Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Aleksander Robnik

ANDROID APLIKACIJA Z UPORABO RAZPOZNAVANJA SPLETNE VSEBINE

Diplomsko delo

Maribor, september 2013

Diplomsko delo visokošolskega strokovnega študijskega programa

ANDROID APLIKACIJA Z UPORABO RAZPOZNAVANJA SPLETNE VSEBINE

Diplomsko delo

Študent:	Aleksander Robnik
Študijski program:	Visokošolski študijski program
	Računalništvo in informacijske tehnologije
Mentor:	doc. dr. Tomaž Kosar, univ.dipl.inž. rač. in inf.

Maribor, september 2013

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Številka: E1041525
Datum in kraj: 08. 04. 2013, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 46/2012)
izdajam

SKLEP O DIPLOMSKEM DELU

1. **Aleksandru Robniku,** študentu visokošolskega strokovnega študijskega programa RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE, se dovoljuje izdelati diplomsko delo pri predmetu Prevajalniki.

2. **MENTOR:** **doc. dr. Tomaž Kosar**

3. **Naslov diplomskega dela:**
ANDROID APLIKACIJA Z UPORABO RAZPOZNAVANJA SPLETNE VSEBINE

4. **Naslov diplomskega dela v angleškem jeziku:**
WEB CONTENT PRESENTATION WITH ANDROID APPLICATION

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 30. 09. 2013 v referatu za študentske zadeve.

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

Dekan:
red. prof. dr. Borut Žalik

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

ZAHVALA

Zahvaljujem se mentorju doc. dr. Tomažu Kosarju za pomoč in vodenje pri opravljanju diplomskega dela.

Posebna zahvala velja staršem, ki so me ves čas podpirali in mi omogočili študij.

ANDROID APLIKACIJA Z UPORABO RAZPOZNAVANJA SPLETNE VSEBINE

Ključne besede: Android, razpoznavalnik, spletne vsebine, Java

UDK: 004.3:004.777(043.2)

Povzetek

Velik napredek na področju razvoja mobilnih naprav je omogočil prenos aplikacij s klasičnih računalnikov na mobilne naprave. V diplomski nalogi smo prenesli spletno aplikacijo v mobilno aplikacijo, ki prikazuje podatke iz spletnega vira Maribor-Pohorje.

Predstavili smo operacijski sistem Android, njegovo uporabo in razvojno okolje, v katerem smo zgradili aplikacijo. V praktičnem delu diplomskega dela smo podrobneje predstavili razpoznavalnik HtmlCleaner. Demonstrirali smo delovanje aplikacije, ki prikazuje podatke iz spletnega vira na mobilnih napravah Android.

V sklepnem delu smo prikazali primerjavo med razpoznavalniki spletnih vsebin in generatorjem aplikacij iz spletnih strani.

WEB CONTENT PRESENTATION WITH ANDROID APPLICATION

Key words: Android, web parser, web content, Java

UDK: 004.3:004.777(043.2)

Abstract

Significant progress in the development of mobile devices has enabled the transfer of applications from traditional computers to mobile devices. In this diploma work we transferred a web application into mobile application that shows data from an online source Maribor-Pohorje.

We presented the Android operating system, its use and development environment in which we built an application. In the practical part of the diploma work we presented web parser HtmlCleaner. We demonstrated the operation of application that display data from an online source on Android mobile devices.

In the concluding section we showed a comparison between the web parsers and generator that creates applications from web source.

KAZALO VSEBINE

Kazalo slik

Kazalo primerov kode

Uporabljeni simboli in kratice

OHA - konzorcij podjetij za razvoj standardov za mobilne naprave

SDK - programski paket za razvijanje aplikacij

HTML - je označevalni jezik za izdelavo spletnih strani

VM - virtualni stroj

JVM - Java virtualni stroj

XML - označevalni programski jezik

ID - identifikator

IPC - medprocesna komunikacija

ADT - orodja za razvijanje Android aplikacij

AVD - Android virtualna naprava

JDK - Java razvojni paket

API - aplikacijski programski vmesnik

GNU - splošna javna licenca

DOM - dokumentni objektni model

XNI - Xerces nativni vmesnik

EPL - Eclipse javna licenca

LGPL - GNU Lesser splošna javna licenca

ASP - Microsoftova tehnologija, ki omogoča aktivno kreiranje spletnih strani na strežniku

JSP - tehnologija Sun Microsystems, ki omogoča razvijanje dinamičnih spletnih strani

PSP - tehnologija za razvoj dinamičnih spletnih strani z uporabo jezika Python

PHP - programski jezik za razvoj dinamičnih spletnih vsebin

SAX - preprosti API za XML

URL - enolični krajevnik vira

1 UVOD

Selitev oglaševanja medijev na svetovni splet je pred desetletjem pomenila velik napredek za prepoznavnost podjetij in njihovih izdelkov oziroma storitev na svetovnem trgu. Ker se informacijska tehnologija nenehno razvija in napreduje, smo priča velikemu napredku na področju mobilnih naprav. Napredek se kaže pri selitvi aplikacij iz klasičnih in prenosnih računalnikov na mobilne naprave. Mobilne naprave so kljub svoji majhnosti lahko enako zmogljive ali po specifikacijah celo presegajo klasične in prenosne računalnike. Mobilne naprave poganjajo zelo zmogljivi operacijski sistemi, kot so Microsoftov Windows Phone, Applov iOS in Googlov Android. Privlačnost uporabe mobilnih naprav je, da imamo velik nabor aplikacij, ki jih lahko dobimo v t.i. marketih. Aplikacije za mobilne naprave so uporabniško prijazne in po uporabnosti enako zmogljive kot aplikacije za klasične računalnike.

Cilj diplomske naloge je predstavitev operacijskega sistema Android, njegovo uporabo in razvojno okolje, v katerem smo implementirali aplikacijo. V ta namen smo prenesli spletno aplikacijo v mobilno aplikacijo, ki prikazuje podatke iz spletnega vira. Za razpoznavanje spletnega vira smo uporabili knjižnico HtmlCleaner. Pogledali smo tudi primerjavo med razpoznavalniki spletnih vsebin in generatorjem aplikacij iz spletnih strani.

Maribor s Pohorjem je drugo največje slovensko mesto in prestolnica Štajerske. Je univerzitetno ter gospodarsko, finančno, upravno, kulturno in turistično središče severovzhodne Slovenije. Maribor je bil Evropska prestolnica kulture leta 2012 in je Evropska prestolnica mladih 2013 [1]. S tem namenom smo se tudi mi odločili, da prenesemo spletno aplikacijo v mobilno aplikacijo, ki prikazuje podatke iz spletnega vira http://maribor-pohorje.si. Tako smo vsem uporabnikom omogočili hiter in preprost dostop do vseh potrebnih informacij, ki jih ponuja mobilna aplikacija. Naša mobilna aplikacija ponuja vse splošne informacije o Mariboru in Pohorju, kulturnih prireditvah, kulinariki, športu in zabavi ter o ostalih splošnih informacijah.

2 ANDROID

Android je operacijski sistem in programska platforma za mobilne naprave, ki temelji na Linux jedru. Razvija ga Google v sodelovanju s podjetji združenja Open Handset Alliance. Android je primarno namenjen pametnim mobilnim telefonom z zaslonom na dotik, vendar ga najdemo tudi na ostalih napravah kot recimo prenosni računalniki, tablični računalniki, pametne ure, avtomobilski računalniki in ostale naprave.

Android je namenjen različnim uporabnikom ne glede na spol, starost in predznanje, ki ga imajo. Je preprost za uporabo, prilagodljiv ter hkrati kompleksen in zmogljiv operacijski sistem in programski paket. Prednosti Androida so, da je odprto-koden in popolnoma brezplačen. Razvijalcem omogoča velik nabor funkcij in možnosti pri razvoju programske opreme. Ponuja zelo dober uporabniški vmesnik, ki je hiter in odziven. Naprave z operacijskim sistemom Android se samodejno sinhronizirajo, kar pomeni, da v primeru okvare naprave ne izgubimo podatkov. Hkrati pa so vsi naši podatki ažurni ne glede na to ali do njih dostopamo preko mobilne naprave ali osebnega računalnika.

Zelo pomemben del Androida so aplikacije, ki jih lahko uporabnik na preprost način nalaga na mobilno napravo. To omogoča storitev Google Play, preko katere je mogoče neposredno na mobilni napravi brskati aplikacije in jih nalagati na mobilno napravo. Aplikacije, ki jih dobimo na marketu, so lahko brezplačne ali plačljive, vendar je cena plačljivih aplikacij razmeroma nizka [2].

2.1 Zgodovina Androida

Zgodovina operacijskega sistema Android sega v leto 2003, ko je bilo v Kaliforniji ustanovljeno podjetje Android Inc., ki se je ukvarjalo z razvojem programske opreme za mobilne telefone. Google je podjetje prevzel leta 2005, do izdaje testne (angl. beta) verzije Androida pa je prišlo novembra 2007. Prva komercialna verzija Androida je izdana septembra

2008 skupaj s prvimi pametnimi telefoni. Prva verzija Androida je bila zelo okrnjena, saj je vsebovala funkcije kot so budilka, brskalnik za internet, android market, kamero in ostale.

Naslednja verzija operacijskega sistema Android verzije 1.5 je bila izdana aprila 2009 in je prva verzija, ki je dobila kodno ime in sicer Cupcake. Od takrat naprej nosi vsaka večja izdaja nove verzije kodno ime, izbrano glede na slaščico, katere ime se začne z naslednjo črko v abecedi.

Do danes imamo 8 večjih izdaj Android operacijskega sistema, imena verzij so naslednja:

- Android 1.5 – Cupcake
- Android 1.6 – Donut
- Android 2.0 in 2.1 – Eclair
- Android 2.2 – Froyo
- Android 2.3 – Gingerbread
- Android 3.0 – Honeycomb
- Android 4.0 – Ice Cream Sandwich
- Android 4.1 in 4.2 – Jellybean

Po podatkih iz leta 2013 je po celem svetu v uporabi več kot 500 milijonov aktivnih naprav z operacijskim sistemom Android, kar kaže na njegovo vsestransko uporabo [3].

2.2 Splošno o Androidu

Android je operacijski sistem za pametne telefone, tablične računalnike, prenosne računalnike, avtomobilske računalnike, ki ga razvija Google v sodelovanju s podjetji združenja Open Handset Alliance. Je odprto-kodni operacijski sistem, ki temelji na Linux jedru in Java programskem vmesniku. Bistvo odprto-kodnega operacijskega sistema je, da ga lahko proizvajalci naprav, ki uporabljajo Android, prirejajo za svoje potrebe.

Na razvoj Androida je vsekakor vplivala Googlova vizija o množični uporabi interneta in potreba po razvoju mobilnih naprav v prihodnosti. Bistvena prednost glede na konkurenco je, da ima zelo dobro podporo za razvijalce, ki lahko samostojno razvijajo programsko opremo. Google ponuja SDK (angl. software development kit), ki omogoča ustvarjanje novih aplikacij za operacijski sistem Android. Aplikacije, ki jih razvijamo s pomočjo razvojnega okolja Eclipse, so običajno napisane v programskem jeziku Java [4].

2.3 Zgradba operacijskega sistema Android

Operacijski sistem Android je sestavljen iz petih komponent. Te komponente so Linux jedro, knjižnice, Android runtime, aplikacije in njihovo ogrodje.

- Linux jedro

Celotni operacijski sistem Android je zgrajen iz Linux jedra, z nekaterimi arhitekturnimi spremembami, ki jih je implementiral Google. Linux jedro je tisto, ki komunicira s strojno opremo in vsebuje vse bistvene strojne gonilnike. Linux jedro deluje tudi kot abstraktna plast med strojno opremo in programskimi plastmi. Android uporablja Linux jedro za vse ključne funkcije kot so upravljanje pomnilnika, upravljanje s procesi, povezovanje, varnostne nastavitve in ostale funkcije. Ker je Android operacijski sistem, zgrajen na zelo priljubljenem Linux jedru, omogoča zelo preprosto prilagajanje operacijskega sistema različnim tipom strojne opreme.

