Bojan Orter

MOBILNA APLIKACIJA ZA OBVEŠČANJE O DNEVNIH PONUDBAH

Diplomsko delo

Maribor, september 2013

Diplomsko delo visokošolskega strokovnega študijskega programa

MOBILNA APLIKACIJA ZA OBVEŠČANJE O DNEVNIH PONUDBAH

Študent:	Bojan Orter
Študijski program:	VS ŠP Računalništvo in informacijske tehnologije
Smer:	Pametni telefoni
Mentor(ica):	Doc. dr. Tomaž Kosar
Lektor(ica):	dip. sloveniska (UNI) Petra Potočnik

Maribor, september 2013

Univerza v Mariboru

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

FERI

Številka: E1041547
Datum in kraj: 19. 04. 2013, Maribor

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

SKLEP O DIPLOMSKEM DELU

1. **Bojanu Orterju,** 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:**
MOBILNA APLIKACIJA ZA OBVEŠČANJE O DNEVNIH PONUDBAH

4. **Naslov diplomskega dela v angleškem jeziku:**
MOBILE APPLICATION FOR DAILY OFFERS

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

UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO
8

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

MOBILNA APLIKACIJA ZA OVEŠČANJE O DNEVNIH PONUDBAH

Ključne besede: **razvoj aplikacij, mobilne naprave, razpoznavanje besedila**

UDK: 004.5:621.39(043.2)

Povzetek

Izdelali smo aplikacijo, ki omogoča prejemanje obvestil o novih ponudbah za pametne telefone. Pri tem spremljanje ponudb opravlja strežnik, medtem ko odjemalec, v našem primeru mobilni telefon z operacijskim sistemom Android, omogoča pregled trenutnih ponudb. Pomembnejši del sistema predstavlja še možnost, da se uporabnik naroči na ponudbe, o katerih želi biti obveščen. Naloga strežnika je, da spremlja dodajanje novih ponudb in o tem sproti obvešča odjemalce, ki so na te ponudbe naročeni.

Za potrebo strežnika smo proučili Microsoft WCF. Za komunikacijo med odjemalcem in strežnikom smo uporabili Google GCM, ki omogoča obveščanje naprav, katere temeljijo na operacijskem sistemu Android. Za samo komunikacijo med strežnikom in odjemalcem, smo uporabili json format (Java-script object notation), ki je namenjen serializaciji *objektov.*

MOBILE APPLICATION FOR DAILY OFFERS

Key words: **application deployment, mobile devices, identifying text**

UDK: 004.5:621.39(043.2)

Abstract

We created an application, that enabels us to receive notifications of new offers on mobile phone. Monitoring of new offers is done with an application on our server side. Client application, that runs on Android OS, allows us to view active offers. Users have the possibility to subscribe for specific offers. When new offer arrives, server notifys all subscribed users.

For server requirements we used WFC (Windows Communication Foundation) service. Also we used Google service GCM (Google cloude messeging), that enables us to send notifications to Android devices. For server-client communication we used Json format (Java-script object notation), that is used for serialization of objects.

KAZALO VSEBINE

KAZALO SLIK

UPORABLJENE KRATICE

OS – Operacijski Sistem

UV – Uporabniški Vmesnik

SOA – Service-Oriented Architecture

SOAP – Simple Object Access Protocol

WCF – Windows Communication Foundation

XML – Extensible Markup Language

JSON – JavaScript Object Notation

API – Application Programming Interface

TCP – Transmission Control Protocol

MIME- Multipurpose Internet Mail Extensions

GCM – Google Cloud Messaging

OS – Operacijski Sistem

HTTP – HyperText Transfer Protocol

TCP – Transmission Control Protocol

SD – Secure Digital

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

SQL –Structured Query Language

1 UVOD

Računalništvo se v sedanjih časih zelo hitro razvija in širi. Trend širjenja je na vedno manjše a vseeno zmogljive naprave, kot so telefoni. S tem dobivamo zelo priročne večnamenske naprave, s katerim lahko že opravljamo skoraj vsa opravila, za katera je prej bil namenjen osebni računalnik.

Ob današnjem hitrem tempu življenja je zelo priročno, da lahko spremljamo novice in prebiramo različne oglase, spletne strani, elektronsko pošto, kar na poti. Predvsem smo vedno bolj povezani z našim telefonom, saj nam služi kot organizator, ki name je vedno pri roki, zaradi česar je zelo koristen. Velikokrat si želimo avtomatskega obveščanja, ki lahko tako zamenja ponavljajoča opravila.

Lahko se informiramo o naših hobijih, o novicah, ki so bile pravkar objavljene, o razvoju tehnologije, kateri sledimo. Obveščanje je lahko koristno tudi pri službenih opravilih, za sprejeto naročilo, odposlano naročilo, itd. Še posebej pa je koristno, če smo obveščeni o ponudbah, ki se iztečejo in so nam trenutno v interesu.

Pri diplomskem delu smo se odločili za zadnji tip obveščanja - za obveščanje o novih ponudbah. Pri tem smo dobili idejo o obveščanju o različnih popustih, ki so časovno, včasih pa tudi količinsko omejeni. Gre za spletne ponudnike, ki ponujajo kupone za popust različnih storitev, dejavnosti, hrane, itd. Ti omogočajo naročanje in prejemanje ponudb na preko elektronsko pošto. A veliko ljudi elektronske pošte ne spremlja redno, zato lahko zamudijo ponudbo, ki bi jim bila všeč. V tem primeru pride v prid obveščanje na telefon, saj imamo telefon veliko časa kar pri sebi.

Ideja aplikacije je, obveščanje na telefonu - ko bo nova ponudba dodana, bomo o tem obveščeni.

Ker pa današnji telefoni še niso dovolj zmogljivi, da bi opravljali težja opravila je glavni del programa narejen na strežniku. S tem bo telefon olajšan težjih opravil in odvečne porabe baterije.

V diplomskem delu smo se ukvarjali tudi s komunikacijo med različnimi platformami. Za odjemalca smo uporabili popularen operacijski sistem Android, med tem ko smo za strežnik uporabili WCF (angl. Windows Communication Foundation).

