Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Pikovnik

Mobilna aplikacija za spoznavanje kulturnih znamenitosti

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

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

Št. naloge: 00218/2012
Datum: 02.04.2012

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

Kandidat: **MATEJ PIKOVNIK**

Naslov: **MOBILNA APLIKACIJA ZA SPOZNAVANJE KULTURNIH ZNAMENITOSTI**

MOBILE APPLICATION FOR LEARNING ABOUT CULTURAL POINTS OF INTEREST

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Zasnujte in razvite mobilno aplikacijo, ki uporabniku prikazuje trenutno sceno preko fotoaparata mobilne naprave. Aplikacija naj bo specializirana za pregledovanje kulturnih znamenitosti in naj skladno s tem uporabniku prikazuje podatke o neki znamenitosti, ki je trenutno na sceni. S pomočjo ogrodja Vaadin naj mobilna aplikacija omogoča pregled kulturne znamenitosti v obliki navidezne resničnosti.

Mentor:

doc. dr. Rok Rupnik

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Pikovnik, z vpisno številko **63080322**, sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za spoznavanje kulturnih znamenitosti

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika,
- 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 10. septembra 2012 Podpis avtorja:

Kazalo

Seznam uporabljenih simbolov in kratic

GPS - Global Positioning System; sistem globalnega določanja lege

MD5 - Message-Digest Algorithm; pogosto uporabljana kodirna funkcija

WiFI - Wireless fidelity; omrežje za povezavo med računalniki brez fizičnega medija

SQLite - Structure Query Language Lite; Podatkovna baza napisana v programskem jeziku C

JSON - JavaScript Object Notation; preprost format za izmenjavo podatkov

SDK - Software Development Kit; zbirke knjižnic za razvoj aplikacij

ARM - Advanced RISC Machine; procesna arhitektura

MIPS - Microprocessor without Interlocked Pipeline Stages; procesna arhitektura

GPS - Global Positioning System; globalni navigacijski sistem

GLONASS - Global Navigation Satellite System; navigacijski siste ruske vojske

NFC - Near field communication; standard za prenos podatkov med mobilnimi napravami

API - Application programming interface; programski vmesnik

HTML - HyperText Markup Language; programski jezik za pisanje spletnih strani

GWT - Google Web Toolkit; ogrodje za izgradnjo bogatih spletnih aplikacij

CSS - Cascading Style Sheets; slogovni jezik za oblikovanje spletnih strani

AR - Augmented reality; navidezna resničnost

URL - Uniform resource locator; edinstven naslov do vira

IP - Internet Protocol; komunikacijski protokol

Povzetek

V diplomskem delu je opisan infromacijski sistem, ki končnemu uporabniku nudi prikaz znamenitosti na mobilnem telefonu, ki ga poganja operacijski sistem `Android`. Sistem je sestavljen iz dveh delov. Prvi del je strežniški, ki zbira podatke jih ureja ter posreduje mobilnim napravam na terenu. Koda za strežniški del je napisana v programskem jeziku `Java` in vsebuje komponente spletnega ogrodja `Vaadin`. Spletna aplikacija teče na serverju `Apache TomCat`.

Mobilni klient predstavlja drugi del informacijskega sistema. Koda mobilnega klienta je napisana v programskem jeziku `Java`, ki teče na pamentih telefonih z operacijskim sistemom `Android`. Klient pridobiva podatke, ki mu jih posreduje strežnik in jih prikaže v uporabniku prijazni obliki. Mobilni klient ima vgrajeno tudi možnost pregleda točk interesa v obliki navidezne resničnosti.

Večina diplomskega dela opisuje izgradnjo obeh aplikacij in podrobnejši vpogled v posamezne dele razvoja. Pri mobilnem klientu je podrobneje opisano tudi delovanje aktivnosti z vgrajeno navidezno resničnostjo.

Ključne besede:

Android, mobilna aplikacija, Vaadin, interesne točke, navidezna resničnost

Abstract

In this diploma thesis is described information system that offers users presentation of points of interest. Whole system is represented by two subsystems. First subsystem is server-side application that collects data and offers it to mobile clients. Code for server-side system is written in programming language `Java` with help of widgets from frame work `Vaadin`. Server collects data from web service for points of interest named `WikiLocation` and weather service `wunderground`. Sever is running on `Apache TomCat`.

Mobile client is second subsystem. Application is build for `Android` smart phones and therefore written in `Java`. Mobile client connects to server and fetches data from it. Application represent data with user friendly presentation, that is very helpful in city tours. Mobile application offers users augmented reality view of interesting points near him.

Diploma thesis describes how system was build, and all the problems that I had to solve while implementing it. In section describing mobile client there is special chapter about augmented reality.

Key words:

Android, mobile application, Vaadin, points of interest, augmented reality

Poglavje 1

Uvod

Pametni mobilni telefoni so v zadnjih letih doživeli neverjeten porast v uporabi med ljudmi. K temu trendu sta zagotovo največ prispevala računalniška giganta `Google` in `Apple`, ki se borita za prevlado na trgu mobilne telefonije. S porastom prodaje mobilnih telefonov pa se je med uporabniki povečala želja za boljše, uporabnejše in še popolnejše aplikacije, ki uporabnikom poenostavijo spremljanje informacij v hitro se spreminjajočem svetu.

Mobilni telefoni predstavljajo enega izmed najbolj množičnih medijev, ki iz dneva v dan raste. Samo Googlov operacijski sistem `Android` naj bi imel v poletnih mesecih leta 2012 dnevno aktiviranih 900000 novih telefonov. To je bil tudi razlog, da sem se odločil svoj informacijski sistem za pregled zanimivih točk napisati posebej za to platformo.

Zamisel za izdelavo informacijskega sistema, predvsem pa mobilne aplikacije sem dobil ob sprehodu po glavnem mestu Slovaške, kjer sem si želel izvedeti več o znamenitostih mesta kar na svojem mobilnem telefonu, pa to zaradi omejenega dostopa do mobilnega omrežja in neobstoju aplikacije ni bilo možno. Mobilni telefoni so za uporabo tovrstnih aplikacij najbolj primerni, saj nam poleg dostopa do svetovnega spleta omogočajo tudi določitev natančne lokacije uporabnika in imajo dovolj velik zaslon, da uporabnik vse predstavljene podatke brez težav prebere.

V drugem poglavju bom natančneje opisal obe razvojni platformi, ki

sem ju uporabil pri izdelavi informacijskega sistema, `Android` in `Vaadin`. Natančneje bodo predstavljene verzije operacijskega sistema `Android` in orodja, ki sem jih potreboval, da sem lahko razvil aplikacijo, ki teče na mobilnih telefonih in na strežniku.

Tretje poglavje je namenjno opisu celotnega informacijskega sistema. Natančneje bom predstavil pridobivanje podatkov iz spletnih virov in urejanje le-teh za prenos na mobilne kliente.

Četrto poglavje vsebuje natančenjši opis aplikacije, ki teče na strežniku `Apache TomCat` in skrbi za prenos podatkov do mobilnih naprav. Poseben del predstavlja integracija `HTTP` servletov v `Vaadin` aplikacijo, ki posredujejo odgovore na zahtevo v `JSON` formatu.

Peto poglavje opiše vse funkcionalnosti in predstavi delovanje mobilnega dela informacijskega sistema. V njem so podrobneje opisane tudi posebne, ne standardne komponente, ki sem jih uporabil pri izdelavi aplikacije. Te kompomente izboljšajo uporabniško izkušnjo in so implementirane tudi v izjemno popularniih aplikacijah, kot so `Facebook`, `Path`, `Twitter`,..

Šesto poglavje je natančen opis implementacije navidezne resničnosti v mobilni aplikaciji. Predstavljen je `SDK`, ki ga je pripravilo nemško podjetje `Metaio`, s katerim sem sodeloval v izdelavi tega informaciskega sistema.

V sedmem poglavju so povzete zaključne misli in možne izboljšave informacijskega sistema.

Poglavje 2

Razvojna platforma

V tem poglavju bom opisal obe razvojni platformi, ki sem ju uporabil za izdelavo končnega informacijskega sistema. Predstavil bom dosedanji razvoj operacijskega sistema Android in spletnega ogrodja Vaadin. Tako za Vaadin, kot Android bom opisal dosedanje verzije in predstavil največje prednosti obeh sistemov, ki so botrovale, da sem ti dve platformi izbral za izdelavo informacijskega sistema.

2.1 Android

Android je mobilni operacijski sistem, ki ga je Google kupil od podjetja Android Inc., leta 2005. Spada v družino operacijskih sistemov Linux, napisan pa je v programskih jezikih:

- `C`
- `C++`
- `Java`
- `Phyton`

Operacijski sistem `Android` teče na pametnih mobilnih napravah, ki morajo podpirati nekatere strojne standarde. Med te standarde spada podpora strojnim platformam:

- `ARM`
- `MIPS`
- `x86`

Naprava, ki želi poganjati operacijski sistem `Android` potrebuje najmanj 128 MB pomnilnika `RAM` in 256 MB zunanjega pomnilnika. Za interakcijo pametnega mobilnega telefona z uporabnikom naprava potrebuje zaslon na dotik.

Med že skoraj obvezne dodatke, ki jih ob nakupu mobilnega telefona z operacijskim sistemom dobite pa sodijo tudi številni senzorji, ki še izboljšajo uporabniško izkušnjo in uporabniku omogočajo uporabo številnih aplikacij.

- Kamera: Android pametni telefoni in tablični računalniki imajo vgrajeno eno ali več kamer.
- `Bluetooth`: Telefoni nudijo podporo za prenos podatkov s pomočjo brezžične tehnologije Bluetooth.
- `GPS/GLONASS`: Senzorja za določanje lokacije telefona.
- `NFC` (angl.*Near Field Communication*): `NFC` je visokofrekvenčna komunikacijska tehnologija kratkega dosega, ki omogoča izmenjavo podatkov na dosegu do 10 cm.
- Merilnik pospeška (angl.*Accelerometer*):To je senzor, ki omogča spremljanje hitrosti premika naprave v tri dimenzijonalnem prostoru.
- Kompas: Kompas je priprava za določanje strani neba.
- Senzor bližnjih objektov (angl.*Proximity*): Senzor, ki omogoča zazavanje bližnjih objektov brez fizičnega dotika.

- Žiroskop (angl. *Gyroscope*): Je senzor, ki spremlja pozicijo telefona v prostoru. Za razliko od merilnika pospeška, žiroskop deluje tudi ko je telefon povsem pri miru.
- `Barometer`: je senzor za merjenje zračnega tlaka.
- `WiFi`: brezžično omrežje

V mobilnem klientu, ki sem ga izdelal za potrebe informacijskega sistema, uporabljam sledeče senzorje:

- `Kamera`
- `GPS`
- `Kompas`
- `WiFi`
- `Omrežni senzor`

Nekateri od teh senzorjev lahko s pomočjo različnih matematičnih operacij opravljajo nalogo drugega senzorja. Primer za tako delovanje je gotovo pridobivanje lokacije.

Lokacijo lahko v moji mobilni aplikaciji pridobimo na dva načina. Prvi način je standardni, za določitev točke nahajanja uporabimo `GPS` senzor. Na ta način pridobimo zelo natančno trenutno lokacijo, vendar pa ima dve pomankljivosti, senzor porabi veliko energije in `GPS` deluje samo na odprtem prostoru.