- Knjižnice

Naslednja plast operacijskega sistema Android so knjižnice. Knjižnice omogočajo napravam, da obravnavajo različne vrste podatkov. Te knjižnice so napisane v programskem jeziku c ali c++ in so specifične za določeno vrsto strojne opreme.

Nekatere od teh knjižnic vsebujejo naslednje:

- medijsko ogrodje: ponuja različne tipe kodekov, ki omogočajo snemanje in predvajanje različnih vrst multimedijskih formatov
- SQLite: je podatkovna baza namenjena shranjevanju podatkov
- WebKit: omogoča, da lahko z brskalniki prikazujemo HTML vsebine
- OpenGL: se uporablja za prikaz 2D ali 3D grafike na zaslon.

- Android izvajalnik kode (runtime)

Android izvajalnik kode (angl. runtime) sestavljata Dalvikov VM (angl. virtual machine) in Java knjižnice.

- Dalvikov VM

Dalvikov VM je vrsta JVM (angl. Java virtual machine), ki se uporablja v napravah z Android operacijskim sistemom za delovanje aplikacij in je optimiziran za procesiranje z nizko procesorsko močjo in v okoljih, kjer imamo na voljo malo pomnilnika. Za razliko od JVM, Dalvikov VM ne poganja razrednih datotek, ampak poganja datoteke s končnico .dex. .dex datoteke so zgrajene iz razrednih datotek v času prevajanja in zagotavljajo večjo učinkovitost v okolju z nizkimi procesorskimi in pomnilniškimi viri. Dalvikov VM omogoča kreiranje več primerkov virtualnih strojev, ki se lahko kreirajo istočasno, kar zagotavlja varnost, izolacijo, dobro upravljanje pomnilnika in večnitno podporo.

- Java knjižnice

Java knjižnice se razlikujejo od Java SE in Java ME knjižnic. Vendar pa te knjižnice zagotavljajo večino funkcionalnosti, ki so definirane v Java SE knjižnicah.

- Aplikacije

Aplikacije so vrhnja plast v arhitekturi Android, kjer imamo nameščene aplikacije. Nekaj standardnih aplikacij je vnaprej nameščenih na novih mobilnih napravah. Te aplikacije so SMS odjemalec, pozivnik, spletni brskalnik, imenik in ostale aplikacije. Kot razvijalci lahko napišemo aplikacijo, ki zamenja katerokoli že obstoječo sistemsko aplikacijo. To pomeni, da nismo omejeni z dostopom do funkcij. Dejansko smo neomejeni in lahko počnemo vse kar želimo z Android operacijskim sistemom in aplikacijami. Tako Android odpira nešteto možnosti za razvijalce.

Slika 1: Arhitektura Androida [6]

- Ogrodje aplikacij

To so bloki, s katerimi naše aplikacije neposredno komunicirajo. Ti programi opravljajo osnovne funkcije mobilnih naprav, kot so upravljanje z viri, klici, SMS sporočila in ostale funkcije. Za razvijalce pomenijo ti bloki osnovna orodja, s katerimi gradimo naše aplikacije.

Pomembni bloki v ogrodjih aplikacij so:

- dejavnostni upravljalec (angl. Activity manager): upravlja z življenjskim krogom aplikacij
- ponudniki vsebin (angl. Content Providers): upravlja z izmenjavo podatkov med aplikacijami
- telefonski upravljalec (angl. Telephony Manager): upravlja z vsemi klici in uporabljamo ga, če želimo priti do klicev v naši aplikaciji
- lokacijski upravljalec (angl. Location Manager): upravlja z lokacijami z uporabo GPS sistema
- upravljalec z viri (angl. Resource Manager): upravlja z različnimi tipi virov, ki jih uporabljamo v aplikacijah [5].

2.4 Razvoj aplikacij za Android

Za razvoj aplikacij za Android se predvsem uporablja programski jezik Java in razvojno okolje Eclipse. Za prevajanje kode uporabljamo Android SDK orodja, skupaj s podatki in datotekami, ki vsebujejo vire. Vsaka aplikacija se združi v en paket, datoteko s končnico .apk. Vsa koda v eni .apk datoteki je aplikacija, ki jo lahko namestimo na Android mobilne naprave.

Ko imamo aplikacijo nameščeno na mobilni napravi, ta živi v svojem varnem okolju.

- Android operacijski sistem je večuporabniški sistem Linux, kjer je vsaka aplikacija drugačen uporabnik.

- Privzeto sistem dodeli vsaki aplikaciji unikaten Linux uporabniški ID. Sistem nastavi dovoljenja za vse datoteke v aplikaciji. Tako lahko samo tisti uporabnik, ki mu je bil dodeljen ID aplikacije, dostopa do teh datotek.
- Vsak proces ima svoj VM (angl. virtual machine), tako koda aplikacije teče ločeno od ostalih aplikacij.
- Privzeto ima vsaka aplikacija svoj Linux proces. Android začne proces takrat, ko se izvrši katera od komponent aplikacije. Proces se zaključi takrat, ko ga več ne potrebujemo ali ko sistemu zmanjka pomnilnika za ostale aplikacije.

Tako Android sistem implementira princip najmanjšega privilegija. To pomeni, da ima vsaka aplikacija privzeto dostop samo do komponent, ki jih potrebuje za svoje delovanje. Na ta način ustvarimo varno okolje, v katerem lahko aplikacija dostopa samo do komponent sistema za katere ima dovoljenje.

Obstajajo tudi načini, da aplikacija deli podatke z drugimi aplikacijami kakor tudi, da aplikacija dostopa do sistemskih storitev.

- Možno je urediti, da si dve aplikaciji delita isti Linux uporabniški ID. V tem primeru lahko aplikaciji med seboj dostopata do vseh datotek. Če želimo ohraniti sistemske vire, lahko aplikaciji z istim uporabniškim ID-jem tečeta v istem Linux procesu in si delita enak VM.
- Aplikacija lahko zahteva pravice za dostop do podatkov, kot so kontaktni podatki, SMS sporočila, interni in eksterni pomnilnik, kamera, WI-FI in ostali podatki. Vse te pravice morajo biti sprejete s strani uporabnika v času namestitve aplikacije [7].

2.5 Komponente aplikacij

Komponente aplikacij so ključni bloki pri gradnji Android aplikacij. Vsaka komponenta je drugačna točka preko katere lahko sistem pride v našo aplikacijo. To pa ne pomeni, da so vse

komponente vstopne točke za uporabnika in nekatere so odvisne med seboj. Vsaka komponenta obstaja kot samostojna entiteta in igra specifično vlogo. Vsaka je unikaten gradnik, ki nam pomaga definirati končno vedenje naše aplikacije.

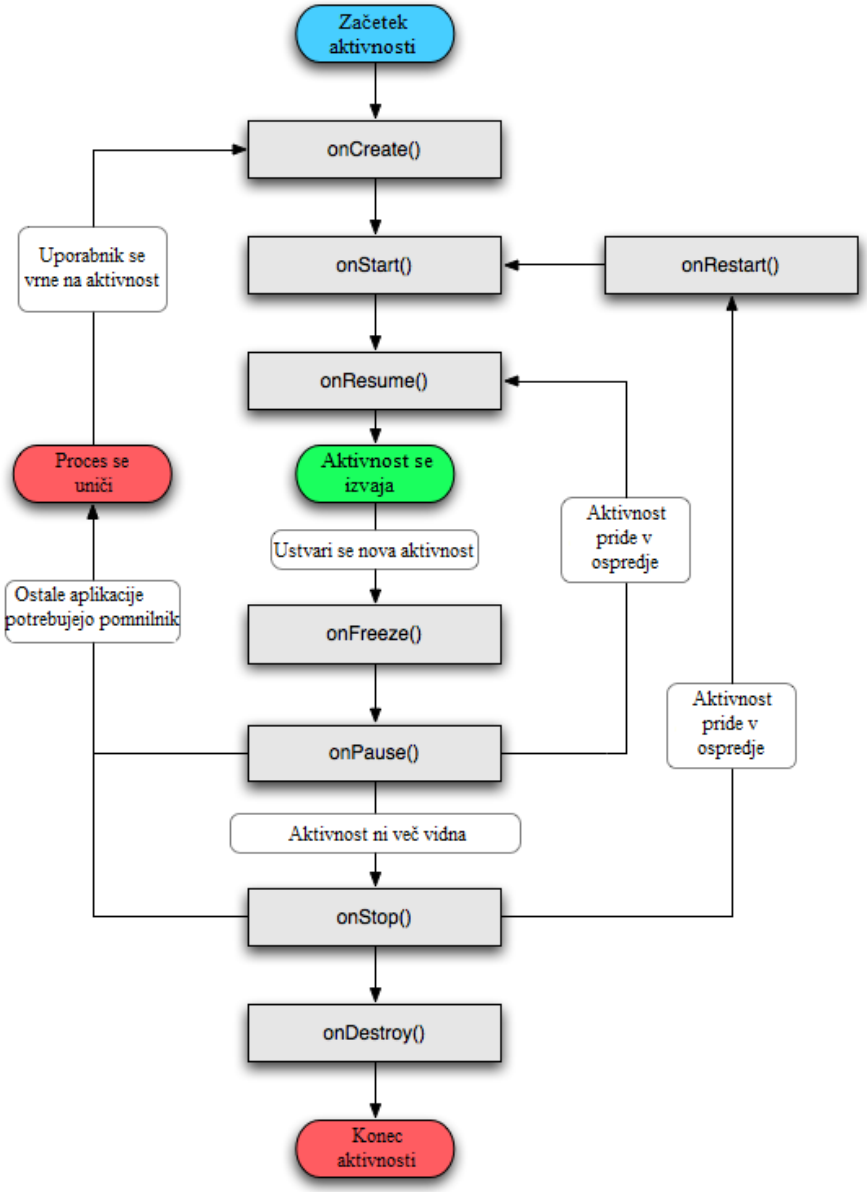
Obstajajo štiri vrste komponent aplikacije. Vsaka komponenta opravlja drugačno nalogo in ima ločen življenjski krog, ki definira kako se komponenta ustvari in uniči [7].

2.5.1 Aktivnosti aplikacije

Aktivnost aplikacije predstavlja en zaslon z grafičnim vmesnikom, s katerim lahko uporabniki komunicirajo z namenom, da izvedejo določeno akcijo. Tako lahko uporabniki brskajo po spletu, pregledujejo elektronsko pošto, SMS sporočila, itd. Vsaka aktivnost ima okno v katerem imamo uporabniški vmesnik. Okno običajno zapolni celoten zaslon, lahko je tudi manjše in lebdi nad ostalimi okni.

Aplikacijo običajno sestavlja več aktivnosti, ki so povezane med seboj. Vsaka aplikacija ima začetno aktivnost, ki se uporabniku prikaže, ko zažene aplikacijo. Vsaka aktivnost lahko začne drugo aktivnost z namenom, da izvede določene akcije. Ko začnemo novo aktivnost, se prejšnja aktivnost ustavi in sistem ohrani stanje prejšnje aktivnosti. Ko zaženemo novo aktivnost, se doda na sklad in se nam prikaže na zaslonu. Sklad uporablja način »zadnji noter, prvi ven«, torej, ko uporabnik zaključi z izvajanjem trenutne aktivnosti, se ta odstrani iz sklada in nadaljuje se izvajanje prejšnje aktivnosti.

V primeru, da se aktivnost zaustavi, ker se je sprožila nova aktivnost, se stanje sporoči preko metod povratnih klicev (angl. callback) v življenjskem krogu aktivnosti. Obstaja več metod povratnih klicev od katerih lahko aktivnost sprejme sporočilo ob spremembi svojega stanja. Aktivnosti lahko ustvarimo, nadaljujemo, zaustavimo in uničimo, pri tem pa lahko v vsakem stanju aktivnosti opravimo določeno delo, ki je primerno za izbrano stanje aktivnosti. Na primer, ko nadaljujemo z aktivnostjo lahko obnovimo vse akcije, ki smo jih prekinili z zaustavitvijo aktivnosti. Vsi ti prehodi med aktivnostmi so del življenjskega kroga aktivnosti, kar je razvidno iz slike 2.



