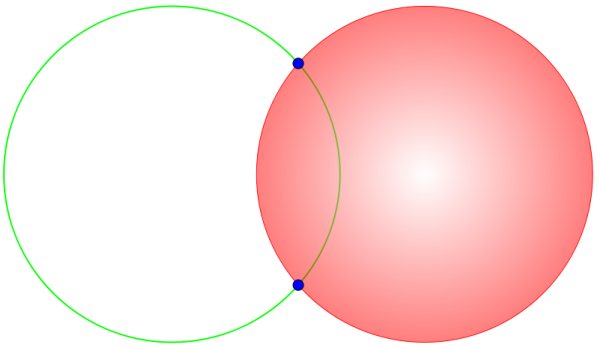
  Uporabniški odsek pa sestavljajo vsi vojaški in civilni GPS sprejemniki.

Večina današnjih mobilnih telefonov že ima vgrajen GPS sprejemnik, ki je vgrajen v osnovno vezje. Osnovna funkcija GPS čipa je sprejemanje radijskih valov satelitov in preračunavanje signalov. Glavne sestavine čipa so antena, mikrokontroler, ojačevalci, filtri, vhodno-izhodne enote in ura. Antena sprejema signale satelitov na frekvenci 1575,42 MHz z zelo velike višine, kar naredi signal zelo nestabilen. Motnje v signalu povzročajo še odboji signala od večjih vodnih površin, stavb in večjih skal. Motnje v signalu lahko naredijo do 25 m napake v natančnosti določitve lokacije. Notranja ura skrbi za čim bolj natančen čas za izračun lokacije, katerega opravi mikrokontroler.

Za izračun lokacije je uporabljena preprosta matematična metoda trilateracija. Gre za presek treh krogel, ki določa vašo lokacijo. Za natančen izračun lokacije potrebujemo lokacije treh satelitov in oddaljenosti od njih. Oddaljenost satelita izračunamo s pomočjo merjenja časa potovanja signala, ki ga oddaja satelit. Ker so radijski valovi elektromagnetno valovanje, potujejo s svetlobno hitrostjo, ki jo množimo z izmerjenim časom in dobimo oddaljenost satelita. Za čim bolj natančno meritev sta ključnega pomena usklajeni uri satelita in notranje ure sprejemnika. Ker so atomske ure izredno drage, so v cenejših GPS modulih navadne kvarčne ure, ki pa se stalno ponastavljajo glede na pridobljen čas s satelita. Sprejemnik zato uporablja uro četrtega satelita in jo uporablja za preračunavanje signalov iz ostalih satelitov. Presek treh krogel sta 2 točki (slika 8 in 9). Ker je le ena točka blizu površja Zemlje, to ni problem pri določanju lokacije na površju. Za določanje lokacije v zraku pa se uporabi četrti satelit za določitev prave lokacije.



Slika 8: Presek dveh krogel (krožnica)



Slika 9: Presek krogle in krožnice (2 točki)

Osnovna funkcija modula je določanje geografskih koordinat in zelo natančnega časa, kar nam zagotovi le podatke, kje smo in kakšen je čas. Popolnoma nove možnosti se odprejo v povezavi z zemljevidi in kompasom. Na tak način lahko določimo pot k točki ali pot po zarisanih točkah. Na ta način delujejo aplikacije za navigacijo in aplikacije z lokacijskimi informacijami.

3.2 DIGITALNI KOMPAS

Digitalni kompas ponavadi pride v paketu s pospeškometrom in sta oba zapakirana v majhen čip. Iz takega senzorja pridobimo podatke o položaju telefona v 3-dimenzionalnem prostoru in pospeške telefona v vse 3 smeri. Te podatke lahko uporabimo na veliko načinov, kar dokazuje nabor aplikacij, ki so danes na trgu. Veliko iger za mobilne telefone si zaradi omejenih kontrol pomaga s temi podatki pri upravljanju z igro. Pri računalniku smo navajeni na tipkovnice ali igralne konzole, kar nam omogoča igranje igre. Pri novejših mobilnih telefonih pa imamo ponavadi samo nekaj gumbov ali pa še tega ne. Če vzamemo na primer vožnjo nekega vozila na mobilnem telefonu, lahko nagib naprej predstavlja dodajanje plina, nagib nazaj zaviranje, nagib levo ali desno pa zavijanje v tej smeri. Te podatke lahko na domiselen način uporabimo skoraj pri vsaki aplikaciji in s tem olajšamo uporabo aplikacije. Poleg tega pa upravljanje igre ali aplikacije s pomočjo premikanja telefona prihrani prostor na ekranu, ki bi ga drugače porabili za prikaz gumbov. Digitalni kompas in pospeškometer sta prinesla v mobilni telefon popolnoma nov način upravljanja aplikacij in s pravim pristopom in načinom uporabe le teh lahko uporabniško izkušnjo močno izboljšajo.

4 RAZVOJ APLIKACIJE VODIČ PO LJUBLJANI

Aplikacija Vodič po Ljubljani je sodelovala v natečaju za mobilne aplikacije na Android mobilni platformi, zato je bil čas razvoja precej omejen. Vse faze razvoja je bilo potrebno izvesti hitro in čimbolj učinkovito. Glavne smernice pri zasnovi aplikacije so bile določene v razpisu natečaja.

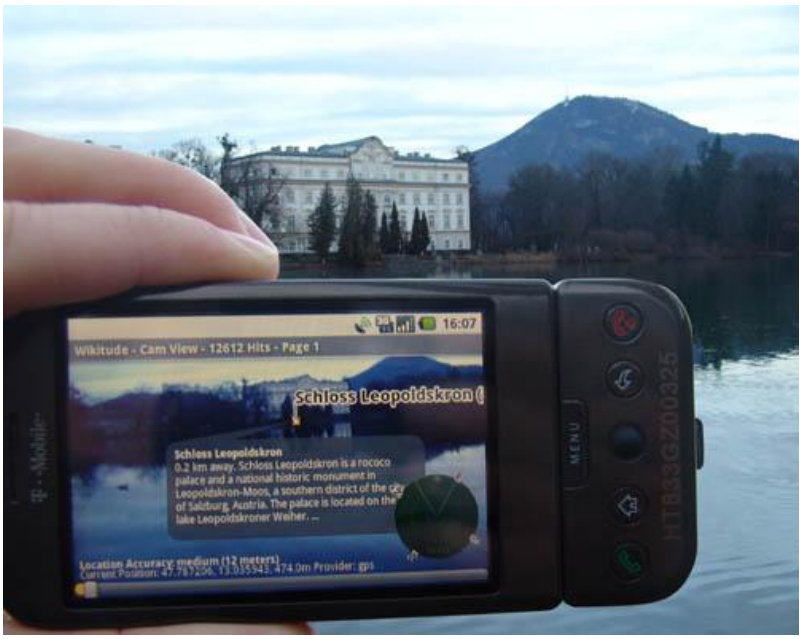
- Lokalna relevantnost
- Izvirnost
- Učinkovita izraba strojne opreme
- Kreativen in sodoben izgled aplikacije
- Uporabnost
- Nadgradljivost

Rezultati natečaja so razkrili 32 prijavljenih aplikacij, med katerimi je Vodič po Ljubljani dosegel odlično 3. mesto. Na natečaju je sodelovalo še nekaj zanimivih aplikacij.

- **Odpiralni časi** – prvouvrščena aplikacija je napredni spletni in mobilni iskalnik za banke, restavracije in ostale storitve v okolici uporabnika.
- **W'sup** – drugouvrščena aplikacija iz spletnih virov shrani prihajajoče dogodke, predstave in filme za kasnejši pregled in lažje načrtovanje aktivnosti uporabnika.
- **Avtocenter** – Aplikacija omogoča iskanje rabljenih vozil in preverjanje možnosti, da je vozilo ukradeno, ali davčnih obremenitev vozila na podlagi VIN številke.
- **Divja odlagališča** – Aplikacija omogoča slikanje, lociranje in prijavo divjih odlagališč.

Po pregledu že narejenih aplikacij je v kategoriji potovanj že precej narejenih aplikacij, vendar pa so se zelo površinsko navezovale na področje Slovenije ali pa sploh ne. Poglejmo si nekatere med njimi [8].

- **Wikitude** – Aplikacija je primerek obogatene resničnosti, ki nam s pomočjo kamere in GPS-a ponuja informacije o naši okolici, ki jih najde med članki Wikipedije. Aplikacija nam sliko s kamere dopolni z najdenimi podatki o naši okolici (slika 10).

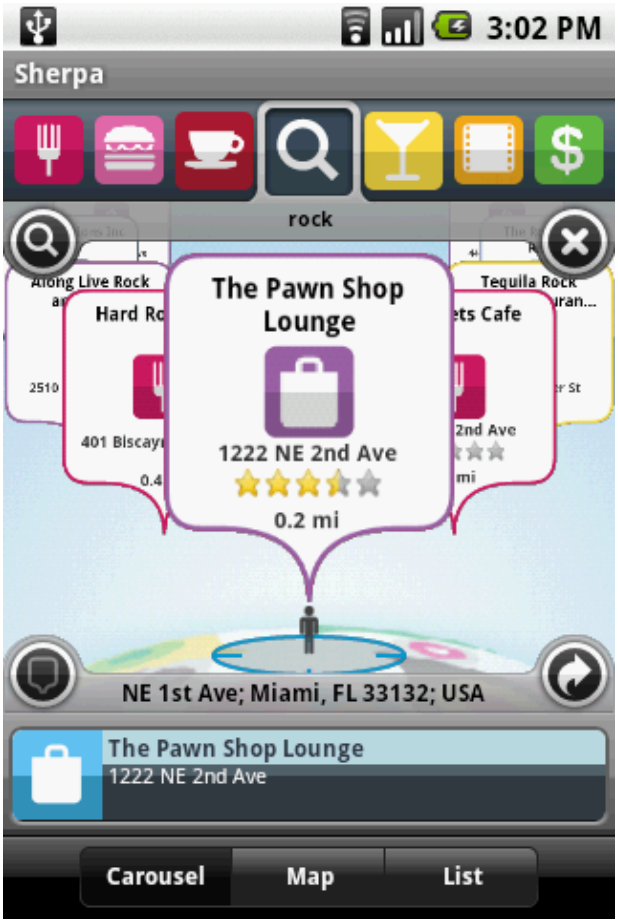


Slika 10: Aplikacija Wikitude

- **Zagat nru** – Aplikacija s pomočjo kompasa in GPS ponuja paleto restavracij, barov, nakupovalnih centrov in zanimivosti v bližini uporabnika (slika 11).