Za komunikacijo med platformama smo skrbeli s formatom Json (Java-Script object notation), ki omogoča serializacijo objektov [8].

Del aplikacije je bil opravljen v Google- tehnologiji GCM (angl. Google Cloud Messaging), ki skrbi zato, da so uporabniki obveščeni, ko se z mobilnikom povežejo v svetovni splet.

2 ANDROID

2.1 Android splošno

Android je operacijski sistem (OS), ki je zasnovan na platformi Linux [2]. Prvotno je zasnovan za mobilne naprave na dotik, kot so pametni telefoni in tablice. Razvit je bil s strani podjetja Android Inc., katerega je financiral Google, v letu 2005 pa ga tudi kupil. Širši javnosti je bil predstavljen leta 2007. Poleg samega Android OS je potrebno omeniti še ustanovitev konzorcija Open Handset Alliance. Gre za konzorcij podjetij za strojno opremo, programsko opremo in medmrežno komunikacijo, ki je pomemben za razvoj odprto kodne programske opreme za pametne telefone.

Celoten Android OS je odprtokoden, saj je Google kodo izdal pod licenco Apache License. S tem je omogočeno, da je operacijski sistem svobodno spremenjen in razdeljen od proizvajalcev naprav in razvijalcev.

Moč Android se kaže v velikosti njegove družbe razvijalcev dodatnih funkcionalnosti. To je pripomoglo k temu, da Android velja za najbolj razširjen OS za pametne telefone. Leta 2010 je prehitel Symbian. Sam OS pa ni omejen samo na pametne mobilne naprave, ampak se zanj vse bolj odločajo tudi podjetja, ki izdelujejo nizko cenovne visokotehnološke naprave, ki podpirajo »lahek« OS, saj si s tem prihranijo čas, da bi pričeli razvijati OS iz nič. Kot rezultat tega se Android vedno pogosteje pojavlja kot OS naprav, kot so televizorji (iTV), igralne konzole, digitalne kamere.

Odprtost Android-a prav tako vzpodbuja razvijalce in ostale navdušence, da se ta uporabi kot temelj za razvoj družbenih odprtih projektov.

Google izboljšave razvija za zaprtimi vrati. Ko je izboljšava končana, je celotna koda predstavljena javnosti, s čemer postane prosto dostopna. Sama koda, torej koda brez sprememb, takšno kot jo izda Google, deluje le na določenih telefonih, ponavadi na tistih, ki so serije Nexus. Za ostale znamke mora proizvajalec sam dodati popravke, da ta deluje tudi na njegovih napravah.

Prav posodobitve bi lahko opredelili kot eno slabših lastnosti Android OS. Google ponavadi poskrbi, do te meje, da je verzija združljiva samo z njegovimi telefoni serije Nexus. Ostali uporabniki, ki pa uporabljajo druge proizvajalce, pa morajo na posodobitev za njihov mobilni telefon čakati dlje časa. Tako se velikokrat zgodi, da so nekateri telefoni lahko posodobljeni šele po kakšnem letu od izida same izboljšave. Včasih pride tudi do tega, da se proizvajalec odloči, da se posodobitve ne splača prilagajati telefonu. Razlog tiči predvsem v času in porabi virov za prilagoditve.

Za konec pa še navedimo nekaj statistike. V tretji četrtini leta 2012 je imelo 75% uporabnikov pametnih telefonov nameščen Android OS. Z 750 milijoni naprav in 1,5 milijona novih naprav vsak dan. Na uradni trgovini Google Play pa je prisotnih več kot 700.000 aplikacij. S tem je Android OS, največji OS na področju pametnih telefonov. [2].

2.2 Uporabniški vmesnik

Android uporabniški vmesnik (UV) je zasnovan za direktno manipulacijo in uporablja na dotik občutljive zaslone. Zaznava lahko različne geste s prsti, kot so pritiskanje, vlečenje s prsti in poteg prstov skupaj ter narazen. UV je zasnovan tako, da je njegov odziv takojšen in tekoč. Velikokrat uporabimo za povratno informacijo tudi funkcije kot so vibracije, pospeškomer (angl. Accelerometer), giroskop (angl. Gyroscope) ali senzor bližine (angl. proximity sensor). Te možnosti lahko uporabimo v naš prid recimo, ko želimo prilagoditi lego zaslona iz horizontalne v vertikalno. S tem je izkušnja OS še izboljšana.

Naprava najprej naloži domači zaslon. Ta zaslon je glavna navigacija za celoten telefon, primerjali ga bi lahko z namizjem OS Windows. Domači zaslon je ponavadi sestavljen iz ikon in pripomočkov (angl. Widgets). Ikone zaženejo program, med tem ko pripomočki prikazujejo živo vsebino, ki se sproti spreminja, tako na primer kažejo trenutno vreme ali stanje našega nabiralnika, če smo prejeli novo e-pošto. Velikokrat je domači zaslon sestavljen iz več namizij med katerimi se lahko pomikamo naprej in nazaj. Uporabnik si lahko ta namizja priredi po svoji želji. Teme s katerimi si lahko uporabniki uredijo domači zaslon je zelo veliko. Kar nekaj programov za urejanje tem lahko dobimo v trgovini Google Play. Nekatere teme posnemajo teme kot jo ima npr. Windows mobile. Na vrhu samega zaslona pa najdemo statusno vrstico. V tej vrstici so prikazana vsa opozorila, opomniki, itd. Statusna vrstica kaže kadar dobimo nov SMS, če imamo zgrešen klic, prav tako so v njej prikazana sporočila, ki jih tja vnese aplikacija. Za več podrobnosti lahko statusno vrstico povlečemo navzdol in vidimo podrobnosti sporočila. UV domačega zaslona lahko vidimo na spodnji sliki 2.1. [2].