Drugi način pridobivanja lokacije je s pomočjo senzorja za povezavo v svetovni splet. Določitev lokacije ni tako natančna, vendar uporabniku ni potrebno prižigati `GPS` senzorja in brez težav pridobimo lokacijo tudi, kadar se uporabnik nahaja znotraj zgradbe.

2.1.1 Verzije operacijskega sistema `Android`

Tabla 2.1 prikazuje vse verzije oprecijskega sistem Android do datuma 2.8.2012.

Verzija	Nivo API	Ime
4.1, 4.1.1	16	Jelly Bean
4.0.3, 4.0.4	15	Ice Cream Sandwich Mr1
4.0, 4.0.1, 4.0.2	14	Ice Cream Sandwich
3.2	13	Honeycomb Mr2
3.1.x	12	Honeycomb Mr1
3.0.x	11	Honeycomb
2.3.4		
2.3.3	10	Gingerbread Mr1
2.3.2		
2.3.1		
2.3	9	Gingerbread
2.2.x	8	Froyo
2.1.x	7	Eclair Mr1
2.0.1	6	Eclair 0 1
2.0	5	Eclair
1.6	4	Donut
1.5	3	Cupcake
1.1	2	Base 1 1
1.0	1	Base

Tabela 2.1: Verzije operacijskega sistema `Android`

Aplikacija, ki sem jo izdelal za potrebe informacijskega sistema in teče na pametnih telefonih `Android`, deluje na vseh verzijah opracijskega sistema, ki so novejše od `API` nivoja 8 znane pod prodajnim imenom `Froyo`. Ta verzija je prinesla kar precej sprememb v uporabniško izkušnjo. Med njenimi najopaznejšimi izboljšavami lahko najdemo posodobitev `Linux` je-

dra, možnost prenosa velikih aplikacij iz pomnilnika telefona na SD kartico telefona, omogočanje shranjevanja podatkov v Google Cloud (angl. *Android Cloud to Device Messaging*) in še številnih drugih. Sam sem se za podporo API nivoja 8 odločil zaradi razširjenosti uporabe le-tega.

Tabela 2.2 prikazuje razširjenost verzij operacijskega sistema Android, na podatkih, ki jih je Google zbral v 14 dnevnem časovnem obdobju, ki se je končal s 1.8.2012.

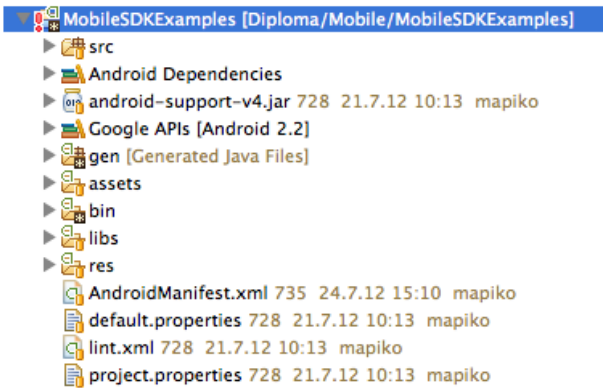
Verzija	Nivo API	Ime	Razširjenost
1.5	3	Cupcake	0.2
1.6	4	Donut	0.5
2.1	7	Eclair	4.2
2.2	8	Froyo	15.5
2.3 - 2.3.2	9	Gingerbread	0.3
2.3.3 - 2.3.7	10	Gingerbread	60.3
3.1	12	Honeycomb	0.5
3.2	13	Honeycomb	1.8
4.0 - 4.0.2	14	Ice Cream Sandwich	0.1
4.0.3 - 4.0.3	15	Ice Cream Sandwich	15.8
4.1	16	Jelly Bean	0.8

Tabela 2.2: Razširjenost verzij operacijskega sistema Android

2.1.2 Struktura projekta Android v Eclipse razvojnem okolju

Ko pričenjamo z razvojem Android aplikacije v razvojnem okolju Android je potrebno najprej spoznati strukturo projekta. Razvoj mobilne aplikacije lahko razdelimo na dva dela. Prvi del je razvoj grafičnega vmesnika, drugi pa razvoj logike in obnašanja aplikacije.

2.1 prikazuje strukturo projekta Android v razvojnem okolju Eclipse.



Slika 2.1: Struktura projekta `Android` v razvojnem okolju `Eclipse`

Struktura gradnikov uporabniškega vmesnika

Za gradnjo uporabnišekega vmesnika uporabljamo jezik `XML` (angl.*Extensible Markup Language*). Vse datoteke, ki vplivajo na izgled in obnašanje uprabniškega vmesnika in niso napisane v programskem jeziku `Java`, se nahajajo v mapi `res`. Datoteke, ki definirajo izgled aplikacije in pozicijo gradnikov se nahajajo v mapi `Layout`. Ker operacijski sistem `Android` teče na mobilnih telefonih z različnimi tipi zaslonov, moramo za vsako gostoto točk (angl.*Screen density*) ustariti poseben izgled. Datoteke za vsako gostoto točk se nahajajo v mapi `drawable-(ime gostote)`. Tako imajo zasloni z resolucijo `hdpi` svoje datoteke v mapi `drawable-hdpi`.

Poleg datotek za izgled imamo možnost v `XML` datotekah prirediti vrednost tudi imenom barv, definirati obnašanje animacij in skrbeti za stile gradniko aplikacije.

Struktura kode in knjižnic

Aplikacije za `Android` so napisane v programskem jeziku `Java`. Vsa koda se nahaja v mapi `src`. Kot vsi javanski projekti, lahko tudi v `Android` aplikacijah dodajomo knjižnice za delovanje. Te knjižnice prenesemo v mapo `libs`. Slika 2.1 prikazuje izgled strukture knjižnic in mape za programsko

kodo v razvojnem okolju `Eclipse`.

Povezava med kodo in gradniki v projektu Android

V sami kodi projekta moramo vse svoje gradnike, med katere spadjo izgledi mask, barve, animacije in ostali v `XML` jeziku definirani elementi, povezati s kodo. Za ta namen `Android` projekt samodejno ustvari `R.java` datoteko, ki se nahaja v mapi `gen`. Elemente deklarirane v `R` razredu pa lahko povezujemo s kodo v naši aplikaciji. Primer povezave preproste labele med datoteko za izgled in programsko kodo je sledeč:

```
textViewTip = (TextView) layout.
findViewById(R.id.textViewTipP);
```

Slika 2.2: Primer povezave med gradnikom uporabniškega vmesnika in programsko kodo

Povezava med izgledom aplikacije in programskim delom mora biti ustvarjena na začetku metode `onCreate`, ki zgradi izgled aplikacije. Datoteko, ki definira izgled in se nahaja v mapi `res/layout` povežemo s kodo na sledeč način:

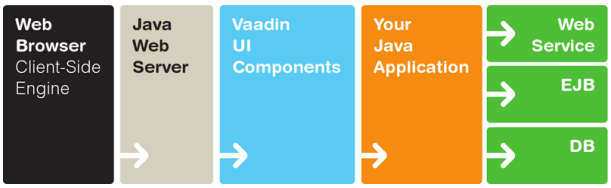
```
setContentView(R.layout.ime_layout_datotke)
```

Slika 2.3: Primer povezave med gradnikom uporabniškega vmesnika in programsko kodo

2.2 Vaadin

`Vaadin` je spletno ogrodje, ki razvijalcu omogoča, da ustvari spletno aplikacijo izključno v programskem jeziku `Java`. Da ustvarimo delujočo spletno aplikacijo ne potrebujemo znanja jezikov kot so `HTML`, `JavaScript` ali `RPC`. Glavni del ogrodja sestavlja `Java` knjižnica, katere namen je razvijalcu omogočiti čim lažje in hitrejše razvijanje zahtevnih spletnih aplikacij.

Slika 2.4 prikazuje osnovno arhitekturo `Vaadin` ogrodja.



Slika 2.4: Struktura projekta `Vaadin`

2.2.1 Arhitektura spletnega ogrodja Vaadin

Arhitektura `Vaadin` ogrodja je sestavljena iz petih osvnovnih elementov.

- Spletni brskalnik
- `Java` strežnik
- `Vaadin` gradniki uporabniškega vmesnika
- `Java` koda aplikacije
- Spletne storitve, podatkovni model,...

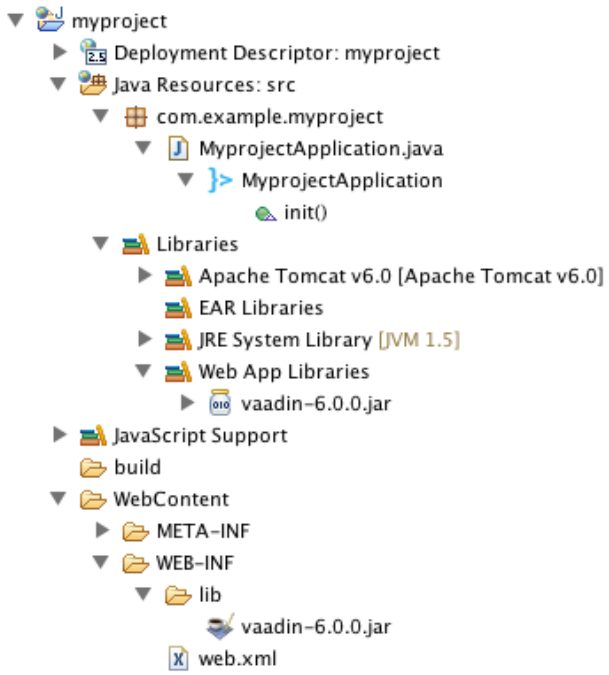
Koda napisana v programskem jeziku `Java` se samodejno prevede v `GWT (Google Web ToolKit)` in `JavaScript` pred izrisom v uporabniškemu brskalniku. To pa nam kot razvijalcem omogoča, da sami napišemo `Vaadin` gradnike s pomočjo `GWT` komponent in jih brez težav vnesemo v svojo `Vaadin` aplikacijo.

2.2.2 Izgled spletne aplikacije Vaadin

Izgled gradnikov aplikacije je poplnoma spremenljiv s pomočjo `CSS` oblikovanja. Celoten izgled aplikacije pa lahko spreminjamo z uvozom tematskih dodatkov, ki jih ponuja Vaadin, kot spletno ogrodje s številnimi možnimi razširitvami.

2.2.3 Struktura projekta Vaadin v Eclipse razvojnem okolju

Kot `Android` projekt ima tudi aplikacija zgrajena s spletnim ogrodjem Vaadin posebno strukturo, ki je precej podobna strukturi `Android` projekta. Slika 2.5 prikazuje izgled strukture `Vaadin` projekta v razvojnem okolju `Eclipse`.



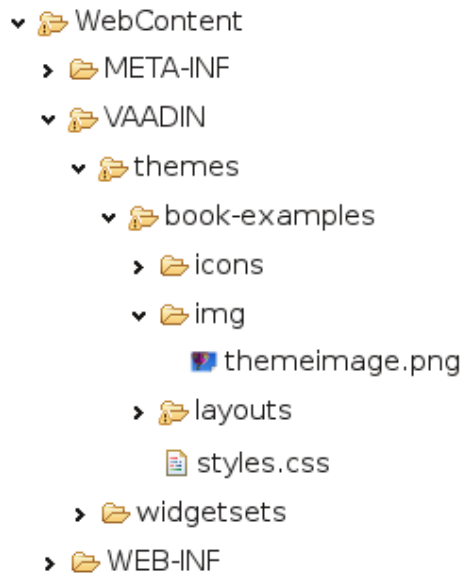
Slika 2.5: Struktura projekta `Vaadin` v razvojem okolju `Eclipse`

Koda, ki jo kot razvijalec projekta napišemo se nahaja v mapi `Java Resources : src`. Vsi ostali elementi aplikacije se nahajo mapi `WebContent`.