Slika 2: Življenjski krog aktivnosti v Androidu [8]

- Ustvarjanje aktivnosti

Aktivnost ustvarimo tako, da ustvarimo podrazred iz razreda »Activity«. V podrazredu moramo implementirati metode povratnih klicev, ki jih sistem kliče ob preklapljanju med

različnimi stanji aktivnosti. Ta stanja so lahko kreiranje, ustavljanje, nadaljevanje in uničenje aktivnosti.

Dve najbolj pomembni metodi povratnih klicev sta:

- onCreate()

  To metodo moramo obvezno implementirati in se pokliče, ko ustvarimo aktivnost. V metodo moramo implementirati vse potrebne komponente, ki jih aktivnost potrebuje. Najbolj pomembna komponenta je setContentView(), ki definira videz našega uporabniškega vmesnika aktivnosti (primer kode 1).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

Primer kode 1: Primer onCreate() metode

- onPause()

  Sistem pokliče to metodo, ko uporabnik zapusti aktivnost (to pa ne pomeni, da se bo aktivnost uničila). V tej metodi shranimo vse spremenljivke in ostale podatke, ki jih želimo ohraniti za nadaljnjo izvajanje aplikacije.

Obstaja še več metod, ki jih moramo uporabljati, da zagotovimo pravilno delovanje aplikacije, kadar preklapljamo med različnimi aktivnostmi. Upravljati moramo tudi z nepričakovanimi napakami, ki se lahko pojavijo med menjavanjem aktivnosti. S tem zagotovimo, da se aktivnosti ne prekinejo ali uničijo.

- Implementacija uporabniškega vmesnika

Uporabniški vmesnik za aktivnost je določen po hierarhiji pogledov (angl. View), izpeljanih iz razreda »View«. Vsak pogled upravlja določen del pravokotnega prostora, ki se nahaja v

oknu aktivnosti in se lahko odzove na zahteve uporabnika. Na primer pogled je lahko gumb, ki izvede določeno operacijo, ko uporabnik pritisne nanj.

Android ponuja številne vnaprej pripravljene poglede, ki jih lahko razvijalci aplikacij uporabijo pri gradnji uporabniškega vmesnika. Gradniki (angl. Widgets) so pogledi, ki ponujajo vizualne in interaktivne elemente, ki jih lahko uporabimo pri gradnji uporabniškega vmesnika. Ti elementi so na primer gumb, tekstovno polje, potrditveno polje, slika in ostali elementi. »Layout« so pogledi izpeljani iz razreda »ViewGroup« in ponujajo unikatne modele za svoje izpeljane poglede kot so »LinearLayout«, »GridLayout« ali »RelativeLayout«. Izpeljemo lahko tudi podrazred iz razredov »View« in »ViewGroup« in s tem ustvarimo svoje lastne gradnike in poglede, ki jih lahko vgradimo v našo aktivnost (primer kode 2).

Najbolj osnoven način za definiranje postavitev (angl. Layout) je z uporabo pogledov, ki jih napišemo v označevalnem jeziku XML. Na ta način imamo uporabniški vmesnik ločen od ostale kode, ki definira obnašanje naše aplikacije. Postavitev lahko nastavimo kot uporabniški vmesnik za našo aktivnost z metodo »setContentView()« tako, da kot argument podamo ID postavitve, ki jo želimo prikazati.

```
</RelativeLayout><RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

</RelativeLayout>
```

Primer kode 2: Primer gradnika RelativeLayout in Button

- Deklaracija aktivnosti v manifestu

V manifestu je treba definirati aktivnost z namenom, da bo dostopna sistemu. Aktivnost definiramo tako, da značko »<activity>« vključimo znotraj značke »<aplication>«, kar tudi lahko vidimo v spodnjem primeru kode 3.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" >

    <application>
        <activity android:name="com.android.androidactivity.MainActivity" >
        </activity>
    </application>

</manifest>
```

Primer kode 3: Primer deklaracije aktivnosti v manifestu

Obstaja še nekaj drugih atributov, ki jih lahko vključimo v deklaracijo aktivnosti. Ti atributi so lahko labela, ikona ali vrsta teme za izbrano aktivnost. Edini obvezni atribut je »android:name« s katerim določimo ime razreda za aktivnost.

- Uporaba namenskih filtrov

Za posamezno aktivnost lahko v manifestu definiramo namenske filtre z uporabo značke »<intent-filter>«. Z namenskimi filtri določimo, kako bodo ostale komponente aplikacije aktivirale posamezno aktivnost.

Ko ustvarimo novo aplikacijo z uporabo Android SDK orodij, se nam samodejno ustvari aktivnost, ki vsebuje privzeto dva filtra. Prvi filter definira aktivnost kot glavno »main«, kar pomeni, da bo to vstopna aktivnost v aplikacijo. Drugi filter »launcher« definira, da lahko to aktivnost zaženemo. Primer kode lahko vidimo v spodnjem primeru.

```
<activity
    android:name="com.android.maribor.ActivityMenu"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Primer kode 4: Primer filtrov v manifestu

- Zagon aktivnosti

Aktivnost lahko zaženemo tako, da pokličemo metodo »startActivity()« ali definiramo namen (angl. intent), s katerim opišemo aktivnost, ki jo želimo zagnati. Z namenom lahko opišemo točno določeno aktivnost. Lahko pa opišemo vrsto akcije, ki jo želimo opraviti in sistem za nas izbere ustrezno aktivnost. Namen tudi lahko sprejme majhno količino podatkov, katere lahko prenašamo med aktivnostmi.

Ko gradimo aplikacije, pogosto vnaprej vemo katero aktivnost bomo naslednjo zagnali. To lahko storimo tako, da v namen podamo ime razreda za tisto aktivnost, ki jo želimo zagnati. V spodnjem primeru kode lahko vidimo, kako zaženemo aktivnost.

```
Intent intent = new Intent(this, ActivityListViewOpis.class);
this.startActivity(intent);
```

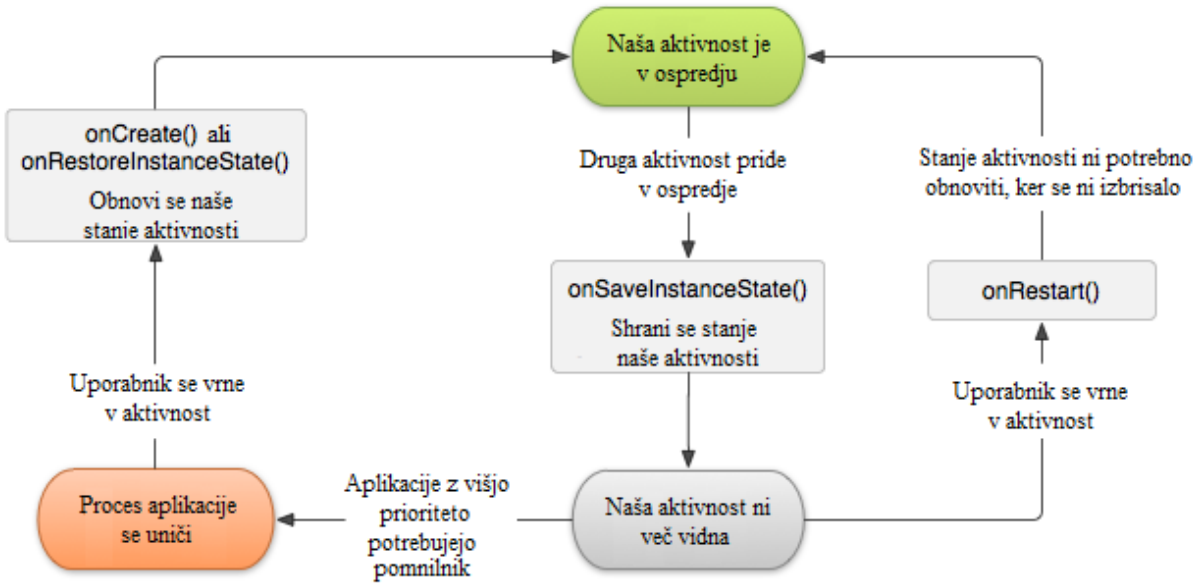
Primer kode 5: Zagon aktivnosti

- Shranjevanje stanja aktivnosti

Kadar aktivnost pavziramo ali ustavimo, se njeno stanje ohrani, ker se objekt aktivnosti še vedno nahaja v pomnilniku in tako ostanejo vse informacije nedotaknjene. To pomeni, da lahko uporabnik, ko se vrne nazaj v to aktivnost, nadaljuje z delom tam, kjer je končal.

V primeru, da je sistemu zmanjkalo pomnilnika, se uniči aktivnost oziroma uniči se objekt aktivnosti, ki je shranjen v pomnilniku. V tem primeru ne moremo več nadaljevati te aktivnosti, sistem pa mora ustvariti novi objekt za to aktivnost, da jo lahko ponovno uporabimo. Uporabnik ni obveščen o tem, da je sistem uničil aktivnost in pričakuje, da bo nadaljeval z delom tam, kjer je ostal. Zato moramo zagotoviti še dodatno metodo povratnih klicev, ki bo obvarovala podatke v primeru, da sistemu zmanjka pomnilnika. Ta metoda se imenuje »onSaveInstanceState()«.

Sistem pokliče metodo »onSaveInstanceState()« preden aktivnost postane izpostavljena uničenju. Sistem pošlje tej metodi snop (angl. Bundle) v katerega lahko shranimo stanje aktivnosti. Stanje lahko shranimo s spremenljivkami z uporabo metod kot so »putString()« in »putInt()«. V primeru, da sistem uniči naš proces aplikacije, bomo lahko povrnili stanje aktivnosti. Potek delovanja metod je prikazan na spodnji sliki [8].



Slika 3: Prehodi med stanji aktivnosti [8]

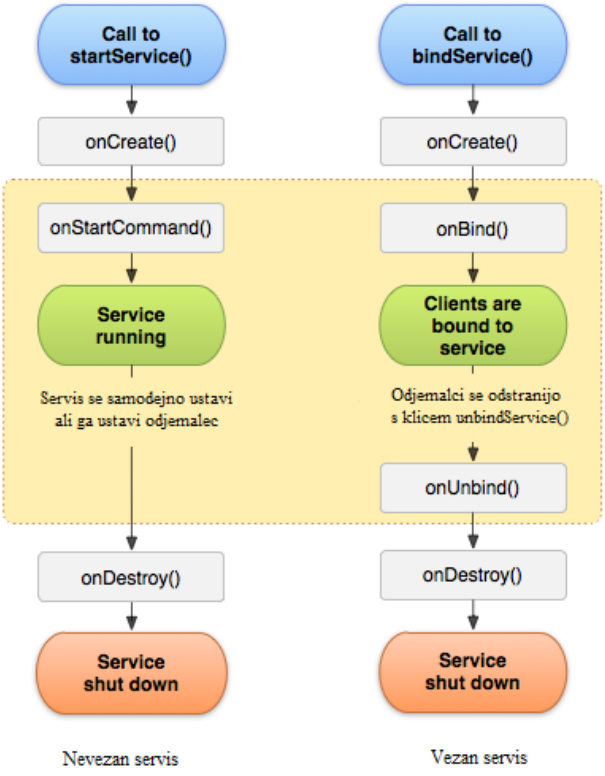
2.5.2 Servisi

Servis (angl. service) je komponenta aplikacije, ki lahko opravlja dolgo trajajoče operacije v ozadju aplikacije in ne vsebuje uporabniškega vmesnika. Komponente aplikacije lahko zaženejo servis, ki se bo izvajal v ozadju tudi, če bomo zapustili našo aplikacijo. Poleg tega se komponenta lahko poveže na servis z namenom, da bi komunicirala z njim ali izvajala komunikacijo z IPC (angl. interprocess communication). Na primer servis lahko hkrati upravlja z omrežjem, predvaja glasbo in izvaja branje/zapisovanje v datoteke iz ozadja.

Servis se lahko izvaja v dveh oblikah:

- Servis se izvaja (angl. started)

Servis se začne izvajati, ko komponenta aplikacije, npr. aktivnost, pokliče metodo »startService()« za zagon servisa. Ko smo zagnali servis, se začne izvajati v ozadju aplikacije za nedoločen čas tudi, če se komponenta, ki je zagnala servis, uniči. Običajno servis opravlja eno samo operacijo in ne vrača rezultata uporabniku. Na primer servis lahko prenese datoteko s spleta in se nato samodejno ustavi.