Slika 2.1 Namizje Android [2]

2.3. Android aplikacija

Glavni gradnik Android aplikacij je aktivnost (angl. Activity). Aktivnost je komponenta aplikacije, ki zagotavlja zaslon, s katerim lahko uporabnik interaktivno komunicira. Vsakemu je dodeljeno okno, v katerega izriše UV. Okno pogosto zapolni celoten zaslon, vendar je lahko tudi manjše, v tem primeru pa je v ospredju na vrhu prejšnjega.

Aplikacija sestoji iz več aktivnosti, ki so med seboj povezane. V večini primerov imamo eno glavno aktivnost („Main“), ki se prikaže, ko je aplikacija pognana. Vsaka aktivnost lahko zažene nove aktivnosti, ki nato opravljajo svoje naloge. Vsakič, ko je zagnana nova aktivnost, se trenutna aktivnost ustavi in doda na sklad, nova pa je na vrhu in v ospredju. Predhodna aktivnost vedno ostane v skladu in ne pride v ospredje, dokler se aktivnost na vrhu sklada ne izvede do konca. Sklad uporablja mehanizem »prvi v vrsto, prvi iz nje« (angl. first in, first out).

Ko je aktivnost zaustavljena, je o spremembi stanja OS obveščen preko povratnih metod, ki jih ima življenjski krog aktivnosti. Obstaja več različnih stanj aktivnosti. Takrat se kličejo različne metode, npr.: aktivnost ustvarjena (onCreate), ko se prične izvajati (onStart), ko je uničen (onDestroy), ko je ustavljen (onStop), postavljen na čakanje (onPause) ali ponovno zagnan (onResume). Sistem nam omogoča, da lahko ob vsakem stanju povemo, kaj naj se izvrši, ko se aktivnost postavi v določeno stanje. Na sliki 2.2 vidimo vsa zgoraj opisana stanja. Vsaka aktivnost je lahko v štirih stanjih [1]:

- aktivnost je v ospredju, na vrhu sklada in je aktivna;

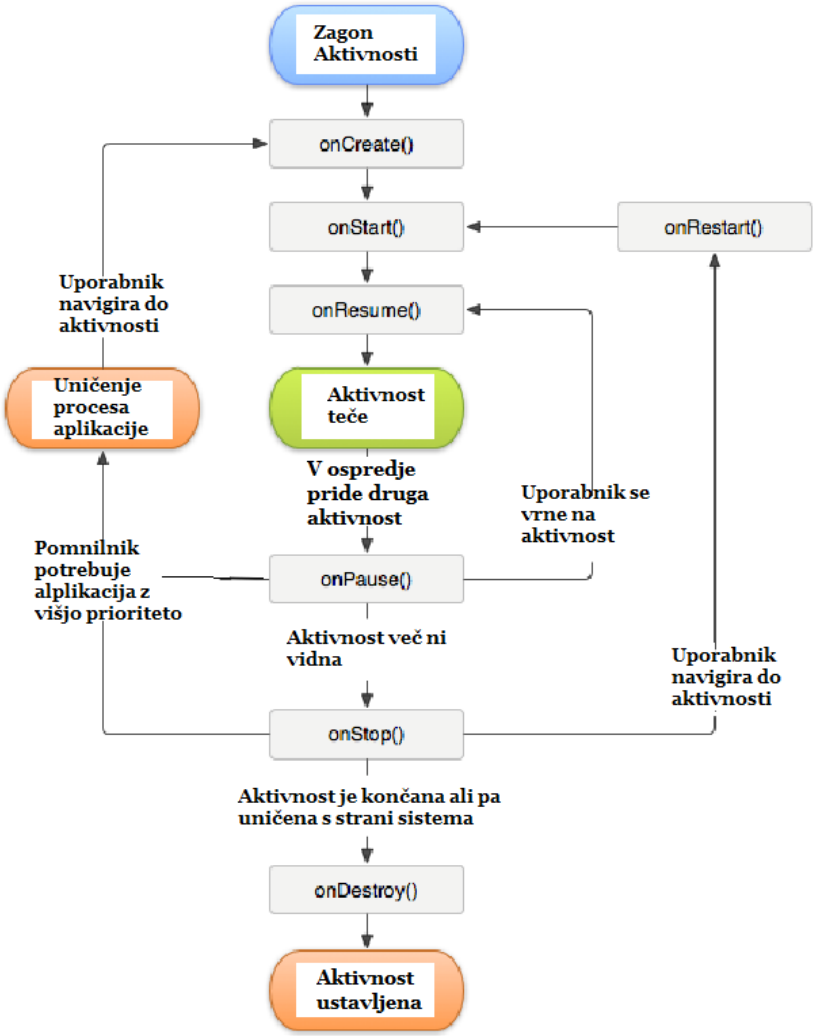
- aktivnost lahko izgubi fokus, a je še vedno vidna, vendar je pred njo nova aktivnost. V tem primeru je aktivnost v stanju premora (angl. pause). Takšna aktivnost je popolnoma živa, ohranjen ima celoten status, vse njegove člane ter

spremenljivke, vendar je lahko takšna aktivnost zaustavljena v primeru pomanjkanja pomnilnika;

- aktivnost je v celoti zamenjana z drugo. V tem primeru bo ustavljena. Ohranilo se bo celotno stanje in vsi člani, vendar ni več vidna uporabniku. V tem primeru je velika verjetnost, da bo takšna aktivnost uničena, ko bo pomnilnik potreboval dodaten prostor za druga opravila;

- če je aktivnost ustavljena ali na premoru, jo lahko sistem izrine iz spomina tako, da od nje zahteva da se dokonča ali pa jo lahko uniči. Ko želi biti ponovno prikazana uporabniku, mora biti ponovno zagnana in povrnjena v preteklo stanje.

Slika 2.2: Android življenjski krog [1]



2.4. Gingerbread

Gingerbread je verzija Android OS od 2.3 – 2.3.7 vendar je med verzijami 2.3.2 in 2.3.3 prišlo do večjih izboljšav. Slednja 2.3.3 – 2.3.7 je trenutno najbolj razširjena verzija

Android OS na pametnih telefonih. Zaseda več kot tretjino (36 %) vseh nameščenih Android OS. Predstavili bomo novosti, ki jih je ta verzija prinesla.