Tu notri lahko najdemo tudi mape v katere vstavimo `Java` knjižnice in dodatke v aplikaciji, kot so slike in `CSS` datoteke za izgled.

Struktura gradnikov uporabniškega vmesnika in dodatkov v razvojnem ogrodju Vaadin

Strukturo map v ogrodju Vaadin za grajenje izgleda aplikacije prikazuje slika 2.6.



Slika 2.6: Struktura map v ogrodju za razvoj aplikacij `Vaadin`

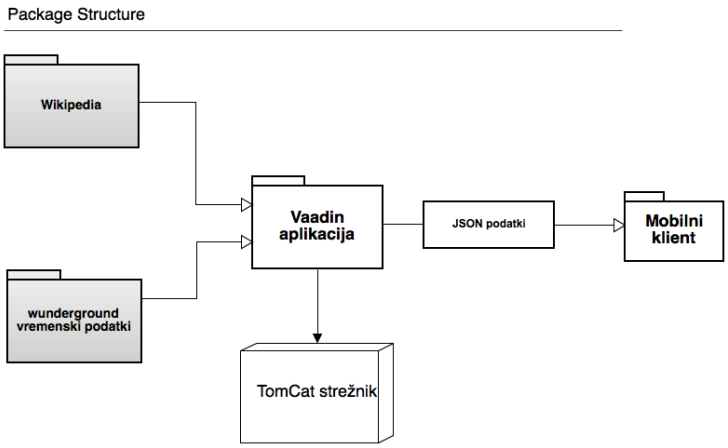
Vse slike, teme in dodatke v aplikaciji tako shranimo v končno mapo `themes`, ki se nahaja v mapi `Vaadin`. Mapa `Vaadin` pa je podmapa `WebContent`. Če želimo svoji aplikaciji prirediti poseben stil gradnikov, to lahko storimo tako, da v mapo `themes` dodamo datoteko `styles.css`.

Poglavje 3

Opis informacijskega sistema

Informacijski sistem "AR vodič po mestu"je sistem sestavlejn iz treh osnovnih komponent. V tem poglavju bom opisal osnovno nalogo vsake izmed njih.

Slika 3.1 prikazuje osnovno shemo delovanja informacijskega sistema.



Slika 3.1: Osnovna shema delovanja informacijskega sistema

Strežniški del prevzema podatke z dveh virov. Pri prvem viru pridobiva podatke o znamenitostih, ki se nahajajo v okolici uporabnika. Drugi vir pa pridobiva podatke o vremenski napovedi glede na lokacijo telefona. Ker spletni servisi ponujajo veliko število informacij in podatkov, ki niso pomembni v mojem informacijskem sistemu jih strežnik uredi v pravilno ob-

liko objektov in šele nato ponudi mobilnim klientom. S tem poskrbimo, da klienti prejemajo samo željene podatke in tako zmanjšamo porabo mobilnega prenosa podatkov, ki zna povročati številne preglavice.

3.1 Vir podatkov o znamenitostih

Strežnik podatke o lokacijah pridobiva s pomočjo spletnega servisa WikiLocation, ki je dosegljiv na http://wikilocation.org/. WikiLocation ponuja razvijalcem podatke za obdelavo s pomočjo arhitekture REST (angl. *Representational state transfer*). WikiLocation je izjemno prilagodljiv spletni servis, saj lahko v pošiljanju zahteve izbiramo med številnimi parametri, ki vplivajo na sam rezultat. Možni parametri so:

- lat - zemljepisna širina (angl. *latitude*)
- lng - zemljepisna dolžina (angl. *longitude*)
- radius - radij
- limit - število pridobljenih elementov
- offset - odmik od začetka prikaza rezultatov
- type - tip podatkov za prikaz. Izbiramo lahko med rekami, mesti,...
- format - format izpisa podatkov. Izbiramo lahko med `XML` formatom in `JSON`
- locale - s katere jezikovne baze pridobivamo podatke, na voljo imamo vse podatkovne baze, ki jih ponuja Wikipedia

Klient izvede `HTTP` zahtevek (angl. *HTTP request*) preko `URL` naslova, ki vrne željene podatke. Uporabnik lahko v mobilnem klientu vse parametre poljubno nastavi in tako prilagaja delovanje aplikacije in količino prenešenih podatkov s svojega pametnega telefona. Primer `HTTP` zahtevka je sledeč:

```
http://tkolitija.javaprovider.net/DiplomaWebApp/
wikiloc?lat=46.0189508&lng=14.8518968
&radius=10000&limit=3&format=json&locale=en
```

Slika 3.2: Primer zahtevka za pridobivanje podatkov o interesnih točkah v okolici

Zahtevek 3.2 ob uspešnem prejemu posredovanih parametrov vrne `JSON` odgovor, ki vsebuje samo podatke, ki jih mobilni klient potrebuje. Izgled uspešno prejetega odgovora prikazuje slika 3.3.

```
{"articles": [
{
"id": "32984683",
"lat": "46.0231",
"lng": "14.8559",
"type": "city",
"title": "Bogensperk, Smartno pri Litiji",
"url": "/index.php?curid=32984683",
"mobileurl": "/index.php?curid=32984683",
"distance": "554m"
}
]}
```

Slika 3.3: Primer odgovora o interesnih točkah v okolici

Odgovor je sestavljen iz glavnega `JSON` objekta "articles", ki vsebuje seznam podobjektov. V našem odgovoru seznam sestavljata dva objekta. Vsak objekt seznama pa vsebuje osem vrednosti, ki so podane točno določenim konstantnim vrednostim (angl.*key-value*).

3.2 Vir podatkov o vremenu

Podatke o vremenu strežnik pridobiva s pomočjo spletnega servisa `Wunderground`, ki je dosegljiv na spletnem naslovu `http://www.wunderground.com/`. `Wunderground` je spletno mesto, ki razvijalcem ponuja `API` (angl.*Application programming interface*) za pregledovanje vremenski napovedi glede na lokacijo. Razvijalec mora pred dostopom do podatkov opraviti registracijo na portal in si s tem zagotoviti ključ, ki je nujno potreben za pridobivanje podatkov s spletnega servisa.

Pošiljanje zahtevka za pridobivanje vremenski podatkov je veliko bolj preprosto, kot pridobivanje podatkov s spletnega servisa `WikiLoc`. Da dobimo željen odgovor moramo servisu `Wunderground` posredovati samo svoj ključ in podatke o lokaciji. Spletni servis vrača številne podatke o vremenski napovedi za željeno lokacjo. Za čiščenje podatkov skrbi moj strežnik, katerega naloga je razbrati pomembne podatke in jih posredovati mobilnemu klinentu.

Da lahko resnično vidimo kolikšno količino podatkov izloči strežnik naredimo `HTTP` zahtevek z enakimi podatki na Wunderground spletni portal in na moj server, ki očisti podatke. Oba zahtevka strežniku posredujeta samo točno lokacijo mobilnega klienta.

HTTP zahtevek na naslov, ki ga lahko vidimo v sliki 3.4 pridobi 8054 bajtov podatkov.

```
http://api.wunderground.com/api/fca9eae8af513f65
/forecast/q/46.0189508,14.8518968.json
```

Slika 3.4: Primer zahtevka za pridobivanje podatkov o vremenu na spletnem servisu `Wunderground`

Zahtevek poslan na moj strežnik z naslovom, ki ga lahko vidimo v sliki 3.5 pa vrne `JSON` datoteko z velikostjo 800 bajtov. Razlika v velikosti pridobljenih podatkov je deset kratna, kar se izjemno pomembno pri mobilni porabi spletnih sredstev.

Odgovor, ki ga vrne moj strežnik si lahko ogledamo v sliki 3.6.

```
http://tkolitija.javaprovider.net/DiplomaWebApp/
weather?lat=46.0189508&lng=14.8518968
```

Slika 3.5: Primer zahtevka za pridobivanje podatkov o vremenu, ki ga vrne moj strežnik

```
{
"weather": [
{
"conditions": "Partly Cloudy",
"icon":"http://icons-ak.wxug.com/i/c/k/partlycloudy",
"address": " 416, 1275 Smartno pri Litiji, Slovenia",
"lowtemp": "15",
"maxtemp": "30",
"date": "1344459600"
}]
}
```

Slika 3.6: Primer zahtevka za pridobivanje podatkov o vremenu, ki ga vrne moj strežnik

Kot lahko vidimo se med rezultati nahaja tudi naslov (angl. *address*) lokacije, na kateri je bila izvedena poizvedba za pridobivanje vremenske napovedi. Sledečo poizvedbo bom opisal v naslednjem podpoglavju.

3.3 Vir podatkov o trenutnem naslovu nahajanja

Informacijo o trenutnem naslovu strežnik pridobi s pomočjo `Google Maps APIja`. `Google Maps API` je spletni servis, ki razvijalcu omogoča, da preko `HTTP` zahtevka z dodanimi parametri pridobi naslov najbližje hišne številke v okolici. Spletni servis je opisan na naslovu `https://developers.google.`

com/maps/documentation/geocoding/.

Kot servis za pridobivanje vremenskih podatkov, tudi geolokacijski servis pridobi preveliko število podatkov, ki niso pomembni za delovanje našega mobilnega klienta. Rešitev je enaka, kot pri pridobivanju podatkov o znamenitostih in o vremenu. Moj strežnik, ustvari HTTP zahtevek na spletni naslov http://maps.googleapis.com/maps/api/geocode/json?latlng=46.0189508,14.8518968&sensor=true, ta pridobi podatke, izloči le pomembne in vrne rezultat preko naslova http://tkolitija.javaprovider.net/DiplomaWebApp/geoloc?lat=46.0189508&lng=14.8518968. Rezultat te HTTP poizvedbe je enovrstični odogovor z naslovom za lokacijo podano s parametri v poizbedbi.

Poglavje 4

Strežniški del

Strežniški del informacijskega sistema predstavlja možgane sistema. Aplikacija, ki teče na strežniku `Apache TomCat` ima sledeče naloge:

- pridobivanje podatkov s spletnih servisov
- urejanje pridobljenih podatkov
- posredovanje urejenjih podatkov mobilnim napravam

Strežniška aplikacija je napisana v programskem jeziku `Java`. Aplikacijo povezuje ogrodje `Vaadin` in v povezavi s spletnimi servisi `Java HttPServlet` omogoča delitev podatkov mobilnim klientom. V razvoju informacijskega sistema sem najprej izvedel načrtovanje in programiranje strežniške aplikacije, ki je osnova za delovanje celotnega informacijskega sistema.

4.1 Vstopna stran spletne aplikacije

Celotna aplikacija je dosegljiva na spletnem naslovu `http://tkoliti.javaprovider.net/DiplomaWebApp`. Ko se uporabnik poveže na prej omenjeni naslov se mu prikaže preprosta maska, ki prikazuje vremensko napoved za prihajajoče dni in 10 najbližnjih znamenitosti.