Slika 11: Radar pri aplikaciji Zagat nru

- **Sherpa** – Aplikacija ponuja informacije o okolici uporabnika z poudarkom na kritikah in ocenah interesnih točk in s pomočjo adaptivnega učenja pomaga uporabniku pri izbiri glede na njegove pretekle izbore (slika 12).



Slika 12: Aplikacija Sherpa

4.1 IDEJA

Skupina turistov si običajno v obiskanem mestu ogleda glavne znamenitosti in turistične točke mesta. Za lažji in hitrejši ogled in pridobivanje informacij jim je v pomoč vodič, ki turiste vodi po mestu in jih seznani s pravimi informacijami.

Ideja aplikacije Vodič po Ljubljani je zamenjava fizičnega vodiča z osebnim vodičem na mobilnem telefonu turista. Zelo širok nabor funkcionalnosti mobilnih telefonov lahko ponudi turistu podobno izkušnjo ob ogledu mesta kot s fizičnim vodičem. Oba načina vodenja imata svoje prednosti in slabosti, vendar vam osebni vodič na mobilnem telefonu omogoča večjo svobodo gibanja, zmanjšanje stroškov vodenja in nekatere dodatne opcije, ki si jih težko privoščimo s fizičnim vodičem.

Navigacija po mestu bi potekala prek GPS sprejemnika, digitalnega kompasa in zemljevidov. Tako lahko turist v vsakem trenutku dobi informacijo o svoji lokaciji, spremlja smer cilja ali lokacije v bližini. Dodatna informacija bi bila tudi oddaljenost same lokacije glede na pozicijo turista. Za vsako lokacijo bi imel turist tudi pripravljene zanimive opise, delovne čase, cenike in ostale informacije, ki jih turist želi poznati. Vsak opis bi bil podprt tudi s slikovnim gradivom.

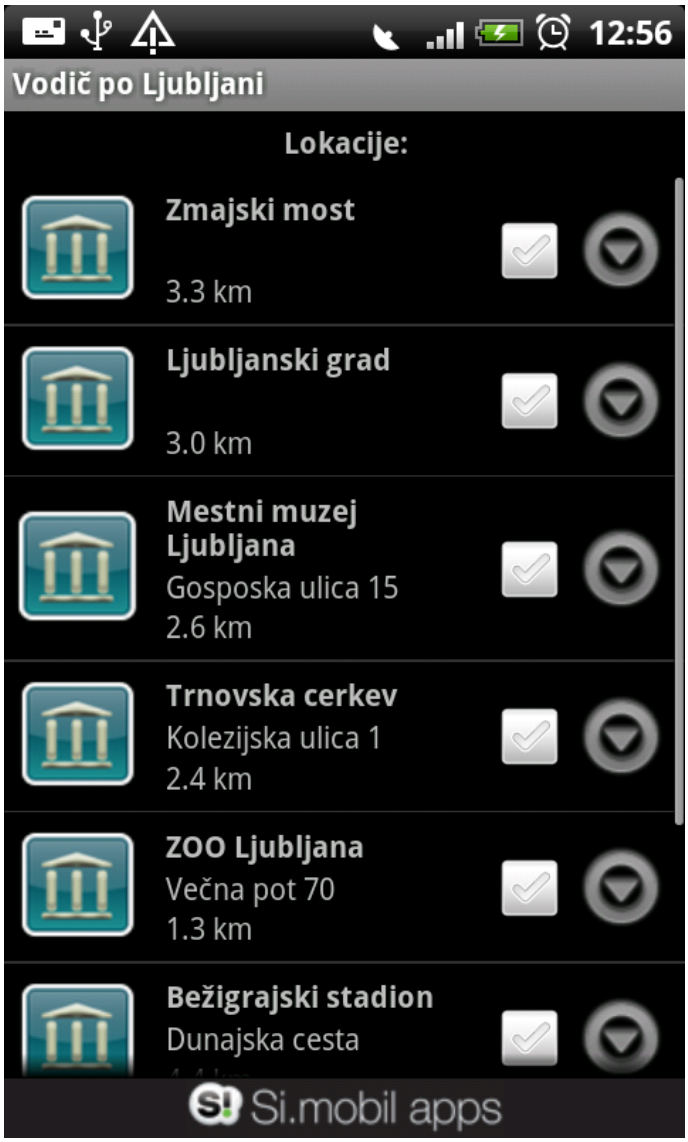
4.2 PREDLOG REŠITVE

Aplikacija bi delovala na mobilni platformi Android od različice 1.5 naprej. Pogoj za uporabo aplikacije je mobilna naprava z nameščenim operacijskim sistemom Android ustrezne različice, internetni dostop, GPS modulom ter digitalnim kompasom. Pogojem zadošča večina današnjih mobilnih telefonov z operacijskim sistemom Android, kar nam nudi zelo širok spekter možnih uporabnikov.
Ciljna množica uporabnikov so vsi domači in tuji obiskovalci Ljubljane, ki posedujejo primerno mobilno napravo. Aplikacijo si uporabniki prenesejo preko uradne spletni strani Android Market ali na kateri od ostalih spletnih strani z Android aplikacijami.

Aplikacija bi delovala v treh načinih delovanja.

- **Seznam lokacij**

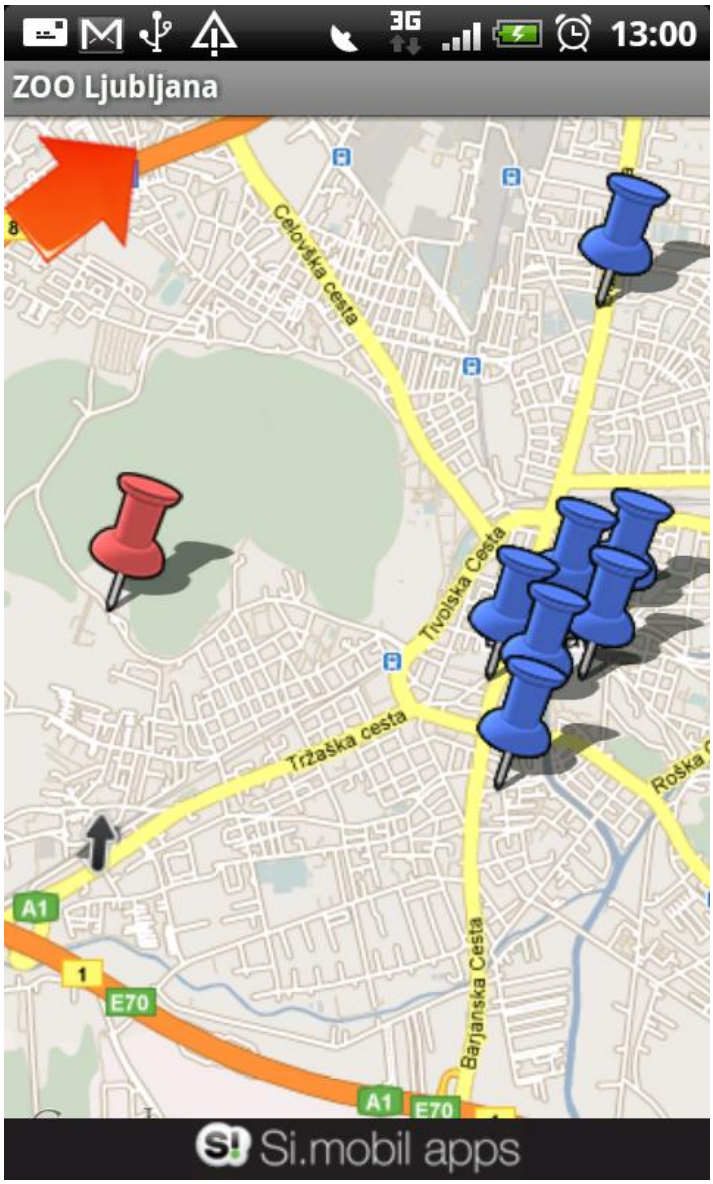
  Vsebuje seznam predpripravljenih lokacij mesta z njihovimi glavnimi podatki (ime in naslov). Ko pridobimo trenutno lokacijo uporabnika, se izračuna tudi oddaljenost posamezne lokacije od trenutne. Na seznamu lokacije tudi pregledujemo in urejamo obiskanost posamezne lokacije. Vidimo, katere lokacije smo že obiskali in jih po želji ponovno odznačimo ali označimo, če lokacijo poznamo od prejšnjih obiskov ali drugih virov. Tukaj lahko tudi pregledamo pripravljeno gradivo lokacij v obliki teksta in slik ter po želji izbrano lokacijo nastavimo kot cilj, kar nam odpre zemljevid z označenim ciljem (slika 13).



Slika 13: Zaslonska maska seznama lokacij

- **Najbližje lokacije**

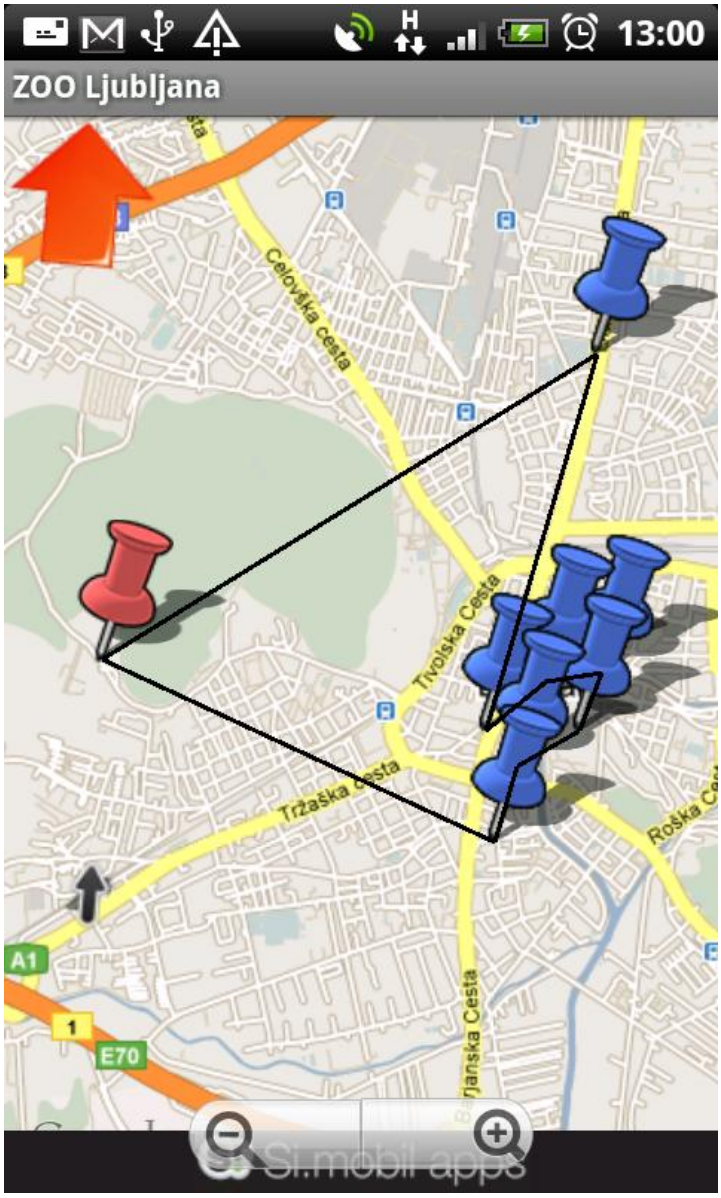
  V tem načinu ima uporabnik na zaslonu zemljevid, ki vsebuje trenutno pozicijo uporabnika in označbe za vse lokacije. Lokacije so različno označene glede na svoj trenutni status, ki se lahko med gibanjem uporabnika po mestu spreminja. Rdeča barva predstavlja najbližjo lokacijo, bela predstavlja obiskane lokacije in modra vse ostale lokacije. Ko se uporabnik prosto giblje po mestu, se lahko spremeni najbližja lokacija, ki jo aplikacija izbira med neobiskanimi lokacijami. Ko je uporabnik v bližini neke lokacije, se mu ponudi tudi možnost pregleda gradiva lokacije (slika 14).