UV je v veliko pogledih spremenjen, kar naredi sistem lažje učljiv, hitreje uporaben in energijsko varčen. Spremembe v meniju in nastavitvah omogočajo uporabniku lažjo in boljšo navigacijo ter kontrolo lastnosti sistema in naprave.

Navidezna tipkovnica v OS Gingerbread je spremenjena in optimizirana za lažje in hitrejše pisanje besedil. Vse tipke so ponovno oblikovane, kar jih naredi bolj elegantne, omogočajo lažje pritiskanje in boljšo vidljivost.

Tipkovnici je dodana možnost popravljanja s pomočjo izbire iz slovarja. Uporabnik lahko izbere že napisano besedo, slovar pa mu poda nasvete za zamenjavo besede. Spremeni lahko tudi na glasovni način vnosa za zamenjavo označenega.

Nova večdotičnostna tipka (angl. multitouch) uporabniku omogoča hitro menjavo med črkami, številkami ter simboli, brez da bi morali to sami spreminjati v nastavitvah. Za nekatere tipke lahko uporabnik z daljšim pritiskom izbira med različnimi variacijami med simboli (npr. Tipka c ima čć).

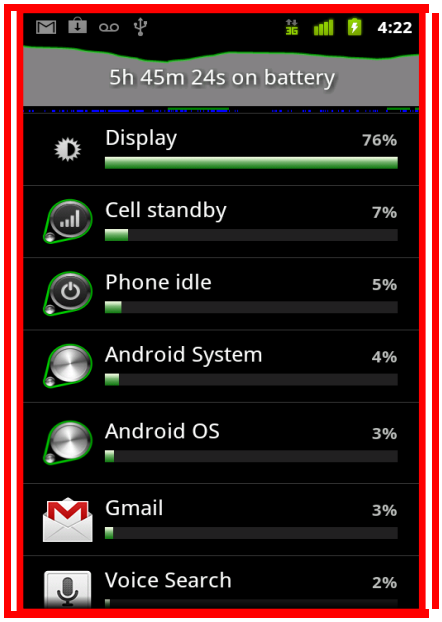
Android OS prevzame večjo, bolj aktivno vlogo pri upravljanju aplikacij, ki zadržujejo OS v aktivnem (angl. awake) stanju preveč časa ali tiste, ki porabljajo centralno procesno enoto, ko tečejo v ozadju. S tem nadzorom sistem sam zapira aplikacije, če je to primerno, in tako izboljša storitev ter podaljša življenjsko dobo baterije.

Sistem omogoča uporabniku večji pregled nad sistemskimi komponentami in aktivnimi aplikacijami. Tako lahko v nastavitvah jasno vidijo kakšno porabo ima katera aplikacija in koliko je porabljenega prostora. To je prikazano na sliki 2.3.

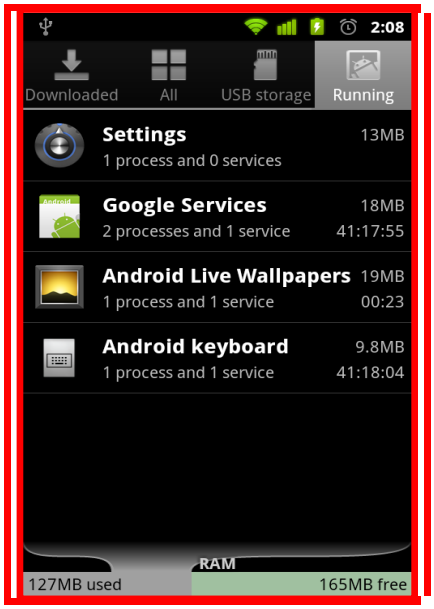
Bližnjica do upravitelja procesov se pojavi v meniju nastavitev in na domačem zaslonu, kar omogoča boljši dostop in s tem boljši nadzor pomnilnika. Upravitelj prikaže vse delujoče procese, nato pa lahko uporabnik izbere proces za še podrobnejši pogled. Na sliki 2.4 lahko spremljamo porabo pomnilnika.

Na novo je zasnovan upravitelj prenosov in omogoča lahek in hiter dostop do vseh datotek, ki so prenesene s spleta ali elektronskega sporočila [5].

Slika 2.3: Android poraba baterije [1]



Slika 2.4: Android poraba pomnilnika [1]



3 WINDOWS COMMUNICATION FUNDATION

WCF (angl. Windows communication fundation) je platforma za razvoj servisov. Zagotavlja servisno-orientirano arhitekturo (angl. Service-oriented Architecture, SOA), ki je uporabljena za izgradnjo porazdeljenih sistemov[3].

WCF zagotavlja model, s katerim lahko implementiramo servis, združljiv z veliko splošno sprejetimi standardi. Vključno s SOAP (angl. Simple Object Access Protocol), XML (angl. Extensible Markup Language) in Json (angl. JavaScript Object Notation).

Dodatno WCF podpira veliko tehnologij Microsoft za gradnjo komponent, kot so Enterprise Services in MSMQ (angl. Microsoft Message Queue). Združljivost s standardi nam omogoča izgradnjo rešitev, neodvisnih od sistema na katerem tečejo in njihovih protokolov za komunikacijo. Z uporabo WCF so te tehnologije združene in je do njih možno dostopati z vsemi vrstami aplikacij.

Skoraj nemogoče je ločiti programersko strukturo aplikacije ali servisa od infrastrukture za komunikacijo, a WCF nam pomaga priti zelo blizu temu. Z njegovo uporabo zagotavljamo združljivost s preteklimi tehnologijami. Omogočena je tudi komunikacija s strežnikom v oblakih (Microsoft Azure) [11].

3.1 Komunikacija z odjemalcem

WCF zagotavlja okolje in nabor programskih knjižnic (angl. Application Programming Interface, API) za izgradnjo sistemov, ki pošiljajo sporočila med servisi in odjemalci. Ista infrastruktura in API so uporabljeni za izdelavo aplikacij, ki komunicirajo z drugimi aplikacijami na istem računalniškem sistemu ali na sistemu drugega podjetja.