4.1.1 Opis izgradnje uporabniškega vmesnika

Maska, ki se nahaja na vstopni strani spletne aplikacije je skupek treh komponent, ki sesatvljajo celoten uporabniški vmesnik. Prva komponenta, ki se nahaja na vrhu spletne strani je preprosta labela, ki prikazuje `IP` naslov in trenutno lokacijo uporabnika.

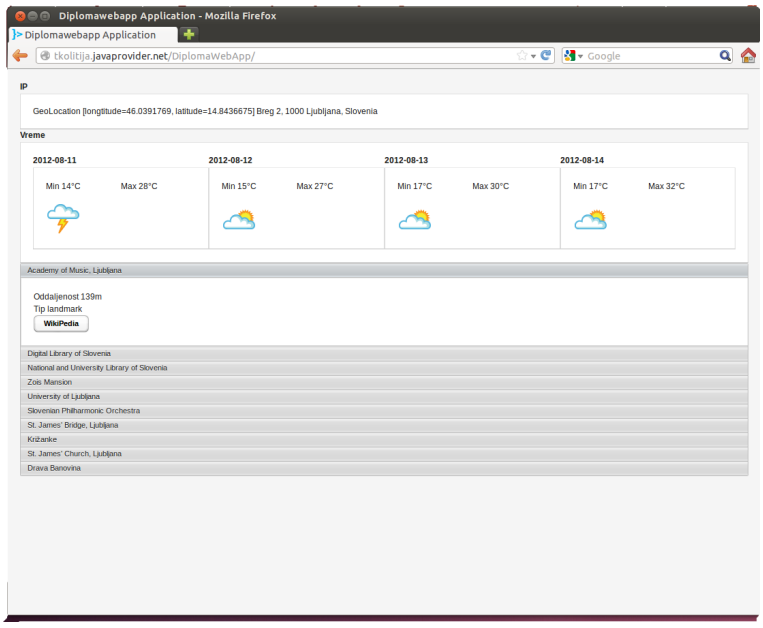
Pod prvo komponento se uporabnuiku prikaže komponenta `VremenskaNapoved`. To je gradnik, ki vsebuje štiri enake podelemente. Podelement gradnika `VremenskaNapoved` se imenuje `VremeZaDan` in je razširitev razreda `Panel`. `VremeZaDan` vsebuje tri labele in gradnik `Embedded`, ki prikazuje sliko vremenske napovedi.

V spodnjem delu zaslona uporabnik vidi deset najbljižih zanimivih lokacij. Lokacije so prikazane na gradniku `Accordion`, ki ob kliku na izbran element prikaže podrobnosti izbrane lokacije. Med podrobnostmi sem izpisal le najnujnejše podatke o lokaciji. Dodal sem tudi poseben element, ki sem ga poimenoval `ButtonLink`. `ButtonLink` element je razširitev elementa `Label`, ki sem mu dodal `CSS` izgled in željene akcije gumba v `HTML` obliki. Slika 4.1 prikazuje programsko kodo gradnika `ButtonLink`.

Izgled spletne aplikacije, ki je dosegljiva na naslovu `http://tkolitija.javaprovider.net/DiplomaWebApp`, prikazuje slika 4.2.

```
public ButtonLink(String caption,
ExternalResource externalResource)
{
super("<a href='" + externalResource.getURL()+
"' target='_blank'>" +
"<div class='v-button' tabindex='0'>" +
"<span class='v-button-wrap'>" +
"<span class='v-button-caption'
style='color:black'>" + caption + "</span>" +
"</span>" + "</div>" + "</a>",
Label.CONTENT_XHTML);
}
```

Slika 4.1: Programska koda gradnika `ButtonLink`



Slika 4.2: Izgled spletne aplikacije

4.1.2 Opis pridobivanja podatkov za prikaz

Pridobivanje podatkov za prikaz na spletni strani je enako, kot pridobivanje podatkov s strani mobilnih klientov. Omenil bi le pridobivanje trenutne

lokacije uporabnika naše spletne strani. Za pridobitev koordinat preko IP naslova sem uporabil spletni servis `http://api.easyjquery.com/ips/?ip=`, ki zaradi varovanja osebnih podatko vrača le celoštevilske vrednosti GPS koordinat. Prav zaradi te omejitve spletna aplikacija deluje kot demonstracija možnosti uporabe informacijskega sistema in ne kot natančna spletna stran.

4.2 Povezava HttpServletov in Vaadin aplikacije

Spletna aplikacija in `HttpServleti` so povezani v datoteki `web.xml`. `Web.xml` je opisnik aplikacie (angl.*Deployment Descriptor*), katerega naloga je povezati `HttpServlete` z glavno aplikacijo in jim prirediti željen dodatek k glavnem spletnemu naslovu. Poleg povezave servletov z aplikacijo, pa so v `web.xml` opisani tudi drugi glavni parametri za delovanje aplikacije. Natančen opis vseh parametrov, ki jih lahko definiramo v spletni aplikaciji je dosegljiv na spletnem na `http://java.sun.com/products/servlet/`.

Za dodajanje našega HttpServleta v aplikacijo pa moramo v web.xml vpisati xml elemente:

- servlet
  - servlet-name
  - servlet-class
- servlet-mapping
  - servlet-name
  - url-pattern

Primer 4.3 prikazuje primer vpisa novega servleta v `web.xml` datoteko.

4.3 Java HttpServlet

`Java HttpServlet` je razred, ki razširja razred `javax.servlet.http.HttpServlet`. Abstraktne metode, ki jih uporabljamo v tem razredu so

```
<servlet>
        <servlet-name>
          Ime servleta
        </servlet-name>

        <servlet-class>
          Polna pot do servleta
        </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>
       Ime servleta
        </servlet-name>

        <url-pattern>
       /url do nasega servleta
        </url-pattern>
    </servlet-mapping>
```

Slika 4.3: Vpis novega `servleta` v datotko `web.xml`

- `doGet`, ki sprejema parametra `HttpServletRequest` in `HttpServletResponse`.
- `doPost`, ki sprejema parametra `HttpServletRequest` in `HttpServletResponse`.
- `getServletInfo`, ki vrne kratek opis servleta.

Metodi `doGet` in `doPost` izvršita metodo `processRequest` v kateri programer napiše kdo, ki izvrši željeno akcijo. V primeru našega informacijskega sistema, te akcije pišejo na zaslon željene rezultate. Pisanje na zaslon je izvedeno s pomočjo razreda `PrintWriter`, ki je član paketa za standardni vhod in izhod.

Na vrhu razreda imamo anotacijo na katero se sklicuje `servlet v web.xml` datoteki. Zelo pomembno je ujemanje parametrov v anotaciji s parametri v opisniku servleta.

4.3.1 Pridobivanje vrednosti parametrov URL naslova v metodi processRequest

Za pridobitev vrednosti parametrov, ki sestavljajo URL naslov do željene akcije na strežniku uporabljamo objekt `HttpServletRequest`, ki ga ponuja HttpServlet. Dostop do paramentrov naslova url deluje po načinu ključ - vrednost (angl.*key-value*). To pomeni, da mora razvijalec poznati vse ključe, ki so posredovani s strani HTTP zahtevka. Od glavnega dela spletnega naslova so parametri ločeni z znakom ?. Če imamo Kako izgleda koda v praksi si lahko ogledamo na spodnjem primeru. V primeru4.4 želimo prebrati vrednosti s spletnega naslova `http://tkoliti.javaprovider.net/DiplomaWebApp/weather?lat=46&lng=15`.

```
protected void processRequest
(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
{
String lat =request.getParameter("lat");
String lng =request.getParameter("lng");
}
```

Slika 4.4: Pridobivanje vrednosti parametrov spletnega naslova

4.3.2 Pisanje na spletno stran s pomočjo PrintWriter razreda

V vseh servletih, ki jih uporabljam na strani strežniškega dela informacijskega sistema predstavlja glavni del kode pisanje očiščenih podatkov na naslov

dosegljiv prko `URL` naslova. `Java` nam zato ponuja razred `PrintWriter`. `PrintWriter` je član paketa `java.io`, ki razvijalcu omogočajo branje in pisanje s pomočjo standardnega vhoda in izhoda. Metoda `processRequest`, katera se izvede ob klicu `doGet` ali `doPost` metode servleta, piše na izhod s pomočjo PrintWriterja na način, ki je prikazan v sliki 4.5.

```
protected void processRequest(HttpServletRequest req,
HttpServletResponse res)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
try
{
pw.write("Pozdravljen svet");
}
catch (JSONException e)
{
pw.write("Prislo je do napake");
}
pw.close();
}
```

Slika 4.5: Pisanje z razredom `PrintWritter` na spletno stran

Kot je razvidno v kodi metode processRequest moramo najprej nastaviti tip vsebine, ki jo vrača odgovor na klic spletnega servisa. Objekta PrintWriter ni potrebno deklarirati, saj nam ga ponuja že objekt HttpServletResponse. Ob koncu pisanja moramo PrintWriter obvezno zapreti s klicem metode close.

4.4 Čiščenje nepotrebnih pridobljenih podatkov s spletnih servisov

Ker spletni servisi, ki jih uporabljam v informacijskem sistemu ponujajo več podatkov, kot jih potrebujem jih je potrebno prečistiti. V ta namen sem ustvaril razred z imenom `ParsersJson`, ki se izvede ob vsaki poizvedbi mobilnega klienta. V tem razredu so glavne štiri metode

- `getData(String url)`, ki s spletnega naslova prenese vso vsebino in jo vrne kot objekt tipa `String`
- `parseSimpleForcast(String input)`, ki očisti pridobljeno vremensko napoved in vrača objekt `JSONArray`
- `parseWikiLocationArticles(String input)`, ki vrne seznam objektov
- `parseReverseLocation(String input)`, ki vrne Objekt `GeoLocation` s vrednostmi trenutne lokacije nahajanja

Koda v sliki 4.6 prikazuje metodo `getData`, ki prebere podatke s spletne strani.

Kot lahko vidimo v metodi najprej pridobimo `URL` objekt našega spletnega naslova. Nato na ta objekt povežemo `URLConnection`, ki zagotovi vhodni tok podatkov, ki se zapisujejo v objekt razreda `BufferedReader`. V zadnjem delu metode moramo prepisati podatke iz `BufferedReader` objekta v String, ki ga lahko vrnemo aplikaciji za nadaljne delo. To storimo s preprosto `while` zanko, ki vsako prebrano vrstico iz `BufferedReaderja` vrne v `String`.

```
public String getData(String url)
throws IOException
{
URL urlToServer;
String data = "";
urlToServer = new URL(url);
URLConnection yc = urlToServer.openConnection();
BufferedReader in = new BufferedReader
(new InputStreamReader(yc.getInputStream()));
String inputLine;
while ((inputLine = in.readLine()) != null)
{
data = data + inputLine;
}
in.close();
return data;
}
```

Slika 4.6: Pridobivanje podatkov s spletne strani

Poglavje 5

Mobilna aplikacija

Mobilna aplikacija je del informacijskega sistema, ki samemu sistemu da največjo vrednost. Aplikacija omogoča uporabniku, da dostopa do podatkov, ne glede na lokacijo nahajanja. Med razvojem mobilnega klienta sem največ časa posvetil razvoju uporabniku prijaznih uporabniških akcij in implementaciji aktivnosti z navidezno resničnostjo. Posebeno sem bil pozoren tudi na pokrivanje nenavadnih primerov uporabe, kot je naprimer izguba podatkovne povezave na svetovni splet ali nezmožnost pisanja na zunanji pomnilnik imenovan `SD` kartica.