Slika 14: Zaslonska maska najbližje lokacije

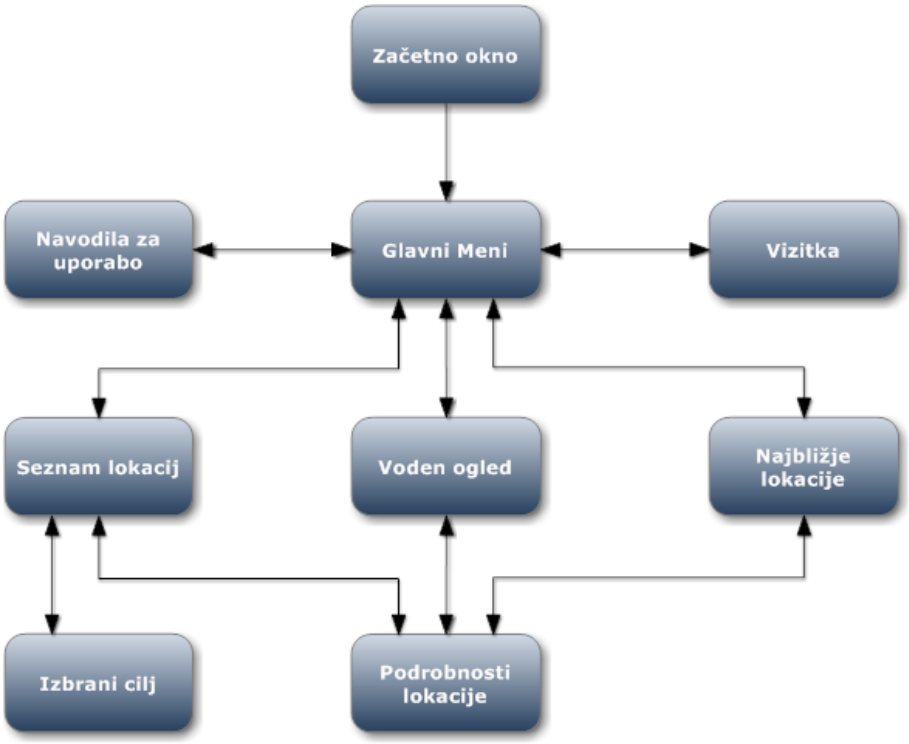
- **Voden ogled**

  Voden ogled predstavlja lokacije, krožno razvrščene po predpripravljeni, poti po kateri uporabnik najhitreje obišče vse lokacije. Začne lahko kjerkoli na začrtani poti in konča na lokaciji pred začetno. Aplikacija glede na trenutno lokacijo uporabnika najde najbližjo lokacijo in jo določi za začetek ogleda. Uporabnik se nato giblje po začrtani poti in si sproti ogleduje znamenitosti. Vmes mu program ponudi za obiskano lokacijo tudi pripravljen gradivo. Ko si je ogledal vse lokacije, se ogled konča. Če si uporabnik ni ogledal vseh lokacij na poti, si aplikacija to zapomni in lahko uporabnik nadaljuje, kjer je zadnjič končal. Lokacije se med ogledom obarvajo glede na njihov status (slika 15).



Slika 15: Zaslonska maska voden ogled

Za pomoč pri navigiranju uporabnika se uporablja še digitalni kompas, ki usmerja uporabnika v realnem svetu. Dodatno se v aplikaciji nahajajo navodila za uporabo same aplikacije in vizitka. Prehodi med zaslonskimi maskami so ponazorjeni z diagramom na sliki 16.



Slika 16: Diagram zaslonskih mask

4.3 RAZVOJ

Priprava razvojnega okolja

Zaradi združljivosti s prejšnjimi različicami je bil uporabljen Android SDK 1.5 na 32 bitnem operacijskem sistemu Windows 7 Professional s prednaloženim Java JDK. Za razvojno orodje je bil uporabljen Eclipse 3.5 z vtičnikom za Android Virtual Devices in dodano zunanjo knjižnico za Google Maps. Za priključitev fizične mobilne naprave na Windowsih je potrebno namestiti še USB gonilnik za operacijski sitem Windows. V kasnejši fazi je bila potrebna še namestitev simulatorja senzorjev SensorSimulator.

Ko smo pripravili razvojno okolje, lahko postavimo nov projekt in začnemo sestavljati aplikacijo. Nov projekt naredimo preko čarovnika in določimo ime projekta, ime paketa, ime aktivnosti, različico Android SDK in minimalno različico operacijskega sistema. Eclipse nam ustvari nov projekt s sledečo strukturo.

src/ - vsebuje vse javanske datoteke naše aplikacije

gen/ - generirane javanske datoteke kot je R.java

Google APIs [Android 1.5]/ - Android in Google Maps knjižnice

assets/ - prazna mapa za vire aplikacije, za katere se ne generira identifikatorjev

res/ - v tej mapi so viri aplikacije kot so slike, teksti, datoteke za postavitev, itd.

AndroidManifest.xml – manifest datoteka naše aplikacije

default.properties – glavne nastavitve projekta

Naša aplikacija je trenutno prazna aktivnost, vendar jo že lahko poženemo. Predpogoj za zagon je priprava emulatorja. Z Android SDK in AVD Manager-jem pripravimo nov AVD z operacijskim sistemom 1.5 z dodano knjižnico Google Maps. AVD- ju dodelimo še potrebno strojno opremo za potrebe naše aplikacije.

- GPS
- Zaslon na dotik
- Senzorji (kompas in pospeškometer)

Če projekt zaženemo, se odpre emulator, kamor se namesti naša aplikacija in se zatem tudi zažene. Trenutno se prikaže prazen ekran, vendar smo s tem preverili pravilne nastavitve in delovanje emulatorja.

Manifest datoteka

AndroidManifest.xml vsebuje vse komponente naše aplikacije in njena potrebna dovoljenja. Glede na diagram zaslonskih mask (slika 16) smo definirali 9 zaslonskih mask, vendar je aktivnosti samo 7. Voden ogled, najbližje lokacije in izbrani cilj se sklicujejo na isto aktivnost, vendar se glede na vhodni prožilec razlikujejo glede na delovanje in izrisano vsebino. Vseh 7 aktivnosti navedemo v manifestu, da jih sistem vidi in jih lahko zažene.

```
<activity android:name=".Splash" android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
<activity android:name=".GPS"/>
<activity android:name=".AllItems"/>
<activity android:name=".Help"/>
<activity android:name=".About"/>
<activity android:name=".LocationDetails"/>
<activity android:name=".MainFrame"/>
```

Poleg navedenih aktivnosti manifest vsebuje še dovoljenja aplikacije. V našem primeru potrebujemo dovoljenje za dostop do interneta in pridobivanje grobe in natančne lokacije prek

GPS modula ali prek ostalih možnosti za pridobivanje lokacij. Dovoljenja se definira kot xml element <*uses-permission*>.

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

Ker pri projektu potrebujemo zunanjo knjižnico za Google Maps, je potrebna definicija še za to knjižnico. Zunanjo knjižnico definiramo z xml elementom <*uses-library*>.

```
<uses-library android:name="com.google.android.maps"/>
```

Ostali podatki, ki jih najdemo v datoteki, so še ime aplikacije, ime paketa, različica aplikacije in opredelitev nabora znakov.

4.3.1 PRIPRAVA IN HRAMBA PODATKOV

Prva različica aplikacije vsebuje samo nekaj osnovnih znamenitosti Ljubljane. Za te znamenitosti je bilo potrebno pripraviti sledeče podatke:

- Ime
- Naslov
- GPS koordinate
- Sliko
- Opis
- Avtorja slike in opisa

Imena in naslovi so določeni glede na izbrani nabor znamenitosti, katerih opisi, slike in avtorji tega gradiva so bili z dovoljenjem lastnika spletne strani https://www.visitljubljana.si pridobljeni na tej strani. Za GPS koordinate pa nam je v pomoč funkcija pridobivanja GPS koordinat na Google Maps. GPS koordinate so določene z geografsko dolžino in širino.
Za kreiranje baze je razmeroma malo podatkov, zato so podatki shranjeni v obliki xml datotek. Android projekt ima v mapi /res pripravljene določene podmape, kjer se hranijo podatki pripadajočega podatkovnega tipa. Podmapa /drawable vsebuje vse pripravljene slikovne datoteke v oblikah GIF, PNG, JPEG ali BMP. Podmapa /xml vsebuje poljubne xml datoteke in hrani posamezne lokacije kot xml elemente. Vsaka lokacija vsebuje atribute za vse svoje podatke ali pa referenco na drug vir. Podmapa /values vsebuje xml datoteke s preprostimi podatkovnimi tip kot so nizi in številke. Tukaj so v xml datoteki shranjeni dolgi opisi posamezne lokacije v obliki html.

4.3.2 UPORABNIŠKI VMESNIK

Aktivnost sama po sebi nima uporabniškega vmesnika, ampak potrebuje posebno xml datoteko. V virih aplikacije se nahaja mapa /res/layout, kjer so shranjene xml datoteke z definicijo pogledov in njihove hierarhije ter posameznih gradnikov vmesnika in razporeditev le teh. Posamezno razporeditveno datoteko lahko večkrat uporabimo. Izgled aktivnosti pa lahko spreminjamo tudi programsko, ko se izgled dinamično določa v fazi delovanja aplikacije.

Glavni meni

Prva aktivnost pri uporabi aplikacije je glavni meni. Postavitvena datoteka definira za to aktivnost 5 gumbov v linearni postavitvi z dodanim vertikalnim drsnikom. Tako postavitev dosežemo s spodnjo xml datoteko.