3.1.1 Pošiljanje sporočil in vstopne točke

WFC je zasnovan na podlagi komunikacije s sporočili in vsem, kar je lahko predstavljeno kot sporočilo (npr. HTTP (angl. HyperText Transfer Protocol) zahteva).

To je lahko predstavljeno v enotni smeri v programskem modelu, kar omogoča enoten API čez različne mehanizme prenosa.

WCF servis odgovarja na zahteve, ki jih pošilja odjemalec. Servis je lahko ob enem tudi odjemalec. Sporočila so poslana med vstopnimi točkami (angl. EndPoints). Vstopna točka je mesto, kjer so sporočila lahko poslana, sprejeta ali oboje hkrati. Vstopne točke določajo pravila, ki so potrebna za izmenjavo sporočil. Servis ima eno ali več vstopnih točk, odjemalec pa ustvari vstopno točko, ki je združljiva z eno izmed vstopnih točk servisa.

Vstopna točka v standardizirani obliki opisuje, kam mora biti sporočilo poslati, kako naj bo sporočilo poslano in kako naj sporočilo zgleda. Servis lahko to prikaže kot metapodatke, ki jih razvijalec prebere, da lahko ustvari primernega WCF odjemalca [12].

3.1.2 Komunikacijski protokol

Zahtevan element komunikacije je transparenten protokol. Sporočila so lahko poslana preko lokalnih omrežij ali protokolov, kot sta HTTP in TCP (angl. Transmission Control Protocol). Vključeni so tudi protokoli, ki podpirajo komunikacijo z MSMQ in vozlišči v mreži omrežij (angl. Networking mash). Več prenosnih mehanizmov je lahko dodanih z uporabo vgrajenih WCF razširitev.

Še en element ki je potreben za komunikacijo je kodiranje, ki pove kako je sporočilo formirano. WCF zagotavlja naslednja kodiranja [12]:

- besedilno kodirano,
- message Transmission Optimization Mechanism
- binarno kodirano.

3.1.3 Json format

Json je odprt besedilni format za izmenjavo objektov med platformami. Zasnovan je tako, da je berljiv brez večjih težav. Ustvarjen je bil za JavaScript, skriptni jezik, za predstavitev enostavnih podatkovnih tipov in asociativnih polj. Kljub njegovi asociaciji z jezikom JavaScript, ki je prisoten tudi v njegovem imenu, je jezikovno neodvisen, tako da je podprt v večini sodobnih programskih jezikov.

Njegov uradni tip večnamenske razširitvene internetne pošte (angl. Multipurpose Internet Mail Extensions, MIME) je »application/json«, datoteke pa imajo končnico ».json«.

Json se veliko uporablja za serializacijo in pošiljanje strukturiranih podatkov preko omrežij. Uporabljen je kot alternativa XML, in sicer za komunikacijo med strežnikom in odjemalcem.

Osnovni podatkovni tipi, ki jih format podpira so, števila, nizi, boolean, polja in objekti, ki so v Json formatu. Json format sprejema tudi vrednost null [8].

Primer json:

```
{"ime":"Bojan", "priimek":"Orter", "status":"Študent"}
```

Primer json-a s poljem:

```
{"Živali":["pes", "mačka", "papiga"]}
```

Primer json-a z vgnezdenim objektom:

```
{"ime":"Bojan", "priimek":"Orter", "naslov":{"kraj":"Radlje
ob Dravi", "ulica":"Vrtna ulica"}}
```

4 GOOGLE CLOUD MESSAGING

GCM (angl. Google Cloud Messaging) za Android je brezplačni servis, ki pomaga razvijalcu pošiljati podatke iz strežnika, do njihovih Android aplikacij in obratno. To so lahko lahka sporočila, ki aplikacijo opozorijo, da obstajajo novi podatki (npr. opozorilo, da jih čaka nova elektronska pošta). Sporočila so lahko velika do 4Kb. GCM skrbi za vse funkcionalnosti, kot so delitev sporočil v vrsto in dostavljanje teh do ciljnih aplikacij [6].

4.1 Glavne lastnosti GCM

GCM dovoli zunanjim aplikacijskim strežnikom (angl., 3rd party server) pošiljanje sporoči Android aplikacijam.

Z uporabo GCM povezovalnega strežnika (angl. Connection Server), lahko strežnik sprejema sporočila z uporabnikove naprave.

Android aplikacija ne rabi biti v ospredju, da sprejme sporočilo. OS bo zbudil aplikacijo, ko sporočilo prispe.

GCM ne prinaša vgrajenega UV ali drugega opravitelja za obdelavo sporočila. GCM posreduje čisto sporočilo, ki je bilo poslano Android aplikaciji, ta pa sama poskrbi, kaj bo s sporočilom naredila. Aplikacija lahko, na primer pošlje sporočilo ali pa v ozadju posodobi podatke.

Minimalna verzija Android OS, ki je podprta s strani GCM, je 2.2, prav tako pa mora biti na napravi prisotna trgovina GooglePlay [5].

4.2 Arhitektura GCM

Implementacija GCM vsebuje Google strežnik, zunanjo strežniško aplikacijo, ki komunicira z Google strežnikom in aplikacijo, ki omogoča komuniciranje s servisom GCM. Arhitektura je prikazana na spodnji sliki 4.1.

Slika 4.1:Arhitektura GCM [6]

Interakcija strežnik, GCM strežnik in odjemalec:

Google povezovalni strežnik GCM (angl., GCM Connection Server) sprejema sporočila iz tujega strežnika in posreduje ta sporočila do Android aplikacij, ki jih imenujemo odjemalčeve aplikacije (angl. Client app).

Tuj strežnik je komponenta, ki jo implementirate za delovanje z izbranim GCM strežnikom. Strežniška aplikacija pošilja sporočila GCM strežniku. Le-tasporočila sprejme in shrani, nato pa jih posreduje napravi, ko se ta poveže v splet.

Odjemalčeva aplikacija ima omogočen GCM in teče na napravi uporabnika. Za sprejemanje sporočil, ki jih posreduje GCM strežnik, se mora naprava najprej registrirati na servisu, s čemer dobi registracijsko številko (angl., registration ID) [6].