5.1 Načrtovanje aplikacije

Z izgradnjo mobilnega klienta sem začel po končanem razvoju strežniškega dela informacijskega sistema. Prvi korak vsakega dobro izvedenega projekta je predhodno načrtovanje. Za začetek sem razdelil izgradnjo mobilnega klienta na več delov. Ti deli predstavljajo potek razvoja klienta in so bili implementirani v sledečem vrstnem redu:

1. izgradnja uporabniškega vmesnika
2. razvoj dela sistema za prenos podatkov
3. razvoj aktivnosti z nastavitvami

4. razvoj aktivnosti za pregled interesnih točk

5. razvoj aktivnosti za pregled vremenske napovedi

6. razvoj aktivnosti za dostop do podatkov o interesnih točkah, ko uporabnik nima na voljo povezave v svetovni splet

7. razvoj aktivnosti s podporo navidezni resničnosti

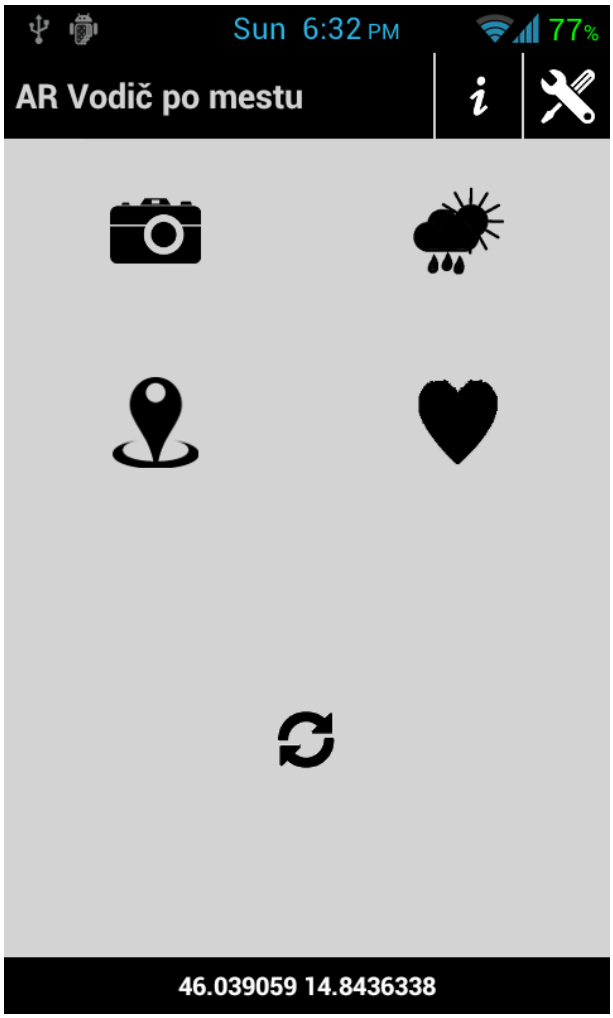
Ko sem določil potek razvoja aplikacije sem se lotil izbire akcij in primerov uporabe, ki jih bo vsaka aktivnost podpirala. Vse akcije bom opisal v sledečih podpoglavjih.

5.1.1 Izgradnja uporabniškega vmesnika

Pred začetkom izgradnje uporabniškega vmesnika apikacije sem prebral številna priporočila razvijalecem s strani `Googla`. Priporočila so dosegljiva na spletenem mestu `http://developer.android.com/guide/topics/ui/index.html`. Za igradnjo izgleda, ki ga uporabnik vidi ob zagonu aplikacije, sem izbral priporočljiv Googlov izgled, ki vsebuje aktivnosto vrstico (angl. *ActionBar*) na vrhu zaslona in velike gumbe za dostop do posameznih akcij aplikacije. V spodnjem delu sem dodal vrstico katere namen je obveščanje uporabnika o trenutni lokaciji. Ob spremembi lokacije se informacija samodejno posodobi. Slika 5.1 prikazuje zaslon ob zagonu aplikacije.

Uporabniški vmesnik sem napisal v `XML` jeziku, ker le ta omogoča največji nadzor nad postavitvijo elementov na zaslonu. Razvijalec `Android` aplikacije ima v razvojnem okolju `Eclipse` navoljo tudi razvoj vmesnika s pomočjo grafičnih elementov. Zaradi nepredvidljivega obnašanja `Eclipsa`, takšen razvoj odsvetujem. Prvi vmesnik je zgrajen s pomočjo elementov `LinearLayout`, ki omogočajo pravilen izris komponent na številnih zaslonskih ločljivostih telefona. Vrstica aktivnosti, ki se nahaja na zgornjem delu zaslona vsebuje tri elemente:

- `TextView` za prikaz imena aplikacije



Slika 5.1: Vstopna aktivnost aplikacije

- `ImageButton` za informacijo o aplikaciji
- `ImageButton` za dostop do nastavitev aplikacije

Pod njo imajo prostor za postavitev štiri glavne aktivnosti, ki so del aplikacije. Vsaka od teh ikon je `ImageButton` s posebnim obnašanjem med uporabnikovo interakcijo z njimi. Posebna iterackija je prav tako, kot izgled, opisana v `XML` jeziku. Izvorna koda interakcije je opisana v sliki 5.2.

Kot lahko razberemo iz slike 5.2 je starševski element tega dokumenta `selector`. Za vsako stanje gumba pa vsebuje tudi `item` element, ki določa

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

Slika 5.2: XML datoteka za upravljanje akcij nad gradnikom `ImageButton` in `Button`.

stanje. Razvijalec lahko za gradnika `ImageButton` in `Button` izbirata med sledečimi stanji:

- `focused`
- `pressed`

Obe zgoraj navedeni vrenosti sta lahko `true` ali `false`, kar nam omogoča da nadzorujemo štiri različna stanja gumbov.

Pod gumbi se nahaja animirana akcija za prenos podatkov s spletnega strežnika na telefon. Za to akcijo sem zgradil posebno komponento imenovano `ArcMenu`, ki ob dotiku v svojo okolico nariše akcije za osvežitev podatkov. `ArcMenu` komponenta je skupek programske kode `Java` in `XML` dokumenta. Kot gradnik uporabniškega vmesnika jo v našo datoteko za izgled dodamo na način ki ga prikazuje slika 5.3.

```
<si.matejpikovnik.diploma.ArcMenu
  android:id="@+id/buttonRefresh"
  style="@style/HomeImageButton" />
```

Slika 5.3: Prikaz dodajanja elementa `ArcMenu` v XML datoteko za izgradnjo uporabniškega vmesnika.

Upravljanje z obnašanjem `ArcMenu` gradnika je možno tako v `Java` kodi, kot v `XML` elementu, ki ga prikazuje slika 5.3.

V začetni maski aplikacije ima uporabnik navoljo tudi dostop do nastavitev. Do nastavitev lahko uporabnik dostopa na dva načina

- klik na gumb z ikono nastavitev
- premik prsta po zaslon od levega do desnega roba

Ob izvedbi ene izmed zgoraj navadenih akcij se uporabniku aplikacije prikaže aktivnost za nastavitve. Aktivnost za nastavitve ne predstavlja standardno aktivnost z nastavitvami v aplikacijah `Android`, ampak novo implementacijo le-te. Nova implementacija omogoča, da se čez predel zaslona s pomočjo animacije narišejo vsi podatki o aktivnostih. Izgled odprte aktivnosti lahko vidimo v sliki 5.4. Implementacija takšnega obnašanja aplikacije s strani grafičnega vmesnika je preprosta. Poskrbeti moramo, da je predel z nastavitvami ob zagonu aplikacije skrit in da se ob določeni akciji prikaže. Za zapiranje pogleda z nastavitvami ima uporabnik na voljo tri možnosti

- pritisk gumba za vrnitev, ki ga vsebuje vsak mobilni telefon z opracijskim sistemom `Android`
- premik prsta po zaslon od desnega do levega roba
- dvoklik po zaslonu

Slika 5.4: Prikaz aktivnosti z nastavitvami

V nastavitvah ima uporabnik na voljo spreminjati vrednosti treh parametrov, ki se uporabljajo za sestavljanje poizvedbe, ki se pošlje na strežnik. Te vrednosti so

- Območje prikaza točk interesa
- Število prikazanih točk interesa
- jezik podatkovne baze s katere pridobivamo podatke

Za spreminjanje vrednosti območja in števila prikazanih točk sem ust-

varil poseben dialog, ki vsebuje dva gradnika tipa `TextView` in popolnoma spremenjen gradnik `SeekBar`. Izgled dialoga prikazuje slika 5.5

Slika 5.5: Prikaz aktivnosti z nastavitvami

Željen izgled dosežemo s priredbo parametrov v gradniku `SeekBar`. Moj primer je viden v sliki **??**.

```
<SeekBar
    android:id="@+id/obmocjeSeekBar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/textViewTrenutnoObmocje"
    android:layout_marginTop="3dp"
    android:maxHeight="3dp"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:progressDrawable="@drawable/slider_progress"
    android:thumb="@drawable/circle" />
```

Slika 5.6: Prikaz dodajanja elementa `SeekBar` v XML datoteko za izgradnjo uporabniškega vmesnika.

Glavno nalogo v izgradnji ne standardnega gradnika `SeekBar` predstavljata elementa `android:progressDrawable` in `android:thumb`. Prvi je zadolžen za risanje izgleda indikatorja napredka, drugi pa za podobo, ki je dodana indikatorju napredka.

Ob prenosu podatkov z interesnimi točkami uporabnik odpre aktivnost `ZemljevnidAktivnost`. Ta aktivnost vsebuje aktivnosto vrstico (angl. *ActionBar*)in gradnik `MapView`, ki je del standardne knjižnice `com.google.android.maps`. Na `MapView` lahko natančno izrišemo interesne točke glede na njihovo pozicijo, ki smo jo prejeli kot odgovor s strani strežnika. V aktivnostni vrstici ima uporabnik možnost vrnitve na glavni zaslon in izbire pogleda zemljevida. Izgled aktivnosti `zemljevidAktivnost` prikazuje slika 5.7.