Slika 4: Življenjski krog servisa [9]

- Vezan servis (angl bound)

Servis je vezan, ko se komponenta aplikacije veže nanj ob klicu metode »bindService()«. Vezan servis ponuja vmesnik med odjemalcem in strežnikom, ki omogoča komponentam, da lahko komunicirajo s servisom, pošiljajo zahteve, pridobivajo rezultate in izvajajo procese z IPC. Vezan servis se izvaja tako dolgo dokler je nanj vezana komponenta aplikacije. Na servis je lahko hkrati povezanih več komponent in ko vse zapustijo servis, se ta uniči [9].

2.5.3 Ponudniki vsebin

Ponudniki vsebin (angl. Content providers) upravljajo dostop do strukturiranega niza podatkov. Enkapsulirajo podatke in zagotovijo mehanizme za zagotavljanje zaščite podatkov. Ponudniki vsebin so standardni vmesnik, ki povezuje podatke v enem procesu s kodo, ki teče v drugem procesu.

Kadar želimo dostopati do podatkov v ponudniku vsebin, lahko uporabimo objekt razreševalnika vsebin (angl. ContentResolver) v kontekstu (angl. Context) naše aplikacije za komunikacijo s ponudnikom kot odjemalec. Objekt razreševalnika vsebin komunicira z objektom ponudnika kot instanca razreda, ki ga implementira ponudnik vsebin. Objekt ponudnika vsebin sprejme podatkovne zahteve od odjemalca, nato izvede zahtevano operacijo in vrne rezultat [10].

2.5.4 Sprejemniki

Sprejemniki (angl. Broadcast receivers) je komponenta, ki se odziva na oddana sporočila celotnega sistema. Mnoga sporočila izvirajo iz sistema. Na primer sporočilo lahko naznanja, da se je zaslon izklopil, da imamo prazno baterijo ali da smo zajeli sliko. Aplikacija lahko oddaja sporočila drugim aplikacijam, na primer sporoči, da so bili podatki preneseni s spleta in so na voljo ostalim aplikacijam za uporabo. Sprejemniki ne vsebujejo uporabniškega vmesnika, lahko pa vsebujejo statusno vrstico, ki opozori uporabnika, da se izvaja nek dogodek.

Bolj splošno so sprejemniki prehod do ostalih komponent, njihova naloga pa je da opravijo minimalno količino dela. Sprejemnik lahko implementiramo, kot podrazred razreda »BroadcastReceiver« in vsako sporočilo je poslano kot namenski objekt [7].

2.6 Procesi in niti

Ko se komponenta aplikacije zažene in aplikacija nima drugih aktivnih komponent, Android sistem ustvari nov Linux proces za aplikacijo z uporabo ene niti. Privzeto se vse komponente ene aplikacije izvajajo v enem procesu in v eni niti, imenovani glavna nit. Če se komponenta zažene in se že izvaja proces za to aplikacijo, se ta komponenta začne izvajati znotraj tega procesa in uporablja enako nit. Možno je tudi ustvariti svoj proces za posamezne komponente, kakor tudi več niti za vsak proces [11].

2.7 Razred AsyncTask

AsyncTask nam omogoča izvajanje asinhronih opravil v našem uporabniškem vmesniku. Izvaja operacije, ki bi ovirale delovanje delovne niti in nato posreduje rezultate niti uporabniškega vmesnika. To pomeni, da nam ni potrebno direktno upravljanje z nitmi.

Če hočemo uporabiti razred AsyncTask, moramo iz njega izpeljati podrazred in v njega implementirati metodo povratnih klicev »doInBackground()«, ki se izvaja v ozadju. Če želimo podatke prikazati v našem uporabniškem vmesniku, moramo implementirati metodo »onPostExecute()«, ki dostavi rezultate iz metode »doInBackground()« in se izvaja v niti uporabniškega vmesnika. Na ta način lahko varno posodobimo uporabniški vmesnik.

AsyncTask zaženemo z metodo »execute()«, ki jo zaženemo iz niti uporabniškega vmesnika, kar lahko vidimo na primeru kode 6.

```
public void onClick(View v) {
    new DownloadImageTask()
            .execute("http://www.feri.uni-mb.si/images/feri.png");
}
```

Primer kode 6: Primer zagona AsyncTaska

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

Primer kode 7: Primer uporabe AsyncTask

Z uporabo AsyncTask zagotovimo varno izvajanje niti uporabniškega vmesnika in implementacija je preprostejša (primer kode 7). S tem ločimo del dela, ki bi se moral izvajati v delovni niti in del, ki bi se moral izvajati v niti uporabniškega vmesnika [11].

3 ORODJA ZA RAZVOJ IN TEHNOLOGIJE

Za razvoj Android aplikacij potrebujemo skupek knjižnic in orodij, imenovanih Android SDK, ki nam olajšajo razvoj aplikacij. Za razvoj aplikacij smo uporabljali razvojno okolje Eclipse in programski jezik Java.

3.1 Eclipse

Eclipse je prosto dostopno razvojno okolje, ki podpira več programskih jezikov. Obsega osnovni delovni prostor in omogoča razširitev preko vtičnikov. Eclipse omogoča razvoj aplikacij v programskem jeziku Java in preko vtičnikov še ostale programske jezike, na primer Ada, C, C++, Cobol, Fortran, PHP, Ruby in še mnoge druge. Razvojno okolje vsebuje Eclipse JDT za razvijanje v programskem jeziku Java, Eclipse CDT za C/C++ in Eclipse PDT za PHP.

Tudi mi smo uporabili Eclipse Juno za razvoj naše aplikacije, v katerega smo namestili vtičnike iz paketa orodij Android SDK. Prvi vtičnik, ki smo ga namestili, je ADT (angl. Android development toolkit), ki je vmesnik za nameščanje ostalih vtičnikov. Nato smo namestili vtičnik Android SDK in AVD Manager. Za ustrezno delovanje razvojnega okolja smo namestili še JDK (angl. Java development kit), ki nam omogoča razvijanje Android aplikacij s programskim jezikom Java [12].

3.2 Android SDK

Android SDK nam ponuja knjižnice za vmesnik uporabniškega programa (angl. API) in razvojna orodja, ki so potrebna, da lahko zgradimo, testiramo in odpravimo napake iz Android aplikacij. Uradno podprto razvojno okolje je Eclipse z uporabo ADT vtičnikov.

Izboljšave za Android SDK so povezane z razvojem operacijskega sistema Android. SDK podpira tudi starejše verzije Androida v primeru, da razvijalci želijo izdelati aplikacije za starejše verzije mobilnih naprav [13].

3.3 Android ADT

ADT (angl. Android Developer Tools) je vtičnik za razvojno okolje Eclipse, ki nam ponuja integrirano okolje v katerem lahko razvijamo aplikacije. ADT razširi zmogljivosti Eclipsa in nam omogoča hitro vzpostavitev novega projekta, ustvarjanje uporabniškega vmesnika, razhroščevanje aplikacij z uporabo Android SDK orodij. Omogoča nam tudi izvoz datotek .apk, ki služijo za distribucijo naše aplikacije [13].

3.4 Android AVD

Android virtualna naprava (angl. AVD) je konfiguracija emulatorja, ki nam omogoča simuliranje dejanske naprave z definiranjem različnih strojnih in programskih izbir.

AVD sestavljajo:

- Strojni profil: določa funkcije strojne opreme za virtualno napravo. Na primer lahko nastavimo tip procesorja, velikost primarnega in sekundarnega pomnilnika, uporabo kamere in še ostale izbire.
- Programski profil: z njim določimo, katero verzijo Android operacijskega sistema bomo imeli nameščeno na naši virtualni napravi.
- Ostale izbire: lahko izbiramo med različnimi tipi naprav, kar nam omogoča nadzor nad dimenzijo zaslona in videzom. Prav tako lahko uporabimo pomnilniško kartico za shranjevanje podatkov na virtualni napravi.
- Namenski prostor za shranjevanje podatkov: v tem prostoru so shranjeni uporabnikovi podatki (aplikacije, nastavitve...) in podatki iz pomnilniške kartice [14].

3.5 DDMS

Android nudi orodja za razhroščevanje, imenovana Dalvik Debug Monitor Server (DDMS), ki omogočajo:

- preusmeritev vrat (angl. port-forwarding)
- zajemanje slike na napravi
- informacije o nitih in kopici
- logcat (sistem za beleženje napak)
- informacije o procesih
- informacije o prejetih klicih in SMS sporočilih
- informacije o lokacijskih podatkih.

DDMS je integriran v razvojno okolje Eclipse, kakor tudi v Android SDK orodjih. DDMS lahko uporabljamo z virtualno napravo (angl. emulator) in s fizično napravo [15].

3.6 Java

Java je splošno namenski, objektno orientiran programski jezik, ki je zasnovan tako, da vsebuje čim manj implementacijskih odvisnosti. Razvil ga je James Gosling s sodelavci v podjetju Sun Microsystems. Posebnost programskega jezika je, da programske kode, ki jo poganjamo na eni platformi, ni treba ponovno prevajati, da deluje na drugih platformah. Java aplikacije so običajno prevedene v bitno kodo, ki jo lahko poganjamo na vseh JVM, neodvisno od arhitekture računalnika. Java je od leta 2012 eden od najbolj priljubljenih programskih jezikov za spletne aplikacije s več kot 10 milijonov uporabnikov. Sintaksa jezika je podobna sintaksi jezika C in C++.

Originalna in referenčna implementacija Java prevajalnika, virtualnega stroja in razrednih knjižnic je bila razvita s strani podjetja Sun leta 1991 in izdana leta 1995. Od maja 2007 je Sun v skladu s specifikacijami Java Community Process prelicenziral večino svojih tehnologij pod licenco GNU (angl. General Public License) [16].

4 PRIMERJAVA RAZPOZNAVALNIKOV

4.1 HtmlCleaner

HtmlCleaner je odprto-kodni razpoznavalnik HTML dokumentov, napisan v programskem jeziku Java. Vsebine HTML dokumentov, ki jih najdemo na spletu, so običajno slabo strukturirane in neprimerne za nadaljnjo procesiranje. Zato moramo tem dokumentom najprej urediti HTML značke, atribute in besedilo. HtmlCleaner preuredi posamezne elemente HTML dokumenta in tako dobimo dobro urejen XML dokument. HtmlCleaner uporablja podobna pravila, kot jih uporablja večina spletnih brskalnikov za ustvarjanje dokumentnega objektnega modela (DOM).

V naslednjem primeru bomo predstavili slabo strukturiran HTML, ki vsebuje nezaključene značke. V primeru kode 8 nam manjkata znački in .

```
<ul class="feriInfo">
        <li>Smetanova ulica 17, 2000 Maribor</li>
        <li>tel.: 02/220 7000</li>
        <li>faks: 02/220 7272</li>
        <li>e-pošta: <a target="_blank" href="mailto:feri@um.si">feri@um.si</li>
```

Primer kode 8: Primer slabo strukturirane HTML vsebine

Ko obdelamo HTML vsebino z razpoznavalnikom HtmlCleaner, dobimo dobro urejen XML dokument (primer kode 9).