5 PRAKTIČNI DEL

5.1 Uporabljena orodja

Microsoft Visual Studio 2010

Orodje Microsoft Visual Studio 2010 [9] smo uporabili za strežniško stran aplikacije, ter za izdelavo WCF servisa in samo stoječega programa (angl. Self hosting application), ki deluje kot strežnik za WCF-servis.

Microsoft Visual Studio 2010 je splošno namensko orodje za razvoj aplikacij. Omogoča nam razvoj v programskih jezikih: c#, c++, Visual basic, itd. V njem lahko razvijamo Windows form aplikacije, .Net aplikacije, WCF servis, konzolne aplikacije in mobilne aplikacije. Microsoft SQL Server 2008 express [10] smo uporabili za povezovanje s podatkovno bazo.

Eclipse JUNO

Eclipse JUNO [4] smo uporabljali za razvoj odjemalca, Android aplikacije. Eclipse je razvojno okolje namenjeno predvsem za razvoj java aplikacij. V njem lahko razvijamo tako konzolne aplikacije, grafične aplikacije in openGL aplikacije. Z dodatnim vtičnikom lahko razvijamo tudi Android aplikacije.

5.2 Odjemalec

Kot smo že omenili, je odjemalec Android aplikacija, razvita v orodju Eclipse. Aplikacija prikazuje aktualne ponudbe, z njo lahko iščemo oglase s ključno besedo in se naročimo na obvestila.

Glavni del aplikacije se odvija v razredu imenovanem `StartUpActivity`. V aktivnosti so prikazani kratki opisi vseh ponudb. Prav tako ima vnosno polje za iskanje s ključnimi besedami. Iz njega lahko preidemo v druge aktivnosti, aktivnost nastavitve (razred `SettingsActivity`) in aktivnosti podrobnega prikaza oglasa (razred `DetailsActivity`).

V nastavitvah lahko izbiramo med vnaprej definiranimi ključnimi besedami in dodajamo lastne.

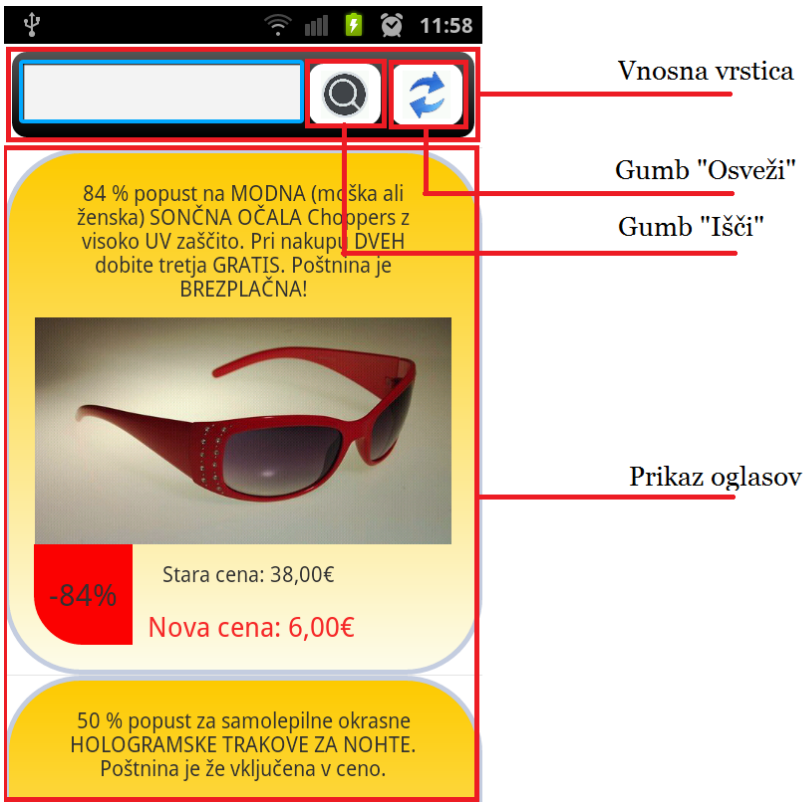
V podrobnem opisu imamo predstavljen celotni izdelek z vsemi podrobnostmi.

Sestavni del aplikacije je tudi razred `GcmBroadcastReceiver`, kateri skrbi za sprejemanje sporočil s strani GCM servisa. Ta sporočila posreduje naprej v obliki obvestil v statusni vrstici naprave.

5.2.1 Aktivnost za »Prikaz vseh ponudb«

Ta aktivnost (imenovana `StartUpActivity`) je osrednji del Android aplikacije. Iz nje dostopamo do vseh ostalih aktivnosti. Sestavljena je iz dveh delov: prikaza oglasov in brskalne vrstice z gumboma »Išči« in »Osveži« (sinhroniziraj). Izgled vidimo na sliki 5.1.

Slika 5.1: Izgled prikaz vseh oglasov



Ob izdelavi aktivnosti se v metodi `onCreate()` najprej preveri, če je naprava že registrirana za GCM strežnik (Slika 5.2). Če ni, se odpre aktivnost nastavitev v katerem nadaljujemo z registracijo. V primeru, da smo že registrirani, se najprej naloži seznam oglasov iz pomnilniške kartice (angl. Secure Digital, SD). Ta seznam predstavlja oglase od takrat, ko smo nazadnje sinhronizirali podatke s strežnikom, zato so lahko zapadli. Za ponovno sinhronizacijo moramo pritisniti gumb »Osveži«. V vmesniku imamo še gumb »Išči«, ob pritisku nanj se seznam oglasov osveži in tako dobimo samo oglase, ki vsebujejo ključno besedo, vpisano v vnosno polje.