Slika 5.7: Prikaz aktivnosti z interesnimi točkami

Ob kliku na ikono *i* se uporabniku prikaže dialog imenovan `QuickAction`. `QuickAction` je dialog, katerega največjo prednost predstavlja pokritost za-

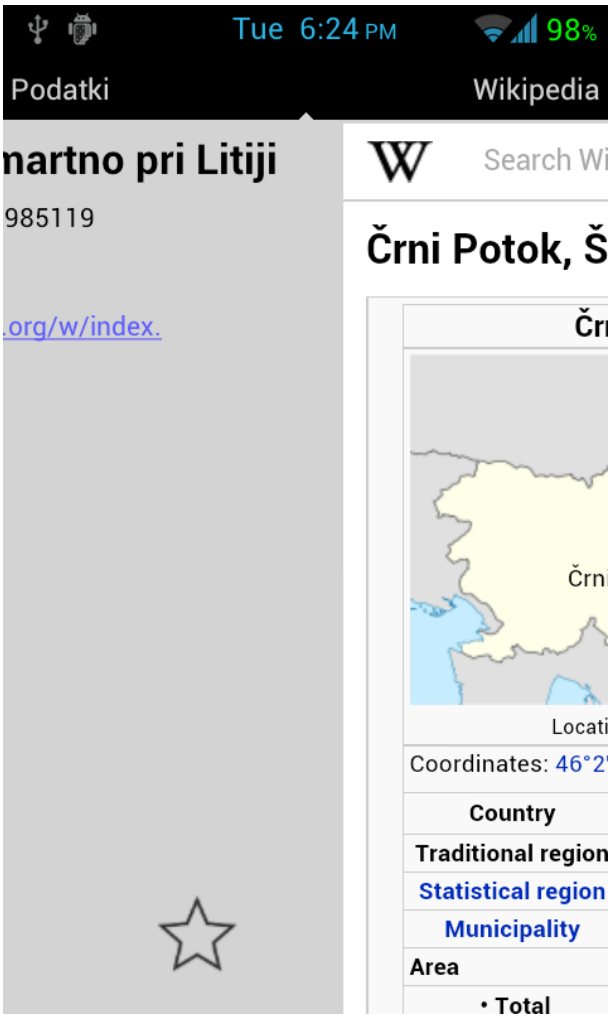
slona le z elementi katere vsebuje. `QuickAction` dialog vsebuje akcije za izbiro pogleda na `MapView` gradniku. Čeprav je `QuickAction` dialog del uporabniškega vmesnika, pa je sestavljen izključno iz programske kode `Java` in ni ustvarjen kot `XML` dokumenut.

Ob izbiri interesne točke se uporabniku prikaže `AlertDialog`, ki mu ponuja možnost zagona aktivnosti za natančnejši pregled podatkov o interesni točki. Aktivnost, ki prikaže podatke o interesni točki se imenuje `PodatkiOLokacijiActivity`. `PodatkiOLokacijiActivity` je del `FragmentActivity` razreda, ki jih je možno uporabljati v aplikacijah z verzijo `Android` 11 ali več. Ker moj mobilni klient podpira tudi verzije operacijskega sistema do `API` nivoja 8 sem moral dodati razširitveno knjižnico paketa `android.support.v4.app`. `FragmentActivity` uporabniku omogoča, da se sprehaja med aktivnostmi s pomočjo potega prsta po zaslonu. Tako mu je prihranjeno pritiskanje na gumbe, ki je časovno potratnejše. Slika 5.8 prikazuje menjavo prikaza podrobnosti s pomočjo `Fragment` dodatka.

Za implementacijo tega izgleda moramo na nivoju programiranja uporabniškega vmesnika v glavno datoteko z izgledom dodati dva gradnika. Prvi se imenuje `TitlePageIndicator`, ki nad vsako aktivnostjo izriše gradnik, čigar naloga je informiranje uporabnika o trenutnem nahajanju. Drugi gradnik, ki se nahaja pod `TitlePageIndicator` pa je `ViewPager`. Na gradnik `ViewPager` dodajamo svoje aktivnosti, ki so razširitev razreda `Fragment`. Slika 5.9 prikazuje dodajanje obeh gradnikov v datoteko z izgledom.

Naslednji izgled aktivnosti, ki jo bom opisal je uporabniški vmesnik za pregled vremenske napovedi. Tudi v tej aktivnosti se držim pristopov, ki sem jih uporabil že v vseh ostalih aktivnostih. Celotna aktivnost je namenjena spremljanju vremenske napovedi za našo trenutno lokacijo. Vremenski podatki so dosegljivi za prihajajoče štiri dni od trenutka pošiljanja poizvedbe. Uporabnik lahko vidi sledeče podatke za prihodnje dni

- najvišja temperatura
- najnižja temperatura



Slika 5.8: Prikaz podatkov o interesni točki med menjavo pogleda

- ikona s slikovno napovedjo

Na vrhu zaslona se nahaja `ActionBar` gradnik, pod njim so `TextView` gradniki, ki opisujejo vremensko napoved za izbran dan. V spodjnem delu zaslona pa sem vstavil še en samodejno izdelan gradnik imenovan `SegmentedRadioGroup`. Ta gradnik je podoben gradniku, ki ga najdemo na `iOS` napravah in se imenuje `UISegmentedControl`. Celoten izgled gradnika je definiran v XML datotekah in je izpopolnjen tako, da uporabniku nudi hitro in lahko navigacijo med posameznimi dnevi. Poleg menjave prikaza podatkov s kliki

```
<si.matejpikovnik.diploma.widgets.pageIndicator.
TitlePageIndicator
    android:id="@+id/indicatorExpense"
    android:layout_width="fill_parent"
  android:layout_height="wrap_content"/>

<android.support.v4.view.ViewPager
    android:id="@+id/pagerExpense"
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_weight="1" />
```

Slika 5.9: Prikaz dodajanja elementov `TitlePageIndicator` in `ViewPager` v XML datoteko za izgradnjo uporabniškega vmesnika.

na element `SegmentedRadioGroup` ima uporabnik možnost menjave prikaza tudi z gestami prstov po zaslonu. Za implementacijo gest sem uporabil razred `SimpleGestureListener`, ki razvijalcu ponuja lahko zajemanje akcij uporabnika na zaslonu.

5.1.2 Razvoj aktivnosti z nastavitvami

Razvoj aktivnosti in razredov z nastavitvami je bila ena prvih nalog pri razvijanju aplikacije. Del aplikacije, ki upravlja z nastavitvami bi lahko razdelil na tri dele

- Konstante nastavitev
- Grafični del nastavitev
- Razred za upravljanje z nastavitvami

Če želimo razumeti zakaj je celoten del aplikacje z nastavitvami zgrajen s treh delov, moramo vedeti kaj nam `Android API` na nivoju nastavitev

omogoča. Za delovanje nastavitev v naši aplikaciji ima razvijalec na voljo razred `SharedPreferences`. `SharedPreferences` predstavlja osnovno ogrodje za shranjevanje primitivnih vrednosti tipa ključ - vrednost (angl.*key-value*). V razredu `KonstanteZaNastavitve.java` imam predstavljena imena vseh spremenljivk, ki predstavljajo ključe v razredu `SharedPreferences`. Razred `NastavitvePomoc.java` je glavni razred, ki operira z vrednostmi zapisanimi v nastavitve. V njem najdemo t.i. `getter` in `setter` metode za dostop in nastavljanje vrednsti nastavitvam. V konstruktorju razreda `NastavitvePomoc.java` odpremo objekta `SharedPreferences` in `SharedPreferences.Editor`. Objekt `SharedPreferences.Editor` nam omogoča neposredno dostopati do spremenljivk nastavitev. Vsebino konstruktorja lahko vidmimo v sliki 5.10.

```
public NastavitvePomoc(Context context)
{
super();
prefs = PreferenceManager.
getDefaultSharedPreferences(context);
editor = prefs.edit();
}
```

Slika 5.10: Konstruktor razreda `NastavitvePomoc`

Kot sem že omenil, sem napisal razred `NastavitvePomoc` za lažje pridobivanje in nastavljanje vrednosti nastavitev v sami aplikaciji. Za prireditev vrednosti v `SharedPreferences` objektu potrebujemo metode, ki kot parameter sprejmejo konstantno vrednost, ki predstavlja ključ in vrednost, ki jo zapišemo na mesto konstante. Za pridobivanje vrednosti pa je poleg konstante, ki predstavlja ključ, podati tudi začetno vrednost, s katero zagotovimo, da aplikacija ob zagonu ne dostopa do objekta, ki ne obstaja. Ko zaključimo z vpisovanjem novih vrednosti v nastavitve aplikacje jih je potrebno še shraniti. To storimo s klicem funkcije `commit()` nad objektom `SharedPreferences.Editor`.

Za grafični del upravljanja z nastavitvami v aplikaciji sem izbral pristop, ki določa, da ima vsaka nastavitvena vrednost svoj dialog, preko katerega uporabnik določa vrednosti v aplikacijo. Vsi dialogi so dostopni iz glavne aktivnosti aplikacije, ki sem jo opisal v podpoglavju Izgradnja uporabniškega vmesnika.

Upravljanje z nastavitvami aplikacije je bilo v `API` nivoju 9 še poenostavljeno. `Google` je razvijalcem omogočil, da s klicem funkcije `apply ()` neodvisno od ostalih akcij zapišejo nastavitve v aplikacijo. S klicem metode `apply()` se podatki zapišejo v `RAM` in so uporabniku navoljo takoj.

5.1.3 Razvoj metod za prenos podatkov s strežnika

Prenos podatkov s stežnika je dolgotrajno opravilo. Za izvajanje časovno potratnih nalog nam `Android` razvijalski `APIji` ponujajo posebno imlementacijo zaganjanja novih niti imenovano `AsyncTask`. Ko uporabnik zažene novo nit razreda `AsyncTask` lahko še vedno uporablja aplikacijo, saj se nit izvede v ozadju in ne vpliva na izris elementov. Razred, ki razširi `AsyncTask` vsebuje metode, ki razvijalcu omogočajo lažje izvajanje zahtevnejših akcij in obveščanje uporabnika o stanju teh akcij. Te metode so:

- `onPreExecute()`
- `onProgressUpdate(Progress...)`
- `onPostExecute(Result)`
- `doInBackground(Params...)`

Izmed zgoraj naštetih metod se v svoji niti izvede le `onProgressUpdate(Progress...)`. Ostale metode se izvedejo v niti, ki izrisuje tudi gradnike na zaslonu. Razred, ki razširi razred `AsyncTask` uporablja tudi posebne parametre, ki razvijalcu omogočajo prenos podatkov iz metode, ki zažene akcijo za prenos do metode, ki izvede akcijo za prenos. Razvijalec mora nastaviti tri spremenljivke, kot kaže slika 5.11. Prvi parameter predstavlja tip parametrov, drugi tip spremljanja napredka prenosa in tretji tip razultata, ki ga vrne metoda `(Result)`.

```
public class PrenesiPOI extends
 AsyncTask<String, Integer, String>
```

Slika 5.11: Prikaz parametrov, ki jih prevzema razred, ki razširja `AsyncTask`

Aktivnost za prenos podatkov v ozadju moramo zagnati v niti, ki poganja grafični vmesnik. Slika 5.12 prikazuje metodo, ki prenese podatke za interesne točke.

```
private void prenesiPOI(String urlOfPOI)
{
PrenesiPOI lokacije = new PrenesiPOI(this);
String[] url = new String[1];
url[0] = urlOfPOI;
lokacije.execute(url);
}
```

Slika 5.12: Prikaz zagona nove niti s pomočjo `AsyncTask`

Za pridobivanje podatkov iz izvedene niti API razvijalcem ponuja klic metode `get()`, ki vrne objekt enakega tipa, kot ga pošljemo v nit. Vendar pa se ob klicu metode `get()` pojavi nepredvidena napaka, ki zaustavi delovanje celotne aplikcije (angl.*Force Close*). To napako sem zaobšel tako, da sem rezultate prirejal statični spremenljivki in do nje tudi tako dostopal. Rešitev ni najbolj elegantna, vendar nam prihrani številne ure raziskovanja.

V svojem mobilnem klientu sem ob začetku izvajanja prenosa ustvaril tudi dialog, ki prikazuje stanje prenosa podatkov. Dialog prične z izvajanjem v metodi `onPreExecute()`, konča se pa ob sprožitvi metode `onPostExecute(Result)`.