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head />
  <body>
    <ul class="feriInfo">
      <li>Smetanova ulica 17, 2000 Maribor</li>
      <li>tel.: 02/220 7000</li>
      <li>faks: 02/220 7272</li>
      <li>e-pošta:
        <a target="_blank" href="mailto:feri@um.si">feri@um.si</a>
      </li>
    </ul>
  </body>
</html>
```

Primer kode 9: Primer dobro strukturiranega XML dokumenta

Z razpoznavalnikom HtmlCleaner lahko razvijamo aplikacije v programskem jeziku Java kot konzolne programe ali Ant naloge. HtmlCleaner je zasnovan tako, da je majhen, neodvisen, hiter in fleksibilen. Čeprav je glavni namen razpoznavalnika HtmlCleaner pretvorba HTML dokumentov v XML dokumente, ki jih lahko procesiramo z XPath, XQuery, in XSLT, lahko podatke tudi procesiramo in obravnavamo na mnoge druge načine [17].

4.2 Ostali razpoznavalniki

- NekoHTML

NekoHTML je preprosti skener, ki omogoča programerjem razpoznavanje HTML dokumentov in dostop do informacij z uporabo XML vmesnikov. NekoHTML lahko popravi dosti napak v HTML datotekah, ki so jih napravili programerji pri pisanju HTML dokumentov. Razpoznavalnik samodejno doda manjkajoče elemente (značke) in zapre elemente z opcijskimi končnimi značkami [18].

NekoHTML je napisan z uporabo XNI (angl. Xerces Native Interface), ki je zgrajena na osnovi Xerces2 razpoznavalnika. To nam omogoča, da lahko razpoznavalnik NekoHTML uporabljamo z obstoječimi XNI orodji, ne da bi bilo potrebno spreminjati kodo [18].

- HTML Parser

HTML Parser je Java knjižnica, ki se uporablja za razpoznavanje HTML dokumentov. Primarno se uporablja za transformacijo ali pridobivanje vsebin. HTML Parser vsebuje filtre, po meri značke in je preprost za uporabo JavaBeans. Je hiter, robusten in enostaven realno časovni razpoznavalnik, s katerim je mogoče razpoznavati zelo kompleksne HTML dokumente.

Dva temeljna primera uporabe razpoznavalnika sta transformacija in pridobivanje. Transformacija vključuje vso procesiranje, kjer imamo za vhod in izhod HTML dokumente. Nekateri primeri so:

- prepisovanje URL naslova: spreminjanje nekaterih ali vseh URL naslovov
- zajemanje strani: premikanje vsebin s spleta na sekundarni pomnilnik
- cenzura: odstranjevanje neprimernih besed in fraz iz spletnih strani
- HTML čiščenje: popravljanje napak na spletni strani
- odstranjevanje reklam: odstranjevanje povezav, ki vsebujejo reklame.

Pridobivanje zajema vse programe za pridobivanje informacij, ki niso sposobni ohraniti izvorne strani. Kot na primer:

- pridobivanje besedila: za uporabo kot vhod za tekstovni iskalnik v podatkovnih bazah
- pridobivanje URL naslovov: za pridobivanje vseh URL in e-mail naslovov v spletni strani
- zaslonsko pridobivanje: za programski vnos podatkov na spletno stran
- pridobivanje virov: za pridobivanje slik in zvokov
- preverjanje povezav: za zagotavljanje pravilnih povezav [19].

- Jericho HTML Parser

Jericho HTML Parser je Java knjižnica, ki omogoča analizo in manipulacijo z deli HTML dokumenta in vključevanje značk. Razpoznavalnik tudi ponuja visoko nivojske funkcije za manipulacijo HTML dokumentov. Jericho HTML Parser je odprto-kodna knjižnica izdana pod licenco EPL (angl. Eclipse Public License) in LGPL (angl. GNU Lesser General Public License).

Od ostalih razpoznavalnikov se razlikuje predvsem po:

- slabo oblikovani HTML dokumenti nimajo vpliva na razpoznavalnik
- razpoznavalnik je sposoben prepoznati ASP, JSP, PSP, PHP in Mason strežniške značke, kakor tudi HTML značke
- omogoča tokovno razpoznavanje z uporabo razreda »StreamedSource«, ki zagotavlja spominsko učinkovito procesiranje velikih količin podatkov
- v standardni obliki ne temelji na dogodkovnem ali drevesnem razpoznavanju, ampak uporablja kombinacijo preprostega tekstovnega iskalnika, učinkovitega prepoznavalnika značk in predpomnilnik za položaj značk
- definiramo lahko poljubne značke, ki jih nato uporabljamo pri razpoznavanju HTML dokumentov [20].

- HotSAX

HotSAX je majhen in hiter SAX2 razpoznavalnik za HTML, XHTML in XML dokumente. SAX (angl. Simple API for XML) razpoznavalnik je razvil David Megginson v sodelovanju z ostalimi razvijalci. SAX razpoznavalnik razpoznava XML dokumente z generiranjem dogodkov za začetne značke, besedilo in značke, ki sprožijo nek dogodek. SAX razpoznavalniki so hitrejši in porabijo manj pomnilnika od enakovrednih DOM razpoznavalnikov. SAX2 še doda leksikalno upravljanje razširitev kot na primer komentarji in CDATA bloki.

Do zdaj smo potrebovali dobro strukturiran XML dokument, da smo ga lahko razpoznali s SAX razpoznavalnikom. Z uvedbo HotSAX razpoznavalnika pa lahko razpoznavamo tudi slabo strukturirane HTML dokumente [21].

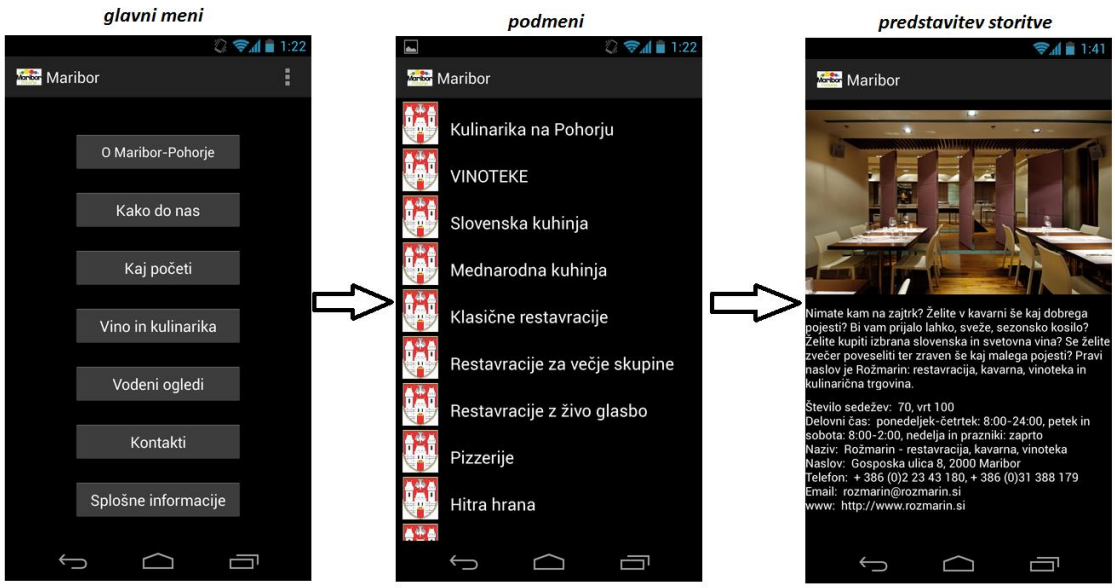
5 NAČRTOVANJE IN IMPLEMENTACIJA APLIKACIJE

Namen diplomske naloge je bil izdelati aplikacijo za mobilne naprave z operacijskim sistemom Android. Aplikacija prikazuje vse informacije o mestu Maribor in Pohorju, ki so pomembne za turiste.

Tako smo se odločili, da prenesemo spletno aplikacijo v mobilno aplikacijo, saj vse več ljudi uporablja mobilne naprave, ki jim omogočajo dostop do potrebnih informacij kjerkoli. Podatke pridobivamo iz spletne strani http://maribor-pohorje.si, z uporabo razpoznavalnika HtmlCleaner.

5.1 Izdelava uporabniškega vmesnika aplikacije Maribor-Pohorje

Najprej smo izdelali okvirni načrt naše aplikacije, ki prikazuje različne aktivnosti in njihove funkcionalnosti. Naša aplikacija vsebuje glavni meni, podmenije in predstavitve raznovrstnih storitev (slika 5).



Slika 5: Struktura aplikacije

Nato smo ustvarili Android projekt v razvojnem okolju Eclipse. Uporabili smo verzijo 4.2.2 Androida (Jelly Bean). Projekt, ki smo ga ustvarili, vsebuje osnovni pogled (Layout), ki smo ga spremenili v glavni meni in razred, v katerega smo implementirali funkcionalnosti menija (slika 6).

Slika 6: Izgled glavnega menija

V pogled (angl. Layout) za glavni meni smo vstavili sedem gumbov (slika 6), katerim smo nato spremenili videz. Za ustrezno delovanje gumbov smo morali za vsak gumb ustvariti metodo, katero smo nato implementirali v razredu tega pogleda. V metode onClick() smo implementirali kodo, ki odpre novo aktivnost, ko pritisnemo na željen gumb, kar lahko vidimo na primeru kode 10.

```
public void onClickOMariborPohorje(View view) {
    Intent intent = new Intent(this, ActivityListViewOpis.class);
    this.startActivity(intent);
}
```

Primer kode 10: Primer metode OnClick() in zagon nove aktivnosti

Nato smo naredili seznam (angl. ListView) s katerim lahko izbiramo vrsto informacij, katere si bomo pogledali podrobneje. Seznam smo naredili tako, da smo v pogled vstavili ListView, kateremu smo nastavili lastnost »listitem« na preprosti enovrstični seznam (primer kode 11).

```
<ListView
    android:id="@+id/listViewListView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true"
    android:background="@color/black"
    android:cacheColorHint="?android:attr/colorBackgroundCacheHint"
    tools:listitem="@android:layout/simple_list_item_1" >
</ListView>
```

Primer kode 11: Primer seznama ListView

Za tem smo seznam implementirali v razredu tega pogleda. Gradnik v pogledu smo povezali s spremenljivko razreda ListView. S tem smo določili kateri seznam bomo uporabili (primer kode 12).

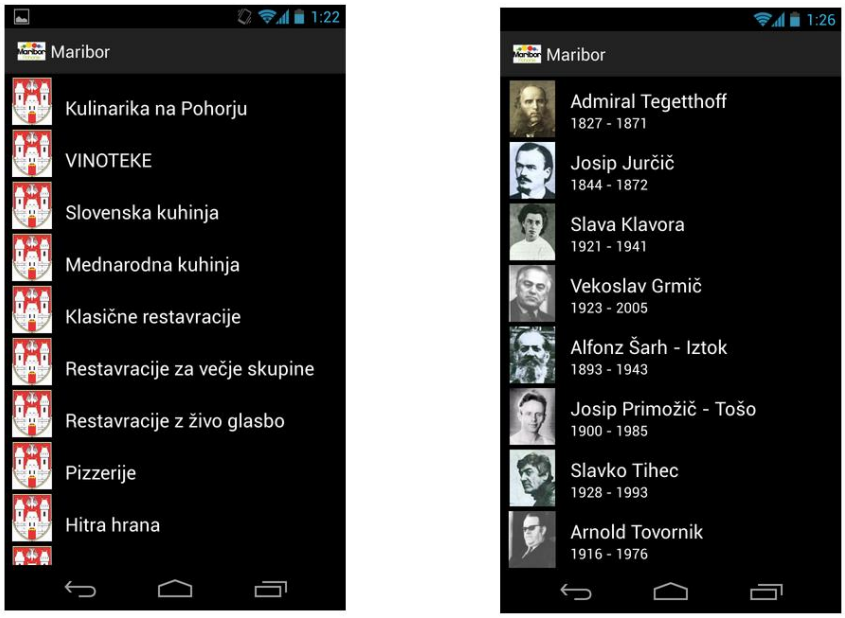
```
listview = (ListView) findViewById(R.id.listViewListView);
```

Primer kode 12: Inicializacija seznama

Seznam smo implementirali tako, da smo vstavili metodo »setAdapter()« v kateri smo določili kakšen tip seznama bomo uporabljali, v katerem pogledu bomo uporabili seznam, ID seznama in vhodne podatke. Nato smo še implementirali vmesnik »OnItemClickListener« v katerega smo vstavili metodo »onItemClick()«. V metodo »onItemClick()« smo dali kodo, ki se bo izvedla glede na to na kateri element v seznamu bomo pritisnili, kar je razvidno iz primera kode 13. Na sliki 7 lahko vidimo preprosti seznam (slika levo) in dvovrstični seznam (slika desno).