Slika 5.2 Koda StartUpActivity

```
public class StartUpActivity extends Activity {

    private List<OglasKratek> vsiOglasi;//predstavlje oglase ki bodo predstavljeni
    private WcfZbitaCena service;//Spremenljivka za povezovanje s strežnikom
    private ProgressDialog bar;//widget za prikaz da se nekaj izvaja
    private Context context;//context activitya
    private ListView lstViewOglasi;//ListView v katerem so prikazani oglasi
    private EditText etxtSearch;//vnosno polje, ki predstavlja brskalno vrstico


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.start_up_activity);
        context = this;
        etxtSearch = (EditText) findViewById(R.id.etxtKeyWordStartUp);
        SDCardChacker();
        if(getIntent().getExtras() != null)
        {
            String oglasID = getIntent().getExtras().getString("oglasID");
            AsyncTaskListItemClick atLoadOglas = new AsyncTaskListItemClick(context, bar, oglasID);
            atLoadOglas.execute(this);
        }

        if(GcmBroadcastReceiver.isGCMRegisterd(this))
        {
            lstViewOglasi = (ListView) this.findViewById(R.id.lstViewOglasi);
            service = new WcfZbitaCena(true);
            AsyncTaskLoadList loadList = new AsyncTaskLoadList(false, false);
            loadList.execute("");
        }
        else
        {
            Intent intentSettingAct = new Intent(context, SettingsActivity.class);
            startActivity(intentSettingAct);
        }
    }
```

Nalaganje in prikaz oglasov se izvrši v razredu `AsyncTaskLoadList`, ki razširja nadrazred `AsyncTask`. Ob kreiranju objekta sprejme štiri parametre (`ctx`, `lstViewOglasi`, `isReadingFromURL in isRefresh`). Implementirane ima tri metode:

- `onPreExecute()`

  V tem delu ustvarimo in prikažemo nalagalni dialog (Slika 5.3).

- `doInBackground()`

  Prestavlja glavni del razreda.V njem se izvede branje iz SD kartice ali povezava s strežnikom in ustvarjanje `OglasKratekArrayAdapter`, ki popularizira seznam (`ListView`), v katerem so prikazani oglasi. Metoda kot rezultat vrne ta adapter (Slika 5.4).

- `onPostExecute()`

Izvrši se, ko je opravljeno delo v `doInBackGround()`.V njej seznam oglasov (`ListView`), ki prevzamejo izgled posameznih oglasov, kot ga je določil `ArrayAdapter`. Skrije se dialog napredka (Slika 5.3).

Slika 5.3 Koda AyncTaskLoadList pre/postExecute

```
@Override
protected void onPreExecute()
{
    bar = new ProgressDialog(context);
    bar.setTitle("Nalaganje");
    if(isReadingFromURL)
        bar.setMessage("Pridobivanje oglasov preko servisa...");
    else
        bar.setMessage("Pridobivanje oglasov iz SD kartice...");
    bar.setIndeterminate(true);
    bar.show();
}
```

```
@Override
protected void onPostExecute( OglasKratekArrayAdapter result )
{
    super.onPostExecute(result);
    if(result != null)
    {
        lstViewOglasi.setAdapter(result);
    }
    bar.dismiss();
}
```

Slika 5.4: koda AsyncLoadList doInBackGround

```
private class AsyncTaskLoadList extends AsyncTask<String, String, OglasKratekArrayAdapter> {
        private boolean isReadingFromURL, isRefresh;
        private Context ctx;
        private ListView lstViewOglasi;
        public AsyncTaskLoadList(Context ctx, ListView lstViewOglasi, boolean isReadingFromURL, boolean isRefresh) {
            this.ctx = ctx
            this.lstViewOglasi = lstViewOglasi;
            this.isReadingFromURL = isReadingFromURL;
            this.isRefresh = isRefresh;
        }
        @Override
        protected OglasKratekArrayAdapter doInBackground(String... params) {
            OglasKratekArrayAdapter lstViewAdapter = null;
            if(isReadingFromURL)
            {
                if(isRefresh)
                {
                    try {
                        vsiOglasi = service.getVsiAktivniOglasi();
                    } catch (IOException e) {e.printStackTrace();}
                }
                else
                {
                    try {
                        vsiOglasi = service.getOglasiWithKeyWord(etxtSearch.getText().toString().replace(' ', '_'));;
                    } catch (MalformedURLException e) {e.printStackTrace();}
                }
                if(vsiOglasi != null)
                    lstViewAdapter = new OglasKratekArrayAdapter(context, vsiOglasi);
            }
            else
            {
                try{
                    vsiOglasi = service.getVsiAkrivniOglasiFromSDCard();
                }catch (IOException e){e.printStackTrace();}

                if(vsiOglasi != null)
                    lstViewAdapter = new OglasKratekArrayAdapter(context, vsiOglasi);
            }
            return lstViewAdapter;
        }
```

Kot smo omenili prej, vrne `doInBackGround()` rezultat tipa `OglasKratekArrayAdapter` (Slika 5.5). To je razred, ki razširja nadrazred `ArrayAdapter`. Njegovo delo je, da vrne izgled posameznega oglasa v seznamu in vnese podatke oglasa na prava mesta za prikaz. To stori z metodo `getView()`, ki vrne

izgled tipa `View`, ki ga nato seznam (`ListView`) prikaže. V konstruktorju dobi argument context in oglasi.

Slika 5.5: Koda OglasKratekAdapter

```
1  public class OglasKratekArrayAdapter extends ArrayAdapter<OglasKratek> {
2      private Context context;//kontekst aplikacije
3      private List<OglasKratek> oglasi;//seznam oglasov nad katerimi bo deloval
4
5      public OglasKratekArrayAdapter(Context context, List<OglasKratek> oglasi) {
6          super(context, R.layout.oglas_row_lo, oglasi);
7          this.context = context;
8          this.oglasi = oglasi;
9      }
10
11     public View getView(int position, View convertView, ViewGroup parent) {
43 }
```

Metoda `getView()` najprej pridobi gradnike UV (Slika 5.6). Nato gradnikom priredi podatke, s katerimi bodo napolnjeni. Gre za nekaj besedilnih polj in sliko. V vrsticah 11 do 24 vidimo pridobivanje slike izdelka. Najprej preverimo, če slika že obstaja na SD kartici, če je tam ni, jo prenesemo s spleta. Na koncu preverimo, da ni pri obeh postopkih pridobivanja slike prišlo do napake (slika ni bila pravilno pridobljena). V tem primeru se namesto slike oglasa prikaže privzeta slika. Besedilnim poljem le nastavimo njim namenjeno besedilo.