5.1.4 Razvoj aktivnosti za pregled interesnih točk

V tem podpoglavju bom opisal razovj aktivnosti za prikazovanje interesnih točk, ki se nahajajo v okolici uporabnika mobilne aplikacije. Aktivnost je

sestavljena s treh delov. Prvi zaslon se uprabniku prikaže ob zagnonu aktivnosti `ZemljevnidAktivnost`. Ta aktivnost vsebuje tudi zemljevid, ki ga razvijalec upravlja z objektoma `MapController` in `MapView`. Oba objekta za upravljanje sta na voljo v paketu `com.google.android.maps`, katerega dostop pridobimo z vstavitvijo `JAR` datoteke `maps.jar` v našo aplikacij. Če želimo v svoji `Android` aplikaciji prikazati zemljevid `Google Maps` moramo pridobiti edinstven ključ, ki bo zagotovil prikaz željenega zemljevida na telefonu. Za pridobivanje ključa moramo izvesti sledeče operacije:

- Pridobiti `MD5` ključ razhroščevalnega certifikata (angl.*KeyStore*), s katerim je podpisana aplikacija v času razvoja
- Povezati se na spletno stran `https://developers.google.com/android/maps-api-signup` in vpisati rezultat akcije pridobivanja `MD5` podpisa
- V element `com.google.android.maps.MapView` vstaviti pridobljen ključ kot vrednost elementa `android:apiKey`

Ob zaključku razvoja aplikacije in pripravo na objavo mora razvijalec podpisati svoj izdelek z razhroščevalnim certifikatom, ki ni enak razvojnemu certifikatu. Sprememba certifikata zahteva tudi menjavo ključa za dostop do zemljevidov `Google Maps`.

Po uspešnem prikazu zemljevida v aktivnosti sem se lotil pridobivanja trenutne lokacije telefona in prikaza zemljevida v centru te lokacije. Za pridobivanje lokacije GPS v aktivnosti moramo uporabiti razred `LocationManager` in ga povezati s sistemsko aktivnostjo (angl.*System Service*). V aktivnosti moramo klicati metodo `requestLocationUpdates`, ki za parametre sprejme

- Način pridobivanja lokacije
- Najmanjšo časovno frekvenco dostopa do novih podatkov
- Najmanjšo spremembo lokacijo za izvedbo nove poizvedbe
- Razred v kateremu se izvedejo akcije ob pridobitvi nove lokacije

Lokacijo uporabnika lahko pridobimo na dva načina. Prvi način je pridobivanje lokacije s pomočjo `GPS` senzorja, ki je vgrajen v telefon. To je način kako pridobimo točno lokacijo uporabnika. Odstopanje med resnično in prikazano lokacijo je nekaj metrov. V primeru, da uporabnik nima vklopljenega `GPS` senzorja lahko njegovo lokacijo pridobimo s pomočjo omrežnega signala ali `WiFI` omrežja. Slika 5.13 prikazuje zagon metode `requestLocationUpdates` glede na vrsto povezave, ki jo ima uporabnik omogočeno.

```
if (PreveriPovezljivost.
jeGPSVkljucen(getApplicationContext()))
{
locationManager.requestLocationUpdates
(LocationManager.GPS_PROVIDER, 0, 0, new
GeoUpdateHandler());
}
else
{
locationManager.requestLocationUpdates
(LocationManager.NETWORK_PROVIDER, 0, 0,
new GeoUpdateHandler());
}
```

Slika 5.13: Dostop do lokacije uporabnika preko `GPS` ali omrežnega senzorja.

Ko ima apikacija pridobljeno lokacijo uporabnika, s klicem metode `runOnFirstFix` postavi center zemljevida na to točko. V nadaljnem razvoju aktivnosti za pregledovanje zanimivih točk sem se lotil postavitve točk na zamljevid. `MapView` objekt, ki prikazuje zemljevid ima možnost dodajanja ikon v plast, ki se nahaja nad zemljevidom. V ta namen sem ustvaril razred `MyOverlays`, ki razširja `ItemizedOverlay` in razvijalcu omogoča dodajanja poljubne ikone na lokacijo zemljevida, ki je podana preko GPS koordinat. Vsak dodan element, ki je prikazan kot ikona, ima možnost zaznavanja uporabnikovh dotikov. To nam ponuja možnost prikazovanja dodatnih informacij o točki ali pa priredbe

novih akcij.

5.1.5 Aktivnost za pregled vremenske napovedi

Izgradnjo uporabniškega vmesnika za aktivnost v kateri lahko uporabnik pregleduje vremensko napoved za prihodnje štiri dni sem že opisal. V tem podpoglavju bom prikazal implementacijo posebnih gradnikov, ki so omogočili, da je aktivnost kar se le da uporabniku prijazna. Prvi poseben gradnik je imenovan `SmartImageView`. To je gradnik, ki razvijalcu omogoča, da prikaže sliko s spleta, le s podanim `URL` naslovom do te slike. Ta gradnik je zelo priročen za uporabo, saj sam poskrbi za implementacijo prenosa slike s spletnega naslova in izris na zaslon. Kot razvijalcu mi je implementacija tega gradnika prihranila veliko časa. Primer uporabe gradnika `SmartImageView` prikazuje slika 5.14.

```
String urlDoSlike = danVremeList.get(dan).getIconUrl();
imageViewIcon.setImageUrl(urlDoSlike);
```

Slika 5.14: Uporaba gradnika `SmartImageView`.

Prav tako pa mi je veliko časa prihranil gradnik `SimpleGestureFilter`. To je odprtokodni gradnik, čigar namen je poenostaviti rokovanje z akcijami oz. gestami, ki jih uporabnik izvaja na zaslonu pametnega telefona. Omenjeni gradnik razširja `SimpleOnGestureListener` in mu doda funkcije za lažje prepoznavanje uporabnikovih akcij na zaslonu. Brez večjih popravkov v kodi lahko prepoznamo sledeče akcije na zaslonu:

- premik prsta od spodaj proti vrhu
- premik prsta od zgoraj dol
- premik prsta iz leve proti desni
- premik prsta iz desne proti levi

Da naša metoda prepozna zgoraj navedene akcije, moramo v metodi `onFling` pravilno preračunati podatke o dotikih. Preračunavanje je izvedeno s pomočjo podatkov, ki jih zagotavlja razred `MotionEvent`. Ob izjvajanju gest na zaslonu, se uporabniku prikazujejo podatki o določenem dnevu. Ko uporabnik doseže zadnji dan, za katerega še imamo na voljo podatke, se ob naslednji akciji prikažejo podatki za prvi dan. Logika je implementirana tudi v obrantni smeri, kar pomeni, da je izvajanje aplikacije lahko, uporabniška izkušnja pa bolj prijetna, kot če bi aktivnost imela končno točko.

5.1.6 Dostop do interesnih točk brez podatkovne povezave

V svoji mobilni aplikaciji sem uporabniku ponudil tudi možnost dostopa do interesnih točk v trenutku, ko pametni mobilni telefon nima vzpostavljene pozave s svetovnim spletom. Uporabniku je omogočen prenos spletne strani `Wikipedia` o članku na pomnilniško kartico mobilnega telefona. Da lahko uporabnik brez podatkovne povezave prebira informacije o interesnih točkah, mora v aplikaciji izvesti sledeče korake:

- Zagnati aktivnost za pregled interesnih točk
- Si izbrati željeno interesno točko in zagnati aktivnost za pregledovanje podrobnosti o tej točki
- Izbral ikono zvezde, ki zažene akcijo za prenos podatkov s spletne strani na pomnilniško kartico telefona

Akcijo, ki zažene prenos podatkov o interesni točki na mobilni telefon lahko vidimo v sliki 5.9. Za uspešno izvedbo te akcije morajo biti izpolnjeni naslednji pogoji:

- Pomnilniška kartica mora biti v telefonu
- Povezava v svet mora biti aktivna
- Ustvarjena mora biti mapa z imenom `arpomestu`

Lažje upravljanje z aktivnostmi pomnilniške kartice sem napisal razred `FileSistemManipulator`, ki vsebuje metode za poenostavljeno delo s pomnilniško kartico.

Za preverjanje pristnosti pomnilniške kartice v telefonu skrbi metoda `isSdPresent()`. `createFolderForApp()` ustvari mapo, v katero se shranjujejo datoteke izbranih interesnih točk. Obe zgoraj navedeni metodi uporabljate klice metod iz paketa knjižnic imenovnem `android.os.Environment`.

Ob uporabnikovem kliku za prenos podatkov o interesni točki na mobilni telefon se zažene izvajati proces, ki je sestavljen iz dveh delov

- prenos podatkov na pomnilniško kartico
- vpis podatkov interesne točke v `SQLite` podatkovno bazo

Prenos podatkov s spleta in zapis na pomnilniško kartico telefona je akcija, ki se izvede v svoji niti preko `AsyncTask` razreda. Pred prenosom spletne strani na pomnilniško kartico v programski kodi preverim, da

- je pomnilniška kartica prisotna v telefonu
- ima uporabnik pravico pisanja na pomnilniško kartico
- je ustvarjena mapa, v katero se shranjujejo spletne strani o interesnih točkah.

V metodi `doInBackground`, ki se izvaja v svoji niti, je zaporedje ukazov za prenos in pisanje na pomnilniško kartico sledeče.

- vzpostavitev `HttpURLConnection` razreda s `URL` naslovom spletne strani
- ustvarjanje nove datoteke z `ID` podatkov o interesni točki v imenu
- pisanje podatkov spletne strani v `InputStream` objekt
- pisanje podatkov s `InputStream` objekta v datoteko na pomnilniški kartici

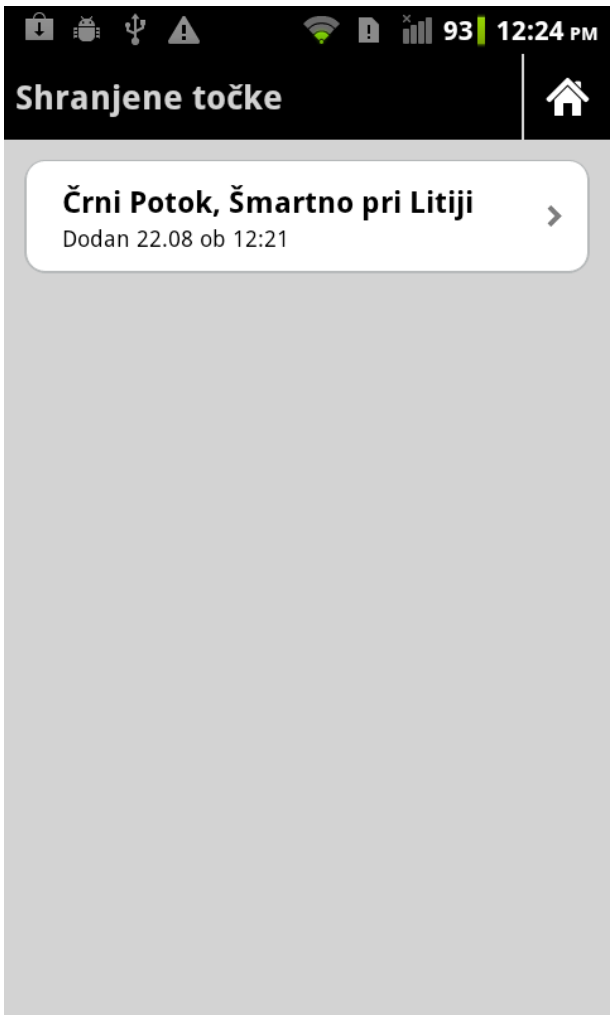
Ko so podatki zapisani v pomnilniški kartici, jih je potrebno zapisati še v podatkovno bazo `SQLite`. Za dostop in delo s podatki v `SQLite` podatkovni bazi, sem napisal dva razreda, ki mi izjemno olajšata upravljanje z bazo v aplikaciji. Prvi razred se imenuje `UstvariBazo`, ki razširja razred `SQLiteOpenHelper`. V tem razredu se najajajo metode, ki ustvarijo in posodobijo podatkovno bazo.