```
listview.setVisibility(View.VISIBLE);
listview.setAdapter(new ArrayAdapter<String>(ActivityListView.this,
        R.layout.activity_list_view, R.id.textViewListView, output));
listview.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
            int position, long id) {
    }
});
```

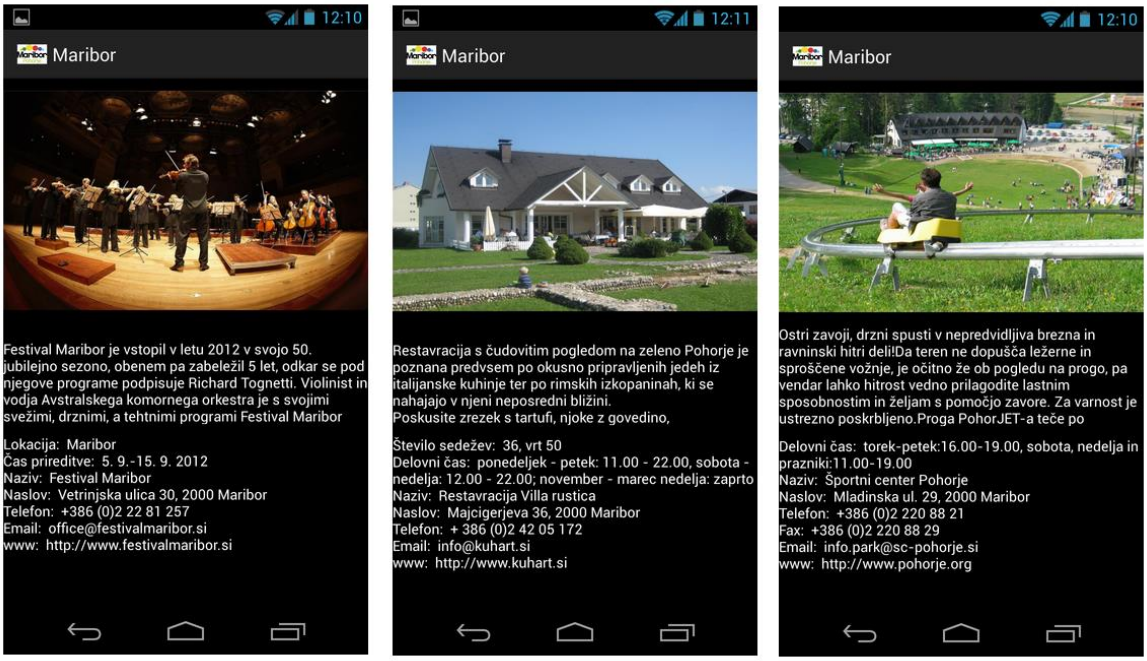
Primer kode 13: Implementacija seznama



Slika 7: Izgled seznama

Nato smo implementirali podroben prikaz informacij z uporabo gradnikov »ImageView« in »TextView«. V primeru, da imamo daljšo besedilo kot je velikost našega ekrana, smo uporabili gradnik »ScrollView«, ki nam omogoči prikaz daljših besedil.

Na sliki 8 lahko vidimo uporabljene gradnike »ImageView« za prikaz slik, »TextView« za prikaz besedila in »ScrollView« za prikaz besedil, ki so večja, kot je fizična velikost ekrana mobilne naprave.



Slika 8: Izgled opisov storitev

5.2 Implementacija funkcionalnosti aplikacije

V nadaljevanju smo implementirali razpoznavalnik HtmlCleaner, s pomočjo katerega smo pridobivali podatke iz spletne strani http://maribor-pohorje.si/. Za ustrezno delovanje razpoznavalnika smo uporabili razred »AsyncTask«, ki omogoči, da se proces izvaja v ozadju aplikacije.

Razpoznavalniku smo kot vhodni podatek podali URL naslov spletne strani, iz katere smo želeli pridobiti podatke. Nato je razpoznavalnik pretvoril HTML vsebino v dobro strukturiran XML dokument. Podatke iz XML dokumenta smo pridobili s pomočjo XPath izrazov, ki smo jih shranili v seznam. Celoten proces se je izvajal v metodi »doInBackground«, ki je na koncu vrnila seznam podatkov, pridobljenih iz XML dokumenta s pomočjo XPath izrazov (primer kode 14).

```
private class Parser extends AsyncTask<String, Void, List<String>> {
    @Override
    protected List<String> doInBackground(String... arg) {
        List<String> output = new ArrayList<String>();
        try {

            HtmlCleaner htmlCleaner = new HtmlCleaner();
            CleanerProperties props = htmlCleaner.getProperties();
            props.setAllowHtmlInsideAttributes(false);
            props.setAllowMultiWordAttributes(true);
            props.setRecognizeUnicodeChars(true);
            props.setOmitComments(true);

            URL url = new URL(arg[0]);

            TagNode root = htmlCleaner.clean(url);

            Object[] nodes = root.evaluateXPath(arg[1]);

            for (int i = 0; i < nodes.length; i++) {
                output.add(((TagNode) nodes[i]).getText().toString().trim());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return output;
    }
```

Primer kode 14: Primer uporabe »AsyncTask« in razpoznavalnika HtmlCleaner

XPath je povpraševalni jezik, ki se uporablja za izbiranje značk iz XML dokumentov. V našem primeru smo uporabili preproste XPath izraze, kjer smo navedli značko in ime razreda (primer kode 15).

```
"//div[@class='content-listPhoto-title']"
```

Primer kode 15: Primer XPath izraza

Za zagon razpoznavalnika smo uporabili metodo »execute()« kateri smo podali za argumente URL naslov spletne strani, XPath izraze in ime atributa, iz katerega smo dobili podatke (primer kode 16).

```
new Parser().execute(
        ActivityListView.getUrls().get(
                ActivityListView.getPozicija()),
        "//div[@class='content-listPhoto-title']",
        "//td[@class='content-listPhoto-div']/a",
        "http://maribor-pohorje.si", "href");
```

Primer kode 16: Primer zagona razpoznavalnika

Ko se je razpoznavanje zaključilo, se je poklicala metoda »onPostExecute()« kateri so bili posredovani podatki, ki smo jih nato prikazali na zaslon, kar je razvidno iz primera kode 17.

```
protected void onPostExecute(final List<String> output) {
    String text = "";
    for (int i = 0; i < output.size(); i++) {
        text += output.get(i) + "  " + outputStr.get(i) + "\n";
    }
    tv1.setText(text);
    tv2.setText(opis);

    if (outputImageUrls.size() > 0) {
        new DownloadImageTask((ImageView) findViewById(R.id.imageView1))
                .execute(outputImageUrls.get(0));
    } else {
        pd.dismiss();
        image.setBackgroundResource(R.drawable.logo);
    }
}
```

Primer kode 17: Primer metode »onPostExecute()«

Slike v naši aplikaciji smo dobili na podoben način kot besedilo. S pomočjo razpoznavalnika HtmlCleaner smo pretvorili spletno stran v XML dokument, ki je vsebovala želene slike. Nato smo s pomočjo XPath izrazov pridobili URL naslove slik iz XML dokumenta. Preko vhodnega toka smo sliko prenesli s spleta in jo pretvorili v bitno sliko. Celoten proces je potekal znotraj AsyncTask in ko smo zaključili s prenosom slike, smo jo prikazali na zaslon (primer kode 18).

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    public DownloadImageTask(ImageView bmImage) {
    }

    protected Bitmap doInBackground(String... urls) {
        String urldisplay = urls[0];
        Bitmap picture = null;
        Bitmap scaledPicture = null;
        try {
            InputStream in = new java.net.URL(urldisplay).openStream();
            picture = BitmapFactory.decodeStream(in);
            scaledPicture = Bitmap.createScaledBitmap(picture, 800, 480, true);
        } catch (Exception e) {
            Log.e("Error", e.getMessage());
            e.printStackTrace();
        }
        return scaledPicture;
    }

    protected void onPostExecute(Bitmap result) {
        pd.dismiss();
        if (result != null)
            image.setImageBitmap(result);
        else
            image.setBackgroundResource(R.drawable.logo);
    }
}
```

Primer kode 18: Primer prenosa slike iz spleta

Ker aplikacija za svoje delovanje potrebuje internetno povezavo, smo v ta namen implementirali funkcionalnost, ki preveri, če imamo internetno povezavo. Uporabili smo razred »ConnectivityManager«, ki nam pošilja informacije o stanju internetne povezave. Tako v primeru, da nismo povezani na internet dobimo sporočilo, ki nas obvesti, da moramo vzpostaviti internetno povezavo, kar je razvidno iz primera kode 19.

```
private boolean haveNetworkConnection() {
    boolean haveConnectedWifi = false;
    boolean haveConnectedMobile = false;

    ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo[] netInfo = cm.getAllNetworkInfo();
    for (NetworkInfo ni : netInfo) {
        if (ni.getTypeName().equalsIgnoreCase("WIFI"))
            if (ni.isConnected())
                haveConnectedWifi = true;
        if (ni.getTypeName().equalsIgnoreCase("MOBILE"))
            if (ni.isConnected())
                haveConnectedMobile = true;
    }
    return haveConnectedWifi || haveConnectedMobile;
}
```

Primer kode 19: Preverjanje stanja internetne povezave

5.3 Uporabnost aplikacije

Aplikacija je namenjena vsem, ki želijo obiskati turistične destinacije v Mariboru in njegovi okolici. Aplikacija nudi informacije o:

- zgodovini Maribora
- geografski legi Maribora
- kulturnih in športnih prireditvah
- športnih in rekreacijskih destinacijah
- vinu in kulinariki
- kontaktni podatki
- ostale splošne informacije.

Ker gre za mobilno aplikacijo, je njena prednost, da lahko dostopamo do informacij kjerkoli. Za delovanje aplikacije potrebujemo le internetno povezavo. Maribor ponuja brezžično internetno povezavo po celotnem Mariboru, nudijo pa jo tudi različni gostinski lokali v Mariboru in okolici.

5.3.1 Prednosti aplikacije

- brezplačna
- preprosta za uporabo
- priročna
- primerna za uporabnike vseh starosti
- hiter dostop do informacij.

Prednosti naše aplikacije so, da je brezplačna in preprosta za uporabo. Omogoča hiter dostop do informacij in je namenjena uporabnikom vseh starosti. Aplikacija ima preprost grafični vmesnik, ki je preprost za uporabo.

Zaradi množične uporabe mobilnih naprav je naša aplikacija zelo priročna, saj lahko dostopamo do informacij kjerkoli.

5.3.2 Slabosti aplikacije

- za delovanje aplikacije potrebujemo internetno povezavo
- samo za Android mobilne naprave
- pridobivanje podatkov iz spletne strani.

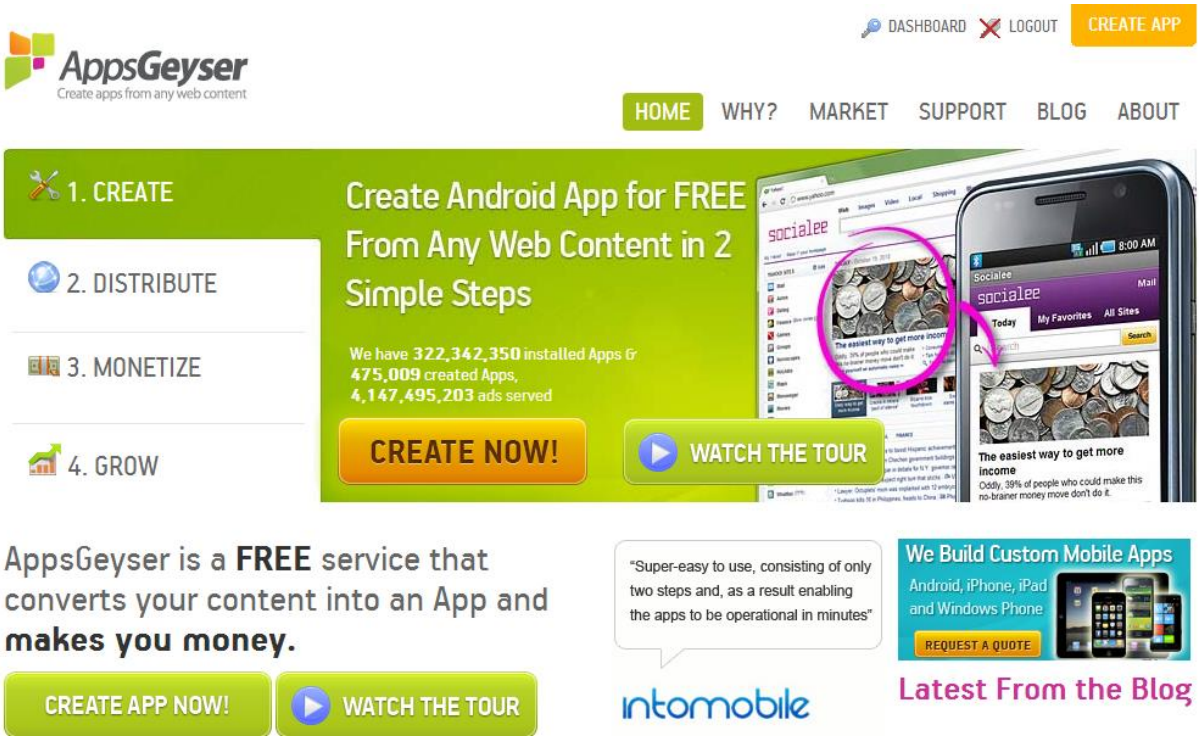
Slabosti, ki jih najdemo pri naši aplikaciji so, da za delovanje potrebuje internetno povezavo. Namenjena je samo mobilnim napravam z operacijskim sistemom Android. Glavna slabost naše aplikacije je odvisnost od spletne strani http://maribor-pohorje.si/ iz katere pridobivamo podatke. V primeru nedelovanje strežnika spletne strani ne bomo mogli dostopati do podatkov. V primeru spremembe strukture spletne strani bo treba posodobiti aplikacijo.