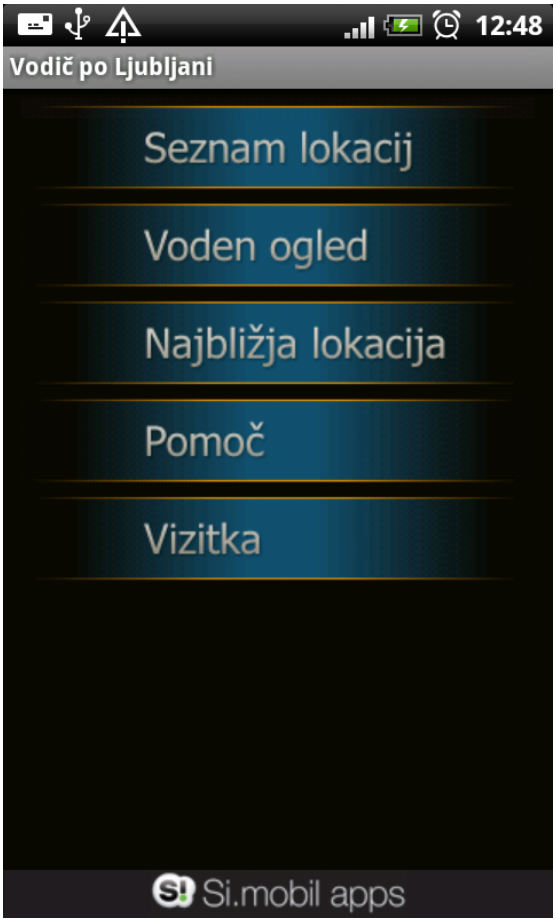
```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView android:id="@+id/ScrollView01"
            android:layout_width="fill_parent"
            android:background="#0e0b06"
            android:layout_height="fill_parent"
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:scrollbars="vertical">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:background="#0e0b06"
              android:gravity="center_horizontal">
<Button android:id="@+id/btnOpenList"
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:focusable="true"
        android:background="@drawable/btn_multi_state_bg"></Button>
<Button ......
</LinearLayout>
</ScrollView>
```

Pogled <*ScrollView*> je oče vseh ostalih elementov in se prikaže samo, ko je vsebina na ekranu večja kot sam ekran. Njegovi atributi določajo njegov identifikator, barvo ozadja, smer drsnika (vertikalna) ter širino in višino, ki v tem primeru napolni celoten ekran. Podrejeni elementi so gumbi <*Button*> v linearni postavitvi <*LinearLayout*>. Linearna postavitev postavi gumbe v vertikalno postavitev (enega pod drugim). Posebnost gumbov je atribut *android:background*, ki s podano xml datoteko opisuje izgled gumba glede na stanje. Za

posamezen gumb sta pripravljeni dve sliki, ki se razlikujeta v nekaj podrobnostih. Prva slika je privzet izgled gumba, druga pa zamenja prvo ob pritisku na gumb. S tem uporabniku jasno prikažemo, kateri izmed gumbov je bil izbran. Tako obnašanje gumba dosežemo s spodnjo xml datoteko.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_focused="true" android:state_pressed="true"
                android:drawable="@drawable/loclisthover" />
    <item android:state_focused="false" android:state_pressed="true"
                android:drawable="@drawable/loclisthover" />
    <item android:drawable="@drawable/loclist" />
</selector>
```

Element *android:drawable* določa sliko za ozadje gumba. Slika *loclist* je privzeta slika, ki pa se ob pritisku gumba zamenja s sliko *loclisthover*. Z elementi *android:state_focused* in *android:state_pressed* določamo pogoje, pod katerimi se ozadje gumba zamenja. Ko kreiramo aktivnost in ji dodelimo postavitveni xml, dobimo sledeči glavni meni (slika 17).



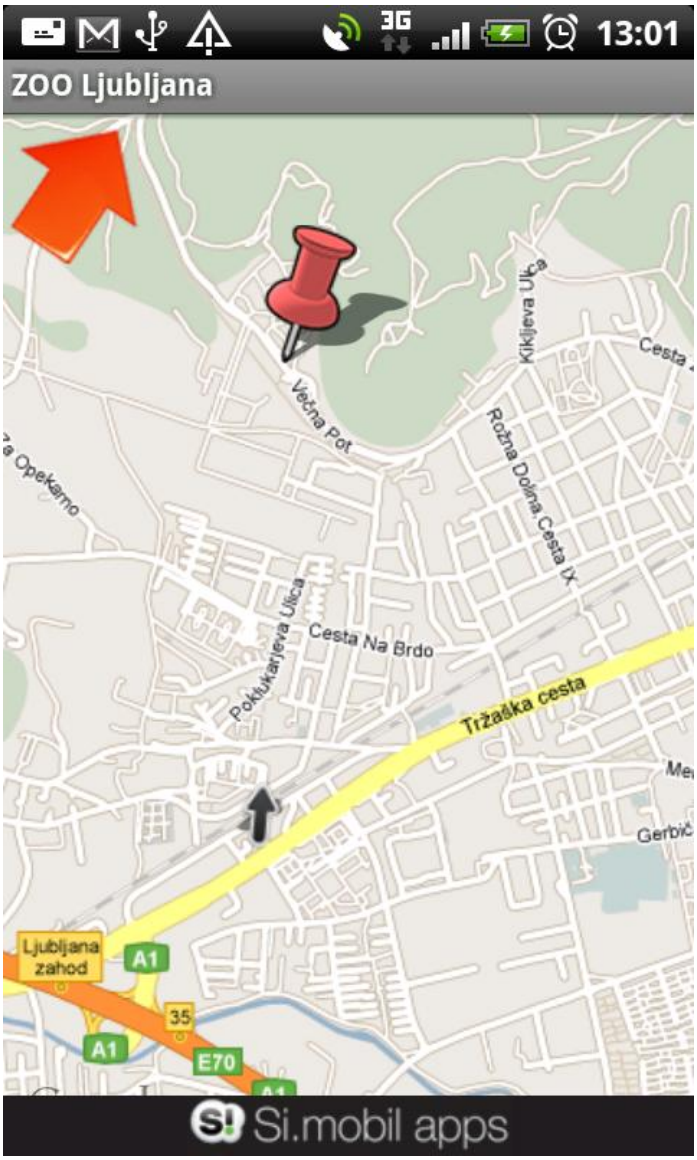
Slika 17: Glavni meni aplikacije Vodič po Ljubljani

Zemljevid

Aktivnosti, ki prikazujejo zemljevid, se od ostalih razlikujejo v postavitveni xml datoteki. Pogled v postavitveni datoteki za aktivnost z zemljevidom je *MapView*, ki zna komunicirati s storitvijo Google Maps in omogoča popolno kontrolo nad upravljanjem z zemljevidom. Spodaj vidimo definicijo za *MapView*, ki je del postavitvene datoteke za zaslonske maske izbrani cilj, najbližja lokacija in voden ogled.

```
<com.google.android.maps.MapView
    android:id="@+id/mapview1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="0P5JB9TFDrrQVu2qFYMY-LDtKXjUizDwwrERDmg" />
```

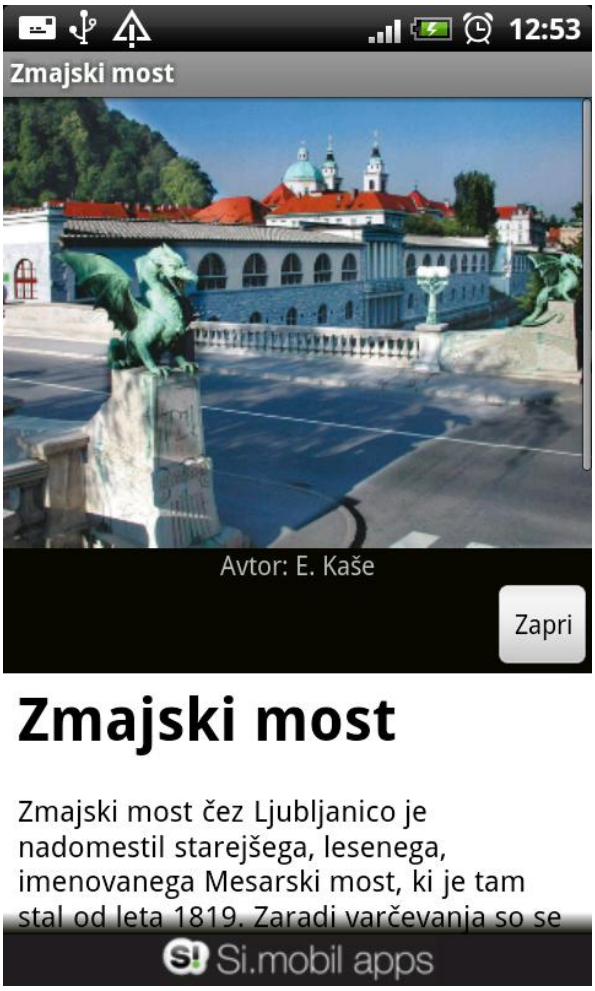
S tem definiramo pogled *MapView* z identifikatorjem *mapview1*, ki prekriva celoten ekran (*android:layout_width="fill_parent"* in *android:layout_height="fill_parent"*). Da lahko uporabljamo kontrole na zemljevidu (premikanje po zemljevidu in povečavo), potrebujemo še element *android:clickable="true"*. Element *android:apiKey* vsebuje ključ, katerega smo pridobili pri prijavi certifikata aplikacije na storitev Google Maps. Če uporabimo napačno kombinacijo certifikata in ključa, ne moremo uporabljati storitve Google Maps, kar se odraža kot prazen siv ekran brez izrisanega zemljevida. Če zemljevidu odvzamemo puščice in buciko, je rezultat pogleda MapView na sliki 18.



Slika 18: Zaslonska maska za vodenje do izbranega cilja

Ostale aktivnosti zgradimo na podoben način z ustreznimi postavitvami in potrebnimi elementi uporabniškega vmesnika. Navodila za uporabo uporabljajo postavitve *RelativeLayout* in *LinearLayout* in pogleda *ImageView* za slike in *TextView* za besedila (slika 19). Podrobnosti lokacije pa uporablja *LinearLayout* s pogledi *ImageView* za sliko znamenitosti, TextView za avtorja slike in *WebView*, ki v sebi drži html opis posamezne znamenitosti (slika 20).

Slika 19: Zaslonska maska podrobnosti lokacije



Slika 20: Zaslonska maska navodil za uporabo

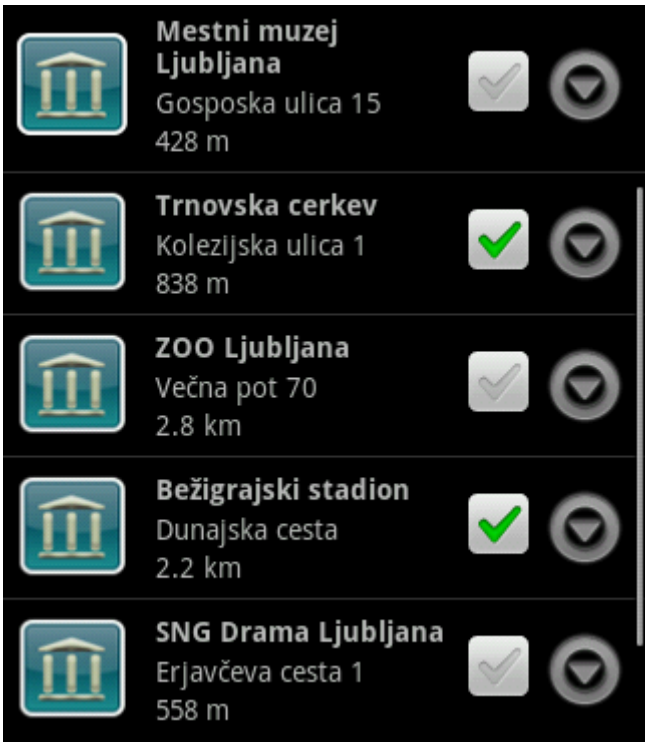
4.3.3 DELOVANJE APLIKACIJE

Vsaka aktivnost potrebuje za delovanje poleg uporabniškega vmesnika še dodatno logiko, ki skrbi za interakcijo z uporabnikom, spremembe na podatkih in uporabniškem vmesniku ter potrebne izračune.

Seznam lokacij

V tem načinu se uporabniku izpiše seznam vseh pripravljenih lokacij. Ob prikazu seznama se izračunajo oddaljenosti posamezne lokacije od trenutne lokacije uporabnika, če imamo na voljo podatek o zadnji znani lokaciji uporabnika, ki jo dobimo z `LocationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER)`. Vzporedno se zažene tudi iskanje trenutne pozicije uporabnika, ker je lahko zadnja znana lokacija uporabnika različna od trenutne. Če podatka o zadnje znani lokaciji ni na voljo, se z izračunom oddaljenosti počaka, dokler ne pridobimo trenutne lokacije. Oddaljenost med

dvema lokacijama tipa *Location* se računa z funkcijo *distanceTo()*, ki vrača oddaljenost v metrih. Če je uporabnik oddaljen več kot 1000 metrov, se izvede še pretvorba v kilometre (slika 21). Poleg imena, naslova in oddaljenosti lokacije je v tema načinu omogočen pregled obiskanosti posameznih lokacij (slika 21). Potrditveno polje je označeno z zeleno kljukico, če je lokacija bila že obiskana.



Slika 21: Seznam lokacij (obiskanost in oddaljenost)

Posamezna vrstica je razdeljena na tri področja, ki ob pritisku nanje izvedejo različno akcijo. Prvi del obsega področje do potrditvenega polja in odpre podrobnosti lokacije (slika 20). Pritisk na potrditveno polje označi ali odznači obiskanost posamezne lokacije. Gumb na koncu vrstice pa nas preusmeri na zemljevid z izbrano lokacijo.

Ob pregledu podrobnosti lokacije se prenese v novo aktivnost preko prožilca identifikator izbrane lokacije, s katerim prikažemo podatke za izbrano lokacijo. Nov prožilec naredimo s sledečimi ukazi.