Drugi razred pa razvijalcu omogoča, da v bazo zapisuje, briše in dostopa do podatkov. Razred se imenuje `DBPomoc`. Med uporabo tega razreda v akcijah programa se moramo držati določenih pravil, ki jih zahteva objekt `SQLiteDatabase`. Pred akcijo nad podatkovno bazo moramo bazo odpreti, izvesti željeno akcijo in nato zapreti bazo. Če ob končani akciji ne zapremo baze, tvegamo napako v programu.

Ko smo izvedli zgoraj navedene akcije lahko do podatkov o interesni točki dostopamo tudi brez podatkovne povezave. Shranjene podatke aplikacije lahko pregledeujemo v aktivnosti `ShranjeniElementi`. Podpoglavje `Izgradnja uporabniškega vmesnika` opisuje izgled te aktivnosti, v tem podpoglavju pa bom opisal njeno obnašanje.

Del aplikacije za pregled shranjenih interesnih točk je sestavljen iz dveh delov. Prvi del predstavlja aktivnost, ki vsebuje gradnik `UITableView`. Ta gradnik uporabniku predstavi vse shranjene podatke o interesnih točkah. Slika 6.1 prikazuje aktvinost za pregled shranjenih interesnih točk.

Ob kliku na željeno interesno točko program zažene aktvivnost za pregled podatkov o izbrani interesni točki. Ta aktivnost predstavlja drugi del. Aktivnost je sestavljena iz gradnika `WebView`, ki prikaže podatke shranjene na pomnilniški kartici.



Slika 5.15: Prikaz aktivnosti s shranjenimi interesnimi točkami

Poglavje 6

Navidezna resničnost

Navidezna resničnost predstavlja pogled na svet z dodanimi neresničnimi objekti. Uporabnik svet spremlja skozi zaslon mobilnega telefona in kamero, ki jo ima telefon vgrajeno. Navidezni objekti se na zaslonu telefona lahko izrisujejo glede na številne parametre, ki jih senzorji telefona prejemajo z okolice. Navidezna resničnost prevzema veliko vlogo v številnih industrijah in panogah po vsem svetu. Aplikacije z navidezno resničnostjo postajo pomemben in vsak dan večji igralec v številnih vsakdanjih opravilih, med katerimi lahko omenim:

- Izvajanje preprostih opravil
  - Uporabnik lahko s pomočjo navidezne resničnosti lažje izvaja opracije, ne da bi bral navodila
- Navigacija
  - Naprava med vožnjo zaznava oznake iz okolice in uporabniku izrisuje pot in ga opozarja na nevarnosti
- Umetnost
  - Na zaslon lahko kupcu izrišemo končni izgled umetnine v 3D pogledu

- Arhitektura
  - Kot pri umetnsoti, lahko tudi tu na zaslon izrišemo izgled celotnih mest in življenja v njih
- Igre
  - Strelske igre bodo prenešene v resnični svet
- Turizem
  - Preko mobilnega telefona lahko uporabnik spremlja interesne točke v svoji okolici

Največkrat mobilne aplikacje za izris navideznih objektov uporabljajo podatke pridobljene s senzorjev za lokacijo (`GPS`), naklon naprave in pozicijo v prostoru. Druga možnost za izris željenega objekta pa je izris ob določenem vzorcu, ki ga kamera prepozna. Vzorec ponavadi predstavlja `QR kodo`, ki v sebi nosi še dodatne podatke. Navidezna resničnost bo v letih, ki prihajajo zagotovo predstavljala novo dimenzijo razvoja programske opreme.

Velik del razvoja mobilnega klienta predstavlja tudi aktivnost z navidezno resničnostjo. Aktvinost sem implementiral s pomočjo knjižnice, ki jo zagotavlja nemško podjetje `Metaio`.

6.1 Metaio

`Metaio` je nemško podjetje, ki razvijalcem ponuja knjižnice za delo z navidezno resničnostjo. `Metaio` ponuja knjižnice za razvoj na platformah

- `Mobile SDK`
  - `Android`
  - `iOS`
- `PC SDK`

- `Web SDK`

Delo s knjižnicami je enostavno. Vendar pa morajo biti za pravilno delovanje aplikacije zadoščeni številni pogoji, ki obsegajo tako programsko kot strojno opremo mobilnega telefona. Prav zaradi teh zahtev pa sem pri razvoju svoje aktivnosti z navidezno resničnostjo naletel na številne težave. Te težave sem razrešil šele po testiranju aplikacije na različnih napravah. Za test in razvoj aplikacije sem uporabil napravo `HTC Desire HD` na kateri teče `Android` verzije 4.0.3 znane pod komercialnim imenom `Ice cream sandwich`. Naprava zaradi nepodpore akcijam knjižnice za navidezno resničnost ni omogočala uspešno delovanje aktivnosti. Težavo sem rešil s telefonom `HTC Desire Z`, ki je brez težav zagnal aktivnost z navidezno resničnostjo in mi pomagal pri dokončanju diplomske naloge.

Razvijalci pri `Metaiu` so pripravili demo projekt, ki mi je pomagal razumeti vse načine delovanja in upravljanja z objekti v navidezni resničnosti. S pomočjo priložene dokumentacije in demo projekta sem ustvaril aktivnost, ki s pomočjo pridobljenih podatkov o lokaciji, izriše znak za najbližjo interesno točko. `Metaio Mobile SDK`, kot se imenuje knjižnica, razvijalcem ponuja številne možnosti uporabe. Med njimi so najpogostejše

- aktivnost z izrisom naprednih vsebin
- aktivnost z izrisom interesnih točk
- aktivnost z izrisom interesnih točk ali naprednih vsebin ob prepoznavi objektov v naravi.

6.2 Aktivnost navidezne resničnosti

Rezultat končnega dela programske kode, ki predstavlja navidezno resničnost, lahko vidmo v sliki 6.1.

Razvoj te aktivnosti je bil sestavljen z implementacijo abstraktnega razreda `ARViewActivity`, ki vsebuje metode razreda `Activity` in pisanjem kode

Slika 6.1: Prikaz uporabe navidezne resničnosti na terenu

za razred `GPSLocationBasedActivity`. Razred `ARViewActivity` vsebuje metode in logiko za zagon aktivnosti z navidezno resničnostjo. V njem skrbim za obnašanje slik in izris na zaslonu mobilnega telefona. Med metode, ki se izvedejo v tem razredu spada tudi metoda `IUnifeyeMobileAndroid.loadNativeLibs`, ki je časovno izjemno zahtevna, saj v pomnilnik telefona naloži vse dodatke, ki so potrebni pri izrisu navideznih objektov na naš zaslon. Prav tako ta razred skrbi za obnašanje uporabnikovih gest na zaslonu telefona. Geste aktivnost prepozna s pomočjo implementacije metod, ki jih zagotavlja `OnTouchListener`. Aktivnost prepozna vse geste, ki so definirane s konstantno številko 1 imenovano `MotionEvent.ACTIONUP`. Akcija zazna dvig uporabnikovega prsta na zaslonu, razred `MotionEvent` je sposoben prepoznati še številne druge uporabnikove akcije, ki pa jih nisem implementiral.

Razred `GPSLocationBasedActivity` razširja obnašanje razreda `ARViewActivity` njegova glavna naloga pa je zagotoviti pravilno delovanje aplikacije ob zazani spremembi lokacije uporabnika. Za osveževanje podatkov o lokaciji uporabnika skrbi metoda `onLocationSensorChanged`, ki kot parameter sprejme objket `LLACoordinate`. `LLACoordinate` je objekt, ki vsebuje točen opis točke na kateri se uporabnik nahaja. Podatki so razvijalcu na voljo preko vrednosti za zemljepisno dolžino in širino ter višino in natančnostjo.

Za pravilno delovanje navidezne resničnosti je bilo potrebno v aplikacijo dodati datoteke, ki aktivnost opisujejo in so del knjižnice, ki jo zagotavlja podjetje `Metaio`. Datoteke sem prenesel v mapo `Assets`, v katero razvijalec shrani grafične in ostale elemente do katerih aplikacija dostopa med delovanjem. Datoteka katero aktivnost največ uporablja se imenuje `TrackingDataGPSCompass.xml` in vsebuje podatke o uporabi senzorjev telefona. V tej datoteki je predstavljeno obnašanje navideznih elementov, ki se bodo izrisali na zaslon uporabnika. Poleg podatkov o senzorju telefona, lahko v datotekah `TrackingData` definiramo tudi ostale metode zajemanja podatkov iz okolice, kot so

- zajemanje podatkov s pomočjo črtne kode
- zajemanje podatkov s pomočjo slik.

Poglavje 7

Zaključek

Rezultat razvoja informacijskega sistema, ki sem ga opisal v diplomski nalogi, je delujoč sistem za zagotavljanje podatkov o interesnih točkah in vremenski napovedi uporabniku na pametnih mobilnih telefonih z operacijskim sistemom `Android`.

Glavna skrb ob načrtovanju in razvoju sistema je bila možnost uporabe mobilne aplikacije neodvisno od položaja uporabnika v svetu. Med aktivnosti razvoja, katerim sem posvetil veliko časa, spada tudi razoj uporabniškega vmesnika in skrb za prijetno uporabniško izkušnjo. Menim, da sem obe zahtevi uspešno izvedel in zgradil aplikacijo, ki je uporabniku prijazna in deluje tekoče.

Celoten sistem je lahko obladljiv, saj je potrebno ob spremembi odgovora spletnih servisov, spremeniti samo klic in nastavljanje vrednosti v objekt, ki se prense na mobilni telefon. S tem lahko omogočimo pravilno delovanje mobilnega klienta in nam ni potrebno objativi nove verzije na spletno trgovino z aplikacijami. Prav zaradi skupne točke sprejema in obdelave podatkov lahko tudi z malo truda v sistem dodamo tudi dodatne vire informacij, ki bi bili uporabniku v pomoč pri potovanjih.

Največ težav pri diplomski nalogi mi je povzročala implementacija aktivnosti z navidezno resničnostjo. Z ostalmi deli informacijskega sistema nisem imel težav, saj sem predhodno poznal tehnologije in ogrodja s kater-

imi sem izdelal mobilnega klienta in aplikacijo, ki teče na strežniku.

Literatura

[1] (2012) Android (operating system); dostopno na:
http://en.wikipedia.org/wiki/Android_(operating_system).

[2] (2012) Vaadin (frame work); dostopno na:
https://vaadin.com/home.

[3] (2012) Wikilocation; dostopno na:
http://wikilocation.org/.

[4] Marko Grönroos "Book of Vaadin," 2012.

[5] (2012) Wunderground; dostopno na:
http://www.wunderground.com/weather/api/d/stats.html.

[6] (2012) Developer Android; dostopno na:
http://developer.android.com/index.html.