6 POSPLOŠEVANJE RAZPOZNAVANJA SPLETNIH VSEBIN

Kot alternativno rešitev in primerjavo naši aplikaciji smo se odločili preizkusiti storitev AppsGeyser, ki samodejno pretvori vsebino spletne strani v Android aplikacijo [22].

AppsGeyser je brezplačna storitev, ki nam omogoča pretvorbo spletnih vsebin v Android aplikacijo. Vse aplikacije, ki jih naredimo s platformo AppsGeyser, lahko prodajamo in izmenjujemo na spletu. To lahko storimo s pomočjo AppsGeyser ali Google Play spletne trgovine.

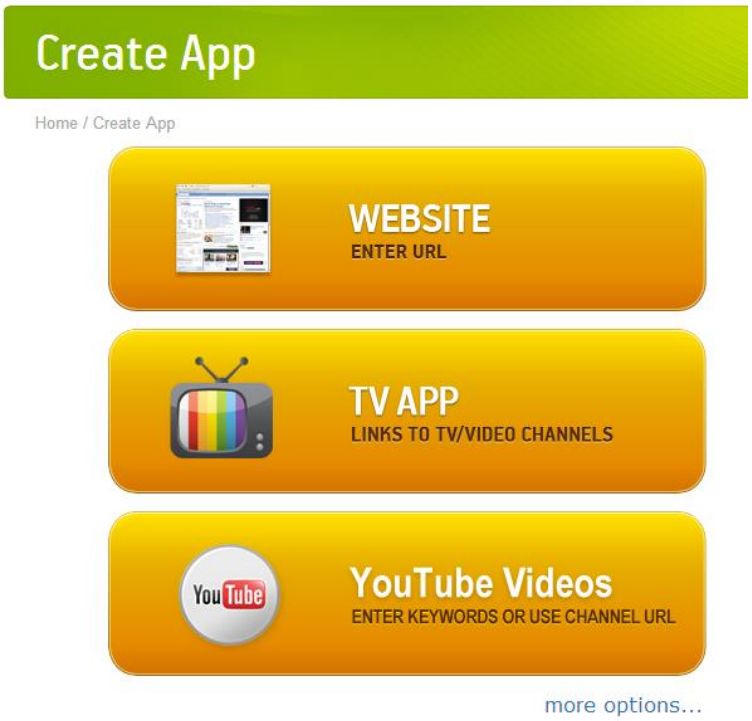
Aplikacije lahko ustvarimo s pomočjo platforme AppsGeyser v dveh korakih in v zelo kratkem času (nekaj minut). Da naredimo aplikacijo, ne potrebujemo znanja iz programiranja. Vse kar moramo storiti je, da vnesemo URL spletne strani. Platforma omogoča distribucijo in prodajo aplikacij.



Slika 99: Spletna stran AppsGeyzer [22]

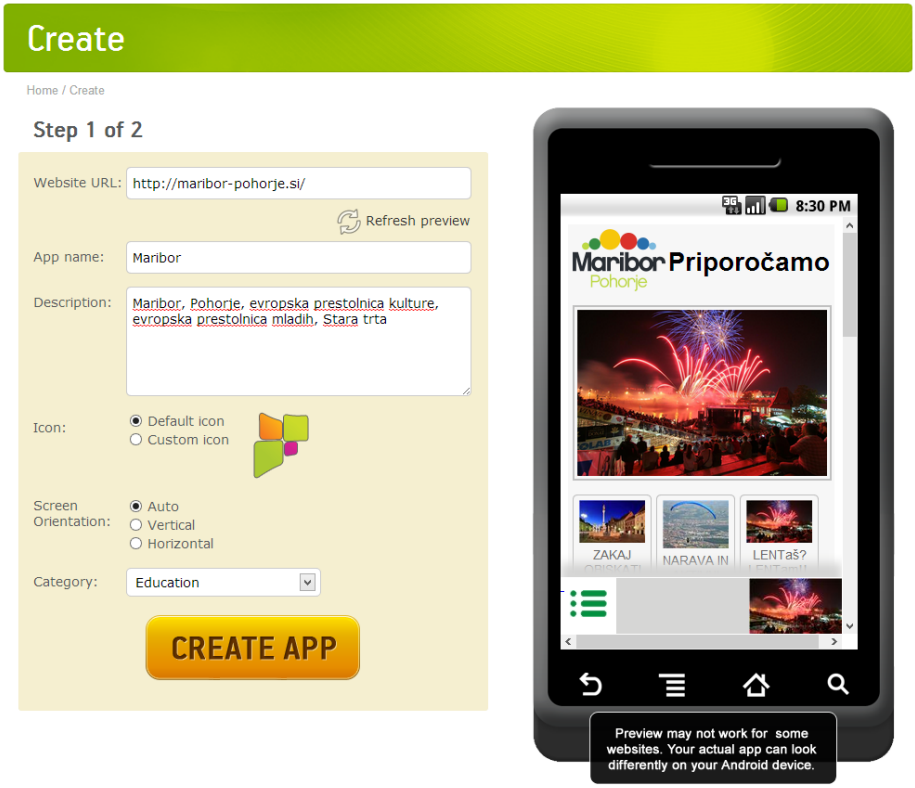
6.1 Ustvarjanje aplikacije s platformo AppsGeyzer

Če želimo ustvariti Android aplikacijo s pomočjo platforme AppsGeyzer, moramo obiskati spletno stran http://www.appsgeyser.com/, kjer pritisnemo na gumb »create app«. Odpre se nam okno, kjer izberemo aplikacijo, ki jo želimo narediti. Mi smo naredili aplikacijo iz spletne vsebine (slika 10).



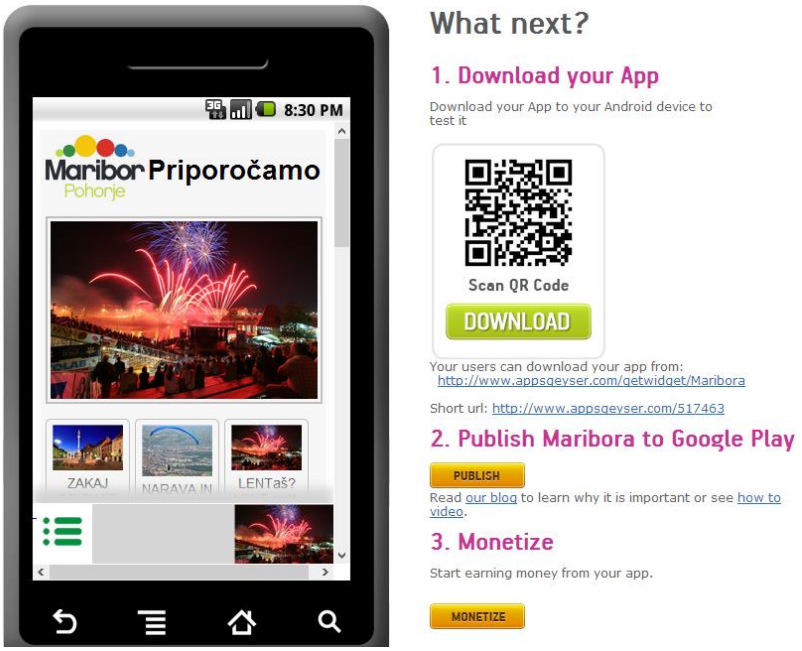
Slika 10: Izbira tipa aplikacije [22]

Nato moramo vnesti URL naslov spletne strani, dati aplikaciji ime in jo opisati. Nastavimo lahko ikono aplikacije, orientacijo zaslona in izberemo kategorijo v katero spada naša spletna vsebina. Na desni strani imamo prikaz, kako naj bi naša aplikacija izgledala (slika 11).



Slika 11: Ustvarjanje aplikacije [22]

Ko smo izpolnili vsa polja, lahko ustvarimo aplikacijo. Nato moramo aplikacijo samo še namestiti na našo mobilno napravo (slika 12).

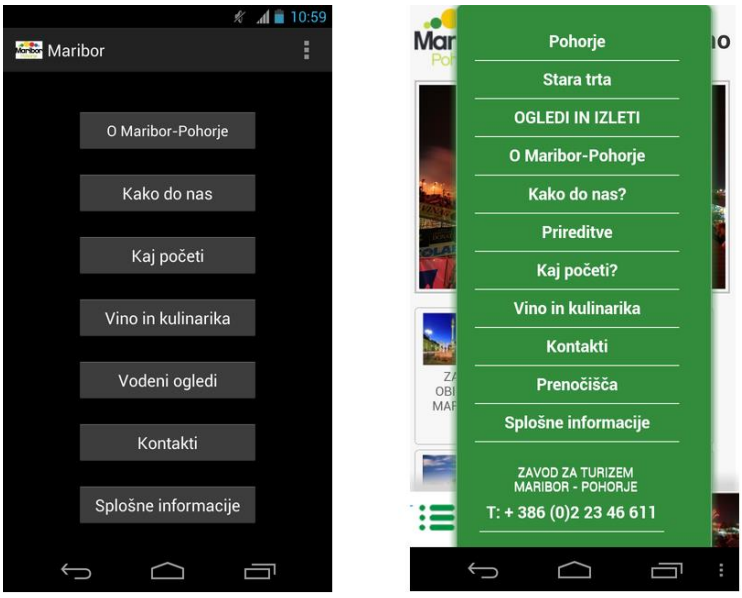


Slika 12: Prenos aplikacije [22]