```
Intent intent = new Intent(this.getContext(),LocationDetails.class);
Bundle bundle = new Bundle();
bundle.putInt("LocationId", o.getId());
bundle.putInt("TourMode", TourModes.SINGLE_LOCATION_MODE);
intent.putExtras(bundle);
int code = 3;
startActivityForResult(intent, code);
```

V naslednjo aktivnost pošljemo identifikator lokacije in identifikator načina delovanja aplikacije, da se prikažejo podatki za izbrano lokacijo. Identifikator načina delovanja je

potreben za vračanje na prejšnja okna. Ker se iz podrobnosti lokacije vedno vračamo na prejšnje okno, odpiramo aktivnost kot *startActivityForResult()*, ker je edina možnost vrnitev na prejšnje okno.

Ob pritisku na gumb za izbiro lokacije kot cilja pa se odpre nova aktivnost z zemljevidom. Sam zemljevid nam ne pove veliko, zato je treba programsko zemljevid opremiti s prekrivnimi elementi (*Overlay*). Pri tem načinu na zemljevid postavimo rdečo buciko kot izbrani cilj in črno puščico kot trenutno lokacijo uporabnika, ki jo usmerimo v smer cilja. Spodnja koda doda na zemljevid rdečo buciko na ciljno točko. Rezultat je prikazan na sliki 18.

```
List<Overlay> mapOverlays = mapView.getOverlays();
Drawable drawable = this.getResources().getDrawable(R.drawable.redpushpin);
MapOverlay itemizedoverlay = new MapOverlay(drawable);
overlayitem = new OverlayItem(point, loc.getName() ,"short desc");
itemizedoverlay.addOverlay(overlayitem);
mapOverlays.add(itemizedoverlay);
```

Zemljevid ima lahko več prekrivnih plasti (vsaka plast ima svoj element). Koda na zgornjem primeru nam iz virov aplikacije pridobi sliko rdeče bucike (*drawable*) in z njo naredimo novo prekrivno plast (*MapOverlay*). Ta plast bo namenjena prekrivanju zemljevida z rdečo buciko. Naredimo nov element plasti *OverlayItem*, ki mu določimo pozicijo, ime in opis. Ta element dodamo v seznam elementov plasti. Samo prekrivno plast pa nato dodamo na seznam prekrivnih plasti zemljevida (List<Overlay> mapOverlays). Ob klicu metode *invalidate()* se vse dodane plasti izrišejo na zemljevidu.

Najbližje lokacije

Ta način nam omogoča pregled lokacij med prostim gibanjem po mestu. Ob vsaki spremembi trenutne lokacije uporabnika se bo osvežilo stanje na zemljevidu. Trenutna lokacija se stalno osvežuje. Z ukazom `LocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 20000, 25,locationListener)` napravo obvestimo, da želimo prejemati obvestila o spremenjeni lokaciji na 20 sekund ali spremembi lokacije za 25 metrov.

Ob vsaki pridobitvi nove lokacije je potrebno ponovno preveriti statuse vseh lokacij. Lokacija je lahko v enem izmed treh statusov (ciljna, neobiskana ali obiskana). Pred osvežitvijo zemljevida se preračuna oddaljenosti lokacij od trenutne lokacije uporabnika in najbližja neobiskana lokacija se označi z rdečo buciko. Vse ostale lokacije, ki še niso obiskane, se označijo z modro buciko. Obiskane lokacije so označene z belo buciko. Tako črna kot rdeča

puščica vedno kažeta v smeri ciljne lokacije. Črna puščica prikazuje smer gibanja glede na zemljevid, zato je izračun kota puščice preprosto izračunati s pomočjo Pitagorovega izreka in kosinusnega izreka s pomočjo trenutne lokacije uporabnika in lokacije ciljne točke. Črna puščica je postavljena na trenutno lokacijo uporabnika in kaže pod predhodno izračunanemu kotu proti ciljni lokaciji. Rdeča puščica pa prikazuje smer gibanja uporabnika proti ciljni lokaciji. Če privzamemo, da uporabnik drži mobilni telefon pred sabo z ekranom obrnjenim navzgor, lahko pridobimo z digitalnega kompasa podatek, v katero smer gleda uporabnik. Da izračunamo kot rdeče puščice oz. pravilno smer gibanja uporabnika, odštejemo kot po digitalnem kompasu od kota črne puščice in po modulu 360 dobimo kot pravilnega gibanja uporabnika. Če je izračunani kot rdeče puščice 0, je gibanje uporabnika v smeri proti cilju. Na tak način se lahko uporabnik usmerja proti ciljni lokaciji.

Voden ogled

V tem načinu delovanja pa se glede na trenutno lokacijo uporabnika izračuna najbližja lokacija, ki postane začetek vodenega ogleda. Ker so lokacije povezane v krog, se za konec ogleda določi predhodna lokacija začetne lokacije. Na isti način kot pri najbližjih lokacijah se izračunavata kota rdeče in črne puščice. Ko uporabnik pride do prve lokacije, se le ta označi za obiskano in se kot cilj nastavi naslednja lokacija na poti ogleda. Obiskanost za voden ogled se hrani ločeno od običajne obiskanosti in nima povezave z možnostjo odznačevanja obiskanosti na seznamu lokacij. Za ponastavitev obiskanosti je pri vodenem ogledu pripravljen meni, ki ga prikličemo s tipko »Menu« na mobilnem telefonu. Za preprost meni uporabimo sledečo kodo.

```
public boolean onCreateOptionsMenu(Menu menu) {
            menu.add(R.string.gps_menu_reset);
            return super.onCreateOptionsMenu(menu);
      }
```

Rezultat te metode se vidi na sliki 22.

Slika 22: Meni za ponastavitev vodenega ogleda (pritisk na gumb »MENU«)

Da pritisk na meni deluje, pa potrebujemo drugo proceduro, ki ponastavi obiskanost vseh lokacij. Procedura preprosto ponastavi začetno in končno točko ogleda in s tem se ponovno določi nov začetek vodenega ogleda.

```
public boolean onOptionsItemSelected(MenuItem item) {
            gpsApp.setFirst_visited(-1);
            gpsApp.setLast_visited(-2);
            UpdateMap();
            return super.onOptionsItemSelected(item);
      }
```

Po končanem ogledu je potrebno obvestiti uporabnika o tem. To preprosto naredimo s proženjem nove aktivnosti, kjer se izpiše obvestilo in nato uporabnika vrne na glavni meni.

4.4 TESTIRANJE

Prvi del testiranja je potekalo na emulatorju in je predstavljal 75% vsega testiranja. Drugi del testiranja pa je potekal na fizični mobilni napravi in je predstavljal 25% testiranj.

Prvi del testiranja na emulatorju je potekal med samim razvojem aplikacije. Po vsakem zaključenem modulu aplikacije se je modul stestiral in odpravile so se ugotovljene napake. Drugi del testiranja na emulatorju se je izvedel po končanem razvoju vseh modulov. S pomočjo orodij, vključenih v Android SDK, in simulatorja senzorjev SensorSimulator lahko pokrijemo testiranje vseh funkcionalnosti aplikacije. Pripravljenih je bilo več AVD-jev z različnimi različicami operacijskih sistemov 1.5, 1.6 ter 2.1 in različnimi resolucijami ekranov (320x480 in 480x800).

Drugi del testiranja je potekal na mobilnem telefonu HTC Magic z operacijskim sistemom Android 1.5 z resolucijo ekrana 320x480. Po testiranju na emulatorju je aplikacija na mobilnem telefonu v prvem poskusu delovala.

Emulator zelo dobro posnema fizično napravo, vendar pa testiranja na emulatorju niso pokazala nekaterih bolj vsebinskih napak. Na ekranu si je bolj težko predstavljati relacijo med severom, usmeritev telefona in samim zemljevidom. Tako so šele testiranja na mobilnem telefonu pokazala, da se smer rdeče puščice ne izračunava pravilno. Pri sprehodu po Ljubljani z uporabo aplikacije so bile izpostavljene tudi druge pomanjkljivosti, ki jih na emulatorju ni bilo zaznati. Pomanjkanje nekaterih informacij na določenih ekranih in napake v algoritmu za voden ogled so se pokazale šele pri testiranju na mobilnem telefonu med sprehajanjem po Ljubljani.

4.5 DISTRIBUCIJA

Po končanem razvoju in odpravljenih napakah, ugotovljenih med testiranjem, lahko aplikacijo pripravimo za objavo. Ker je aplikacija sodelovala na natečaju, je pred objavo na Android Market-u paket z aplikacijo dobila ocenjevalna komisija. Po objavi rezultatov so morale biti vse nagrajene aplikacije objavljene na Android Market-u. Za objavo pa je nujnih nekaj korakov.

- **Podpis aplikacije**

  Vsako aplikacijo, ki jo želimo objaviti na Android Market-u, je potrebno podpisati. Za podpis lahko uporabimo lastne certifikate, ki jih generiramo z orodjem keytool. Certifikat, ki se uporablja pri razvoju aplikacije, je generiran s strani razvojnega orodja. Ker razvojni certifikati uporabljajo privzete podatke, niso primerni za podpisovanje aplikacij, objavljenih na Android Market-u. Vsaka aplikacija potrebuje nov certifikat s privatnim ključem, ki ga pozna samo razvijalec. S produkcijskim certifikatom prevedemo kodo v .apk paket, ki je primeren za objavo.

- **Pridobitev ključa za storitev Google Maps**

  V času razvoja smo storitev Google Maps uporabljali s ključem, vezanim na razvojni certifikat, ki pa ga ne bomo uporabili za distribucijski paket. Ker smo v prejšnjem koraku že pripravili produkcijski certifikat moramo izvesti ponovno prijavo na storitev Google Maps. Za prijavo certifikata na storitev potrebujemo njegov MD5 prstni odtis, ki ga ponovno pridobimo z orodjem keytool. Nato obiščemo spletno stran za prijavo na storitev Google Maps na naslovu http://code.google.com/android/maps-api-signup.html, kamor vpišemo MD5 našega certifikata in pridobimo ključ, ki ga shranimo v Manifest datoteko naše aplikacije.

- **Upravljanje različic aplikacije**

  Uporabnik mora imeti jasno informacijo, katero različico aplikacije ima nameščeno. Tako lahko določi novejše različice aplikacije ali ugotovi, katere posodobitve se nanašajo na njegovo različico. Zato imamo v Manifest datoteki dva elementa, ki vsebujeta podatke o različicah.

  - *android:versionCode* – ta element sprejme numerično vrednost, ki predstavlja različico aplikacije. Upošteva se, da so višje številke novejše različice in nižje

številke starejše različice. Ta vrednost nima povezave z *android:versionName* in se uporabniku ne prikazuje.

- *android:versionName* – ta element sprejme niz znakov, ki predstavljajo različico aplikacije uporabniku. Sistem ga interno ne uporablja in se uporablja samo za predstavitev različice aplikacije uporabniku.

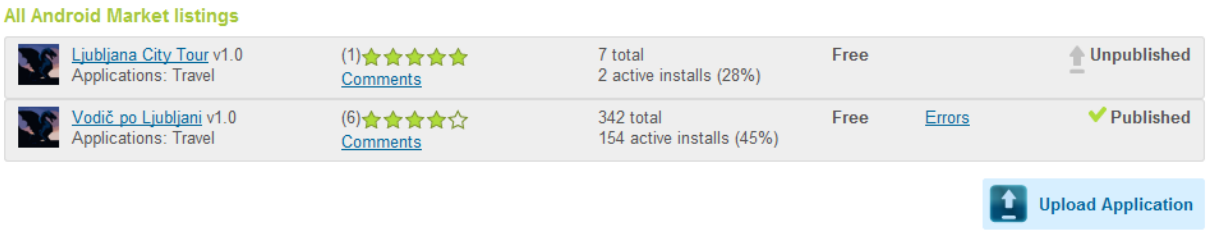
- **Odstranitev razvojnih/testnih podatkov in kode**

  Po končanem razvoju lahko iz Manifest datoteke odstranimo element za razhroščevanje *android:debuggable="true"* ter vso kodo, ki je opravljala naloge logiranja. Poleg tega odstranimo še vse podatke, ki so bili pripravljeni za namene razvoja ali testiranja. V primeru uporabe simulatorja senzorjev SensorSimulator je potrebno nekaj vrstic kodo spremeniti ter odstraniti uvoženo knjižnico. V bistvu je potrebno odstraniti vse, kar za pravilno delujočo aplikacijo ni potrebno in samo zaseda prostor ali upočasnjuje aplikacijo.

- **Končno testiranje na fizični napravi**

  Po vseh predhodnih korakih je čas za pripravo produkcijskega apk paketa. Kodo prevedemo s produkcijskim certifikatom in produkcijskim ključem za storitev Google Maps. Aplikacijo namestimo na mobilni telefon in naredimo končni test. Če vse deluje pravilno, smo pripravljeni na objavo aplikacije na Android Marketu.

Pred odpiranjem računa na Android Marketu si pripravimo nekaj slik ekranov aplikacije in opise aplikacije v jezikih, za katere bomo objavili aplikacijo. Spletno stran Android Marketa najdemo na naslovu http://www.android.com/market/. Odprtje računa za objavo aplikacij stane 25 dolarjev. S tem dobite možnost objave svojih brezplačnih aplikacij, opremljenih s slikami in opisi v več jezikih, na Android Marketu. V primeru plačljivih aplikacij je potrebno odpreti poseben račun (brez dodatnih stroškov) s podatki kot so bančni račun in davčna številka. V primeru plačljivih aplikacija se pri vsaki transakciji zaračuna 30% transakcijskih stroškov. Po odprtju računa dodamo apk z aplikacijo, nekaj slik ekranov ter opise aplikacije. Po objavi je aplikacija takoj vidna vsem mobilnim napravam z operacijskim sistemom Android in lastniku aplikacije (slika 21).