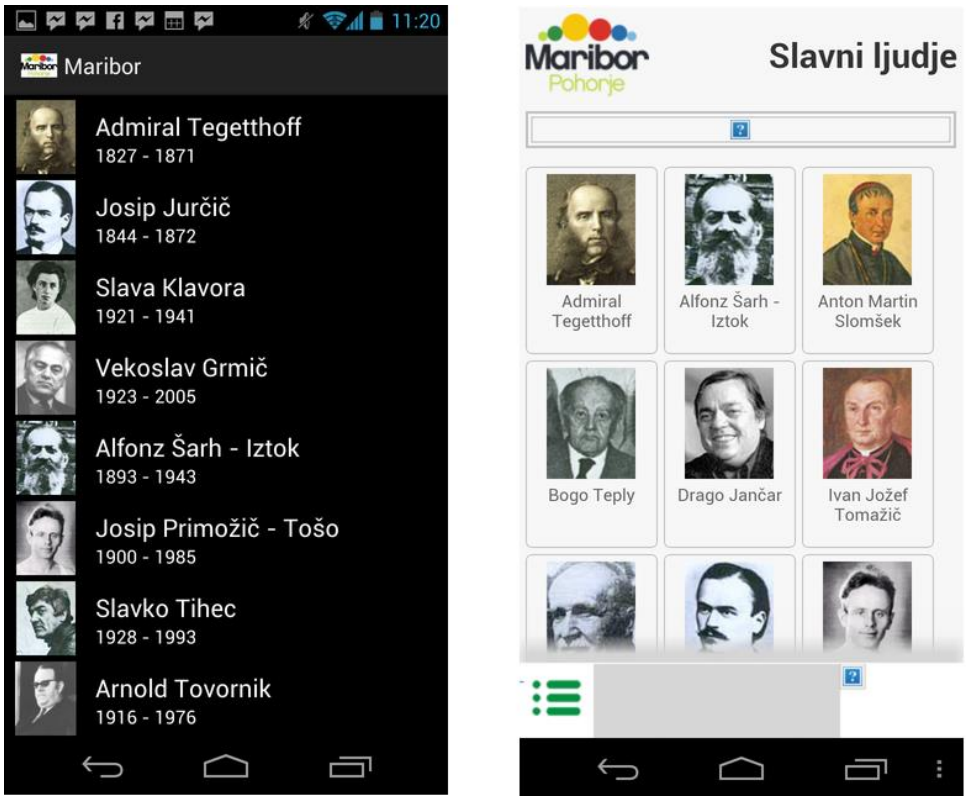
6.2 Primerjava aplikacij

Največja razlika med aplikacijama je v načinu izdelave aplikacije. Za našo aplikacijo potrebujemo znanje iz programiranja aplikacij za Android, medtem pa za izdelavo aplikacije s pomočjo platforme AppsGeyzer ne potrebujemo nobenega znanja iz programiranja. Druga večja razlika je časovna zahtevnost izdelave aplikacije. Za izdelavo naše aplikacije smo potrebovali nekje en mesec, za izdelavo aplikacije z AppsGeyzerjem pa smo potrebovali le nekaj minut.

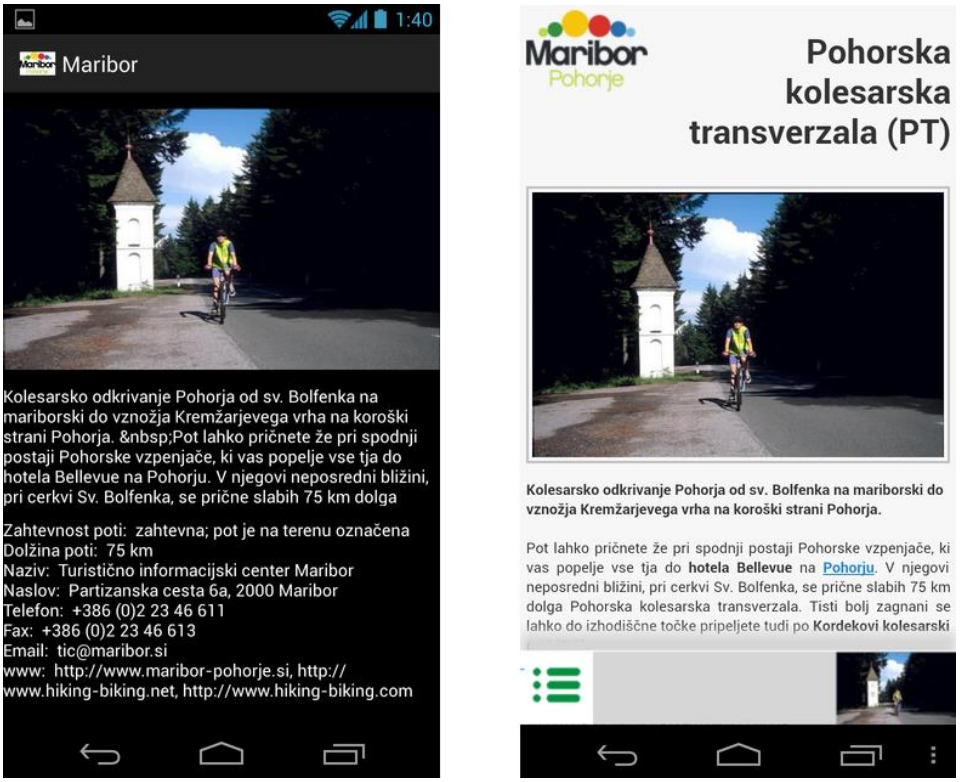
Razlike se tudi pojavijo v uporabniškem vmesniku aplikacije. V naši aplikaciji smo uporabili gumbe za izdelavo glavnega menija. V aplikaciji, narejeni s pomočjo AppsGeyzerja, pa je uporabljen seznam. Slabost aplikacije, narejene s platformo AppsGeyzer, je, da nimamo vpliva pri oblikovanju uporabniškega vmesnika aplikacije. Prav tako ne moremo odpraviti napak, ki se pojavijo v aplikaciji. Razlog, da nimamo možnosti popravljati in spreminjati aplikacije, je v načinu ustvarjanja aplikacije s platformo AppsGeyzer. AppsGeyzer nam samo ustvari .apk datoteko, ki je ne moremo več spreminjati. Tako je velika prednost izdelave naše aplikacije v tem, da imamo možnost oblikovati poljuben uporabniški vmesnik in pri tem tudi odpravljati morebitne napake, ki se pojavijo v aplikaciji. Na slikah 13, 14 in 15 lahko vidimo primerjavo aplikacij (levo je naša aplikacija in desno je aplikacija, narejena s platformo AppsGeyzer).



Slika 13: Primerjava menija



Slika 14: Primerjava seznamov



Slika 15: Primerjava opisa storitev

7 ZAKLJUČEK

V diplomski nalogi smo opisali operacijski sistem Android ter razvojno okolje in tehnologije, ki smo jih uporabili. S pomočjo teh orodij in tehnologij smo razvili aplikacijo, ki teče na operacijskem sistemu Android.

Diplomsko nalogo smo razdelili na dva dela, na teoretični in praktični del. V teoretičnem delu smo opisali operacijski sistem Android, razvojno okolje Eclipse, orodja za razvoj Android aplikacij in razpoznavalnik HtmlCleaner. Podrobneje smo opisali zgradbo operacijskega sistema Android, katere komponente so pomembne v Android aplikacijah in kako razvijamo aplikacije s pomočjo Android orodij. Primerjali smo razpoznavalnik HtmlCleaner z ostalimi razpoznavalniki. S tem smo dobili teoretično znanje, ki smo ga nato uporabili pri izvedbi praktičnega dela, to je implementacija aplikacije.

V praktičnem delu smo implementirali aplikacijo za operacijski sistem Android. Aplikacija, ki smo jo naredili, omogoča prikaz podatkov iz spletne strani http://maribor-pohorje.si/ na mobilnih napravah. Aplikacija nam prikazuje informacije o Mariboru, športnih in kulturnih prireditvah, turizmu, rekreaciji in zabavi, kulinariki ter ostalih splošnih informacijah. Podatke, ki smo jih prikazali v naši aplikaciji, smo dobili iz spletne strani http://maribor-pohorje.si/ s pomočjo razpoznavalnika HtmlCleaner. Aplikacija vsebuje preprost uporabniški vmesnik, ki je primeren za uporabnike vseh starosti.

V nadaljevanju smo naredili primerjavo med našo aplikacijo in aplikacijo, ki smo jo ustvarili s pomočjo platforme AppsGeyzer. Ugotovili smo, da je aplikacija narejena s pomočjo platforme AppsGeyzer preprostejša, saj ne potrebujemo znanja iz programiranja. Prav tako je njena izdelava hitrejša, vendar pa pri tem nimamo vpliva na videz in delovanje aplikacije.

Skozi praktični del diplomske naloge smo ugotovili, da bi v našo aplikacijo lahko vključili dodatne funkcionalnosti. Vključili bi lahko funkcionalnost, ki bi uporabniku aplikacije prikazala, katere dele mesta in znamenitosti je že obiskal. S tem bi uporabnik vedel, katerih

delov mesta in znamenitosti še ni obiskal. Za še bolj uporabniško prijazno aplikacijo bi lahko dodali funkcionalnost »priljubljene«, ki bi prikazovala tiste informacije, do katerih je uporabnik največkrat dostopal. Določene informacije bi lahko shranili v podatkovno bazo na mobilni napravi. To bi uporabniku omogočilo uporabo aplikacije tudi, kadar nismo povezani na internet. Kot izboljšavo za našo aplikacijo bi lahko dodali večjezičnost, kar bi še dodatno povečalo njeno uporabnost. Tako bi lahko naša aplikacija zaživela tudi na tujem trgu.

V prihodnosti se bo uporaba mobilnih naprav še povečala in potreba po mobilnih aplikacijah. V ta namen bi lahko našo aplikacijo posplošili tako, da bi lahko pretvorili katerokoli spletno stran v Android aplikacijo. S tem bi dodatno povečali uporabnost in število uporabnikov aplikacije.

8 VIRI

[1] Slovenija - Mariborsko Pohorje,
http://www.intelekta.eu/wellness/slovenija/mariborsko-pohorje

[2] Kaj je Android?,
http://slo-android.si/prispevki/kaj-je-android.html

[3] Android version history,
http://en.wikipedia.org/wiki/Android_version_history

[4] Android (operating system),
https://en.wikipedia.org/wiki/Android_(operating_system)

[5] Android Architecture,
http://www.android-app-market.com/android-architecture.html

[6] Android Architecture,
http://www.j2eebrain.com/java-J2ee-android-architecture.html

[7] Application Fundamentals
http://developer.android.com/guide/components/fundamentals.html

[8] Activities,
http://developer.android.com/guide/components/activities.html

[9] Services,
http://developer.android.com/guide/components/services.html

[10] Content Providers,
http://developer.android.com/guide/topics/providers/content-providers.html

[11] Processes and Threads,
http://developer.android.com/guide/components/processes-and-threads.html

[12] Eclipse (software),
http://en.wikipedia.org/wiki/Eclipse_(software)

[13] Android SDK,
http://developer.android.com/sdk/index.html

[14] Managing Virtual Devices,
http://developer.android.com/tools/devices/index.html

[15] Using DDMS,
http://developer.android.com/tools/debugging/ddms.html

[16] Java (programming language),
http://en.wikipedia.org/wiki/Java_(programming_language)

[17] HtmlCleaner,
http://htmlcleaner.sourceforge.net/

[18] NekoHTML,
http://nekohtml.sourceforge.net/

[19] HTML Parser,
http://htmlparser.sourceforge.net/

[20] Jericho HTML Parser,
http://jericho.htmlparser.net/docs/index.html

[21] HotSAX,
http://hotsax.sourceforge.net/

[22] AppsGeyzer,
http://www.appsgeyser.com/

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko

Smetanova ulica 17
2000 Maribor

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Aleksander Robnik,

z vpisno številko E1041525,

sem avtor/-ica diplomskega dela z naslovom:

Android aplikacija z uporabo razpoznavanja spletne vsebine

S svojim podpisom zagotavljam, da:

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

  doc. dr. **Tomaž Kosar**, univ.dipl.inž. rač. in inf.

  in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.

- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 9.9.2013 Podpis avtorja/-ice: Aleksander Robnik

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko

Smetanova ulica 17
2000 Maribor

IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Podpisani mentor Tomaž Kosar izjavljam, da je
(ime in priimek mentorja)
študent Aleksander Robnik izdelal diplomsko
(ime in priimek študenta-tke)

delo z naslovom: Android aplikacija z uporabo razpoznavanja spletne vsebine

(naslov diplomskega dela)

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

Datum in kraj: 9.9.2013

Podpis mentorja:

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko

Smetanova ulica 17
2000 Maribor

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE DIPLOMSKEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek diplomanta-tke: Aleksander Robnik

Vpisna številka: E1041525

Študijski program: Računalništvo in informacijske tehnologije VS

Naslov diplomskega dela: Android aplikacija z uporabo razpoznavanja spletne vsebine

Mentor: doc. dr. **Tomaž Kosar**, univ.dipl.inž. rač. in inf.

Somentor: _____

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

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

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

Datum in kraj: 9.9.2013

Podpis diplomanta-tke: Aleksander Robnik