Slika 23: Glavno okno prijavljenega uporabnika na Android Marketu

5 ANALIZA APLIKACIJE

5.1 MOŽNE IZBOLJŠAVE

Predstavljena je bila prva različica aplikacije, ki pa ima še veliko možnosti za nadgradnjo. Nekaj idej za nadgradnjo je povzetih v nadaljevanju.

5.1.1 ROTACIJA ZEMLJEVIDA GLEDE NA GIBANJE UPORABNIKA

V tej različici aplikacije se orientacija zemljevida na telefonu ne spreminja glede na smer telefona, pač pa se uporablja dodatno puščico, ki predstavlja smer cilja v realnem svetu. Izboljšava aplikacije bi bila rotacija zemljevida proti severu ne glede na smer telefona. S tem bi lahko rdečo puščico odstranili, ker bi stanje zemljevida vedno ponazarjalo okolico. Trenutno zemljevidi nimajo podprte rotacije kot jo imajo slike in nekateri drugi pogledi. Domnevno obstaja možnost, da bi bil zemljevid podrejen nekemu drugemu pogleda in bi z rotacijo očeta rotirali tudi podrejeni zemljevid.

5.1.2 RAZŠIRITEV MULTIMEDIJSKE VSEBINE LOKACIJ

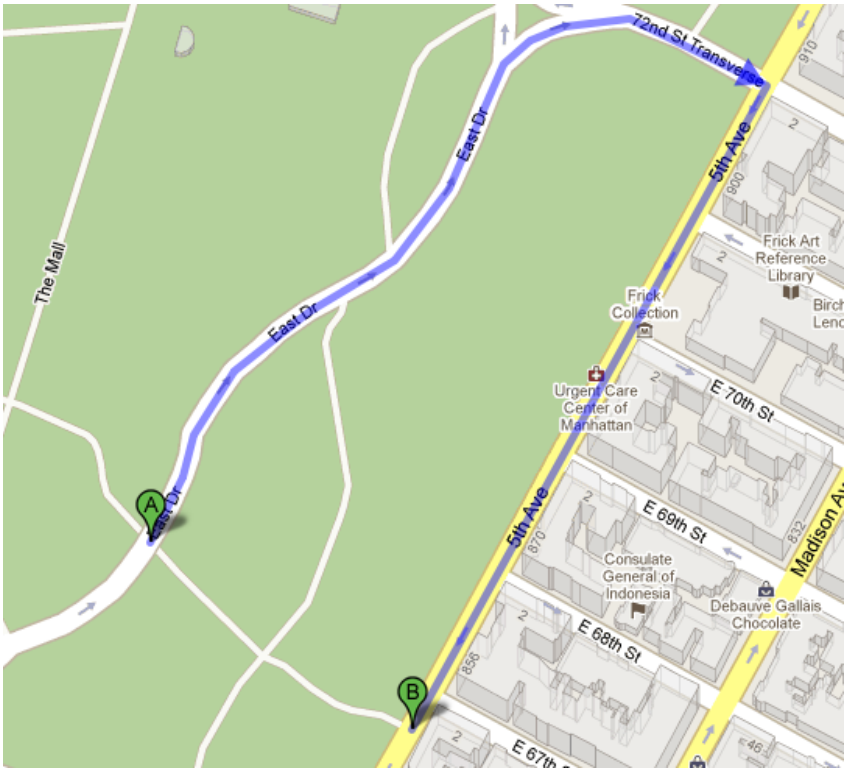
V tej različici imamo za posamezno lokacijo na voljo eno sliko in opis v html obliki, kar zadostuje za osnovno spoznavanje lokacije. Podrobnosti lokacije bi lahko razširili še z galerijo slik ali kratkimi predstavitvenimi filmi. Druga ideja pa je glasovno vodenje pri načinu vodenega ogleda. Vsi telefoni imajo v paketu tudi slušalke, s katerimi bi pripravljeno glasovno gradivo tudi predvajali uporabniku. Na tak način bi lahko uporabnik že med pešačenjem spoznal naslednjo lokacijo preko glasovnega gradiva.

5.1.3 OZNAČITEV POTI NA ZEMLJEVIDU

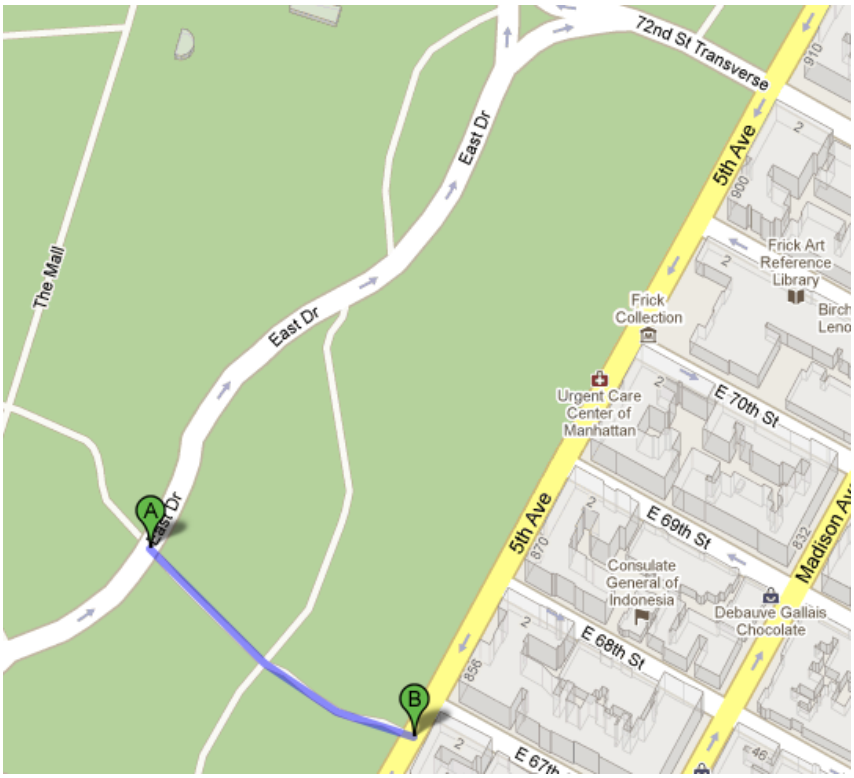
V tej različici so lokacije predstavljene kot točke na zemljevidu. Če je to tudi ciljna lokacija, se puščica obrača v smer cilja. Pri tem načinu uporabnik pozna samo smer in točko cilja, reliefne značilnosti terena in možnih ovir na poti do cilja pa ne pozna.

Rešitev bi bila označena pot na zemljevidu, kot poznamo to iz naprav za navigacijo. Google-ovi zemljevidi že ponujajo funkcionalnost izrisa poti do izbranega cilja. Spodnja slika predstavlja možno pot do cilja z vozilom. Vendar pa se običajno obiskovalci mesta ne vozijo z avtomobili, ampak pešačijo. Google-vi zemljevidi imajo v beta različici že dodatno možnost izračuna poti za pešce, kar bi ob sprehajanju po mestu močno skrajšalo izrisano pot, ker bi sistem poznal vse ulice, kjer je možen prehod samo za pešce in kolesarje. Ker področje

Slovenije še ni pripravljeno za izračune poti za pešce, si poglejmo primer razlike na dolžino poti v New York-u. Če hočemo iz lokacije A na lokacijo B peš, potem nam izris poti za vozila močno podaljša pot. Sliki 24 in 25 predstavljata pot med istima točkama z vozilom ali peš.



Slika 24: Pot od točke A do točke B z avtom



Slika 25: Pot od točke A do točke B peš

5.1.4 INTEGRACIJA S PROGRAMI ZA NAVIGACIJO

V primeru, da se uporabnik aplikacije nahaja daleč od Ljubljane in ima željo obiskati naše mesto in si ogledati znamenitosti, bi bila uporabna tudi integracija z aplikacijami za navigacijo. Ko bi pridobili trenutno lokacijo uporabnika, lahko aplikacija izračuna najbližjo znamenitost v Ljubljani. S tem pridobimo začetno in končno točko potovanja. S temi podatki bi lahko zagnali drugo aplikacijo za navigacijo, kar bi pripeljalo uporabnika na prvo znamenitost. Nato pa uporabnik zapre aplikacijo za navigacijo in nadaljuje ogled mesta s pomočjo Vodiča po Ljubljani.

5.1.5 RAZŠIRITEV VODIČA NA DRUGA MESTA

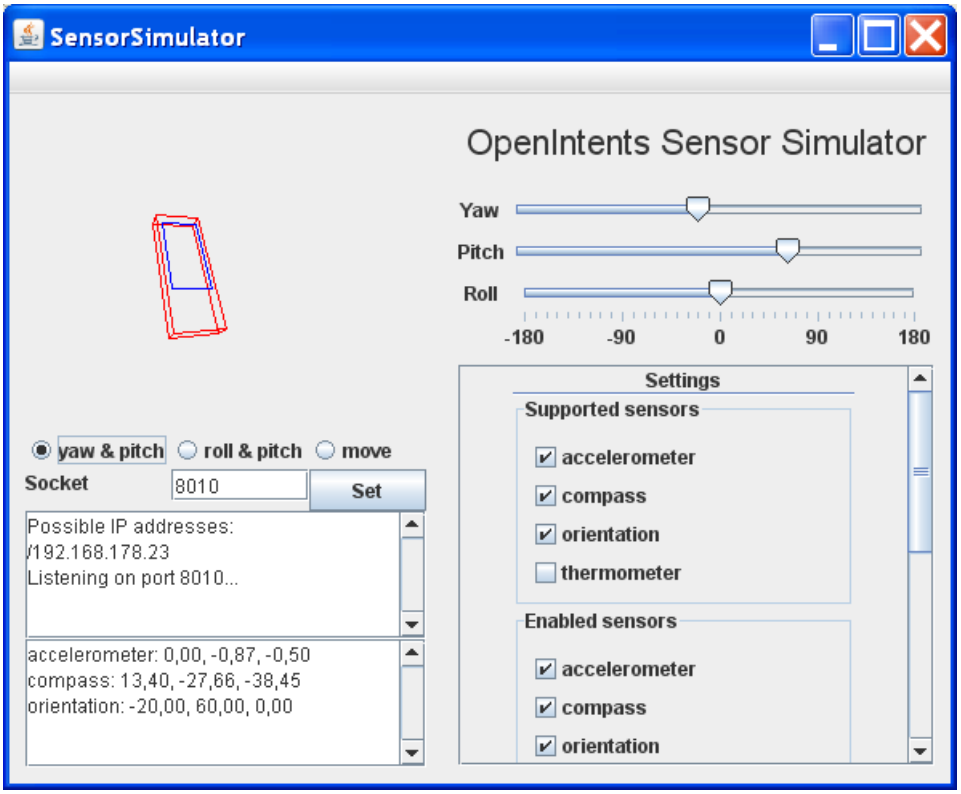
Aplikacijo bi lahko razširili na tudi na ostala slovenska ali tuja mesta. Prva možnost je predpriprava xml datoteke z vsemi lokacijami nekega mesta ter podatki in sliko posamezne lokacije. Z manjšo predelavo aplikacije bi lahko pred prikazom glavnega menija izbirali med mesti za katere imamo na mobilnem telefonu pripravljene podatke. Druga možnost pa bi bila uporabniški vmesnik, ki bi dopuščal ročni vnos posamezne lokacije. Na ta način bi si lahko uporabnik sam ustvarjal lasten izbor lokacij. Če pa združimo oba načina, dobimo možnost izmenjave lokacij med uporabniki. Če bi aplikacija omogočala izvoz podatkov v xml datoteke, bi lahko te datoteke drugi uporabniki preprosto uvozili. S tem bi lahko posamezniki, turistična društva ali drugi pripravljali posebne pakete lokacij za namen kasnejše uporabe ostalih uporabnikov aplikacije.

5.2 TEŽAVE PRI RAZVOJU

5.2.1 TESTIRANJE KOMPASA NA EMULATORJU

Android emulator ima svoje omejitve glede posnemanja delovanja fizične mobilne naprave. Med omejitvami je tudi premikanje mobilne naprave za testiranje digitalnega kompasa ali pospeškometra. Težave so nastopile pri testiranju izračuna kota rdeče puščice za smer cilja v realnem svetu, kjer je najpomembnejši podatek položaj mobilnega telefona. Podatek za kot odklona telefona od severa je v emulatorju vedno 0, kar nam pri testiranju pravilnosti izračuna ne pomaga veliko.

Rešitev problema je bil simulator gibanja telefona SensorSimulator, narejen pri skupini OpenIntents. Simulator nam omogoča simulacijo kompasa, pospeškometra, orientacijskega in temperaturnega senzorja na Android emulatorju. Prvi del simulatorja je aplikacija, ki jo namestimo na emulator. Drugi del je zunanja javanska aplikacija, ki simulira premikanje telefona. Obe aplikaciji povežemo preko IP številke in emulator že lahko sprejema podatke o premikih telefona. Parametre senzorjev nato preprosto prilagajmo v pripravljenem uporabniškem vmesniku z uporabo miške (slika 26).



Slika 26: Glavno okno aplikacije SensorSimulator za simulacijo gibanja telefona

Da uporabimo senzor pri razvoju naše aplikacije, pa je potrebno še nekaj vrstic kode. V samem projektu v Eclipse-u moramo uvoziti zunanji JAR, ki ga dobimo ob prenosu simulatorja in zamenjati upravljavca senzorjev *SensorManager* z našim simulatorjem, kar naredimo s sledečo kodo.

```
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
mSensorManager = (SensorManager) new SensorManagerSimulator((SensorManager)
                 getSystemService(SENSOR_SERVICE));
```

Ker je *SensorManagerSimulator* izpeljan iz *SensorManager*-ja, ima implementirane vse funkcije kot *SensorManager*-ja z dodanima *connectSimulator()* in *disconnectSimulator()*.

5.2.2 SHRANJEVANJE HTML OPISOV LOKACIJ

Pri podrobnostih lokacije je za lep prikaz teksta uporabljen *WebView*, ki je namenjen prikazovanju html strani. Za aplikacijo so bili teksti pripravljeni v html obliki, ki bi se naj prikazovali v tem pogledu. Ob testiranju se je pokazalo, da se html elementi ne prevedejo in ostanejo kot običajen tekst. Kasneje se je izkazalo, da ta pogled sprejme predelan html. Posebne znake '#', '%', '\', '?' je potrebno zamenjati s pripadajočimi kodnimi zamenjavami. Na internetu se najdejo spletni html kodirniki, ki zamenjajo posebne znake. S pomočjo teh kodirnikov so bili vsi opisi pretvorjeni in so se potem tudi pravilno prikazovali.

6 ZAKLJUČEK

Končna različica aplikacije je dostopna na Android Marketu in si jo je do danes namestilo že prek 300 uporabnikov. Prva različica aplikacije je zaradi omejenega časa razvoja samo nakazala smer, kamor bi se ta aplikacija lahko razvijala. Za dodano vrednost uporabniku in širšo uporabo bi bilo potrebno močno razširiti nabor predpripravljenih lokacij in po možnosti dodati katero od idej, ki so predstavljene v poglavju o možnih izboljšavah.

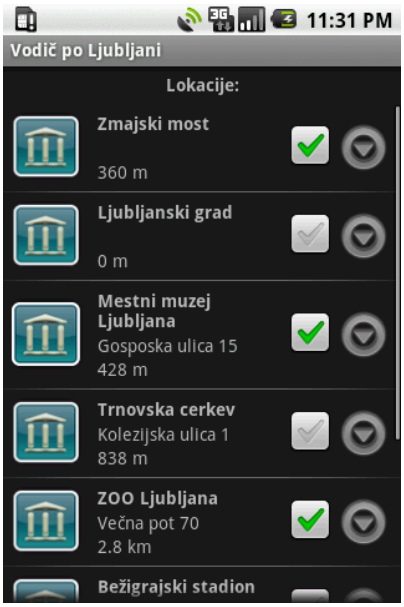
Težave pri razvoju na platformi Android so prisotne kot pri drugih platformah. Reševanje le teh pa je bolj zahtevno, ker je skupnost Android razvijalcev razmeroma majhna in brskanje po internetu pogosto ne razkrije rešitve. Tako stanje lahko pripišemo mladosti same tehnologije, ki še ni doživela tretjega rojstnega dne. V prihodnje pa lahko glede na trende pričakujemo močno širitev kroga razvijalcev in vse več novih aplikacij in uporabnih informacij na internetu.

7 PRILOGE

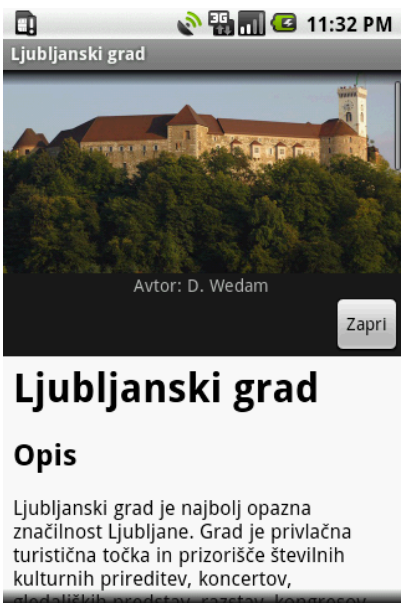
7.1 NAVODILA ZA UPORABO APLIKACIJE

Po zagonu aplikacije se med nalaganjem aplikacije in podatkov prikaže pozdravno okno, ki se po končanem nalaganju spremeni v glavni meni. V meniju uporabnik izbira med gumbi za izbor med tremi načini delovanja: seznamom lokacij, vodenim ogledom in najbližjimi lokacijami ter dvema dodatnima gumboma za navodila in vizitko.

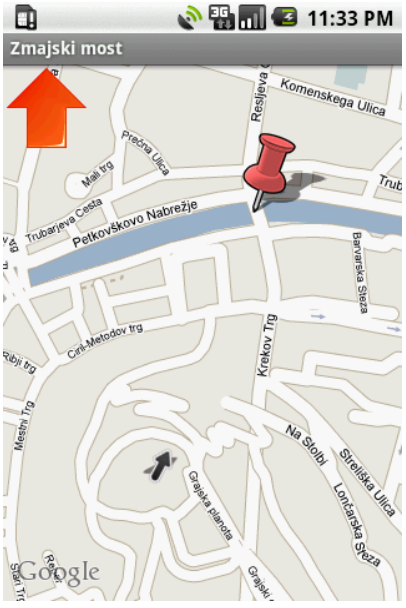
Izbira seznama lokacij prikaže novo okno s seznamom predpripravljenih lokacij z imenom, naslovom, oddaljenostjo uporabnika od lokacije in informacijo o obiskanosti (slika 27). S potrditvenim gumbom spreminjamo obiskanost posamezne lokacije. Gumb na desni strani nam odpre novo okno z zemljevidom in označeno izbrano lokacijo kot cilj (slika 29). Začetek vrstice pa nam na dotik odpre novo okno s podrobnostmi izbrane lokacije. Podrobnosti lokacije nam prikažejo sliko lokacije in tekst z glavnimi informacijami (slika 28).



Slika 27: Seznam lokacij



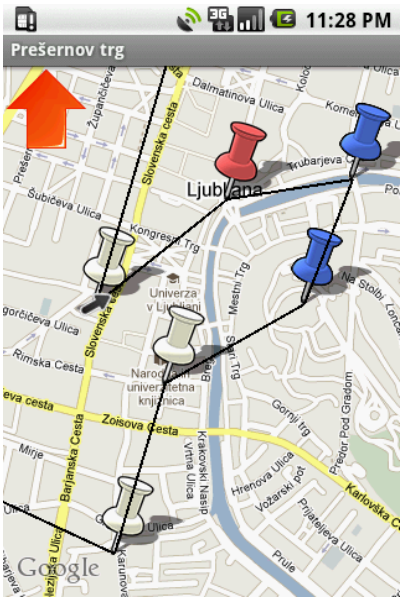
Slika 28: Podrobnosti lokacije



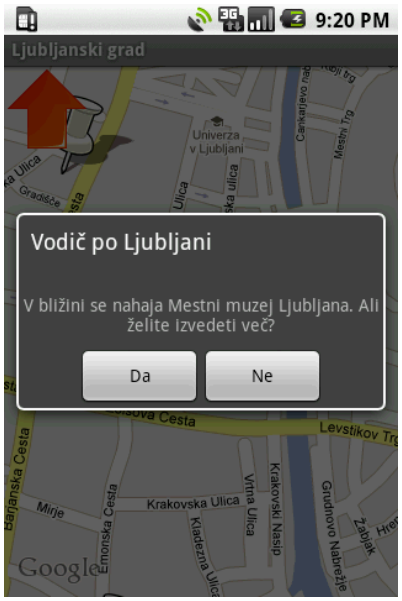
Slika 29: Ciljna lokacija

Izbira vodenega ogleda prikaže zemljevid z lokacijami, povezanimi v pot ogleda. Glede na položaj uporabnika se za začetno lokacijo določi uporabniku najbližjo lokacijo. Zemljevid in rdeča puščica bosta uporabniku pomagala pri poti do ciljne lokacije. Ko se uporabnik približa lokaciji na 25 metrov, bo dobil možnost pregleda podrobnosti lokacije (slika 31). S tem bo lokacija označena kot obiskana in se bo ciljna lokacija premaknila na naslednjo lokacijo na poti ogleda (slika 30). Tako se uporabnik premika po poti ogleda do zadnje lokacije. Če se uporabnik med ogledom odloči prekiniti ogled, se bo stanje vodenega ogleda shranilo in bo

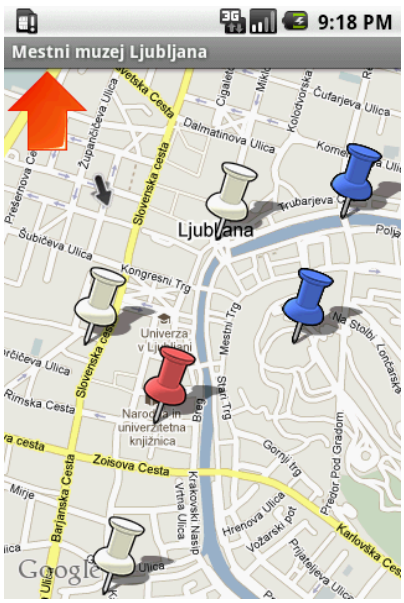
uporabnik lahko ogled nadaljeval ob drugi priložnosti. Po obiskani zadnji lokaciji ga aplikacija premakne nazaj na glavni meni.



Slika 30: Voden ogled



Slika 31: Možnost preusmeritve na podrobnosti lokacije



Slika 32: Najbližje lokacije

Pri izbiri načina najbližje lokacije prikaže aplikacija uporabniku zemljevid, kjer so posamezne lokacije označene glede na njen status (slika 32). Med prostim gibanjem uporabnika po mestu se sproti osvežuje tudi stanje na zemljevidu. Če bo katera od lokacij v radiju 200 metrov, bo uporabnik na to opozorjen in se lahko odloči za obisk lokacije. Ko bo prišel v radij 25 metrov, mu bodo ponujene še podrobnosti lokacije in s tem se bo lokacija označila kot obiskana (slika 31).

Simboli na zemljevidu.

Na zemljevidu označuje trenutno ciljno lokacijo (slika 32).

Slika 33: Rdeča bucika

Na zemljevidu označuje neobiskane lokacije (slika 34).

Slika 34: Modra bucika

Na zemljevidu označuje obiskane lokacije (slika 35).

Slika 35: Bela bucika

Označuje smer gibanja v realnem prostoru. Če se uporabnik giba v pravilni smeri, potem puščica kaže v smeri gibanja uporabnika (slika 36).

Slika 36: Rdeča puščica

Označuje trenutno lokacijo uporabnika in pravilno smer gibanja glede na zemljevid (slika 37).

Slika 37: Črna puščica

Ostali dve možnosti pa prikažeta novo okno z navodili za uporabo aplikacije ali vizitko aplikacije.

7.2 SEZNAM SLIK

8 VIRI

[1] (2010) ITU sees 5 billion mobile subscriptions globally in 2010.
Dostopno na: http://www.itu.int/net/pressoffice/press_releases/2010/06.aspx

[2] (2010) Gartner Says Worldwide Mobile Phone Sales Grew 17 Per Cent in First Quarter 2010.
Dostopno na: http http://www.gartner.com/it/page.jsp?id=1372013

[3] (2010) Android (operating system).
Dostopno na: http://en.wikipedia.org/wiki/Android_(operating_system)

[4] Jerri Ledford, Bill Zimmerly , Prasanna Amirthalingam, *Web Geek's Guide to the Android-Enabled Phone*. Indianapolis: QUE, 2009, pogl. 10.

[5] (2010) Android dokumentacija.
Dostopno na: http://developer.android.com/index.html

[6] (2010) Introduction to the Global Positioning System.
Dostopno na:
http://en.wikipedia.org/wiki/Introduction_to_the_Global_Positioning_System

[7] (2010) GPS.
Dostopno na: http://sl.wikipedia.org/wiki/GPS

[8] (2010) AppBrain.
Dostopno na: http://www.appbrain.com/