Slika 5.6 : Koda OglasKratekArrayAdapter getView

```
1
2  public View getView(int position, View convertView, ViewGroup parent) {
3      LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
4      View rowView = inflater.inflate(R.layout.oglas_row_lo, parent, false);
5
6      TextView txtNaslov = (TextView) rowView.findViewById(R.id.txtOglasRowTitel);
7      ImageView imgSlika = (ImageView) rowView.findViewById(R.id.imgOglasRowSlika);
8      TextView txtPopust = (TextView) rowView.findViewById(R.id.txtOglasRowPopustProc);
9      TextView txtNovaCena = (TextView) rowView.findViewById(R.id.txtOglasRowNovaCena);
10     TextView txtStaraCena = (TextView) rowView.findViewById(R.id.txtOglasRowStaraCena);
11     Bitmap image = null;
12     if(ImageHelper.isImageOnSDCard(String.valueOf(oglasi.get(position).getId())))
13         image = ImageHelper.getBitmapFromSDCard(String.valueOf(oglasi.get(position).getId()));
14     else
15         image = ImageHelper.getBitmapFromURL(oglasi.get(position).getImageUrl(), String.valueOf(oglasi.get(position).getId()));
16
17     if(image != null)
18     {
19         imgSlika.setImageBitmap(image);
20     }
21     else
22     {
23         imgSlika.setImageDrawable(context.getResources().getDrawable(R.drawable.def));
24     }
25     txtNaslov.setText(oglasi.get(position).getTitle());
26     txtPopust.setText("-" + oglasi.get(position).getDiscount() + "%");
27
28     Formatter format = new Formatter();
29     txtStaraCena.setText("Stara cena: " + format.format("%.2f", oglasi.get(position).getOriginalValue()) + "€");
30     format = new Formatter();
31     txtNovaCena.setText("Nova cena: " + format.format("%.2f", oglasi.get(position).getDiscountedValue()) + "€");
32     format.close();
33
34     return rowView;
35 }
```

V omenjeni metodi `getView()` sliko pridobivamo s pomočjo pomožnega razreda `ImageHelper` (Slika 5.7). Razred ima tri statične metode. Prva pridobi sliko iz SD kartice, druga iz URL naslova, tretja pa samo preveri, če določena slika že obstaja na SD kartici.

Slika 5.7: Koda ImageHelper

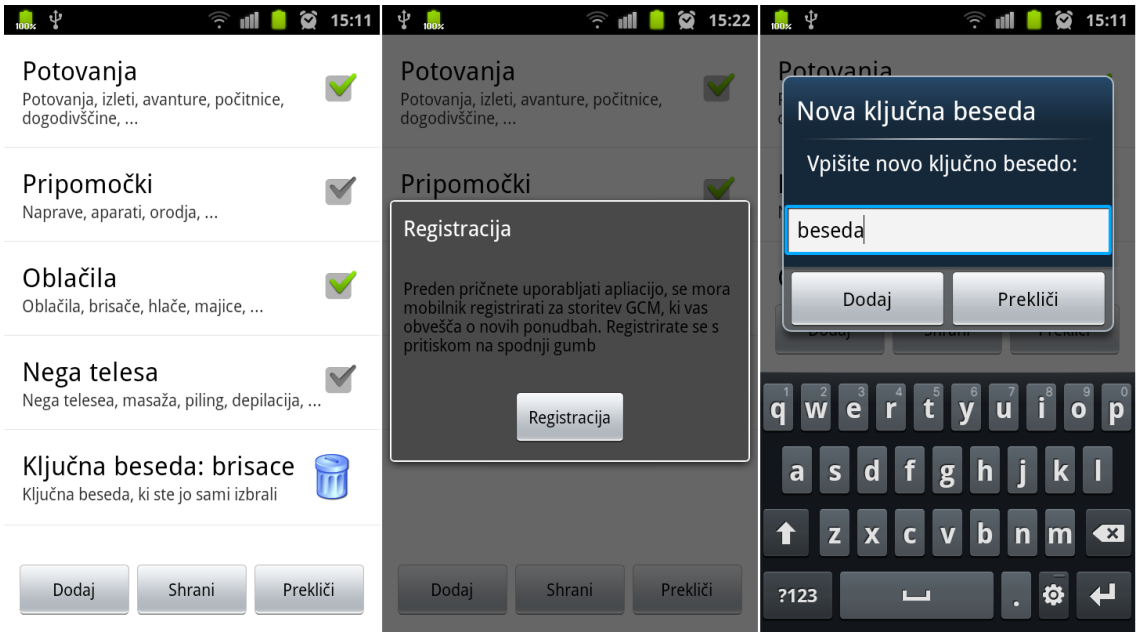
```
1  public class ImageHelper {
2      public static Bitmap getBitmapFromURL(String uRL, String imageID) {
19
20     public static Bitmap getBitmapFromSDCard(String imageID){
33
34     public static boolean isImageOnSDCard(String imageID){
49 }
```

5.2.2 Aktivnost »Nastavitev«

V to aktivnost lahko pridemo na dva načina. Prvi način je, da smo odprli aplikacijo in še nismo registrirali naše naprave za GCM servisu. To se stori v razredu `StartUpActivity` (Slika 5.5, vrstice 33 do 36). Drugi način poteka tako, da v glavni aktivnosti pritisnemo na gumb »meni« in pritisnemo »Settings«.

V tej aktivnosti opravimo izbiro naših ključnih besed (Slika 5.9). V aktivnosti se izvede tudi GCM registracija. Izgled aktivnosti je prikazan na sliki 5.8.

Slika 5.8: Izgled SettingsActivity



Če še nismo registrirani za storitev GCM, se nam prikaže opozorilno okno. S pritiskom na gumb »Registracija«, se registracija izvede. Registracija naprave se izvede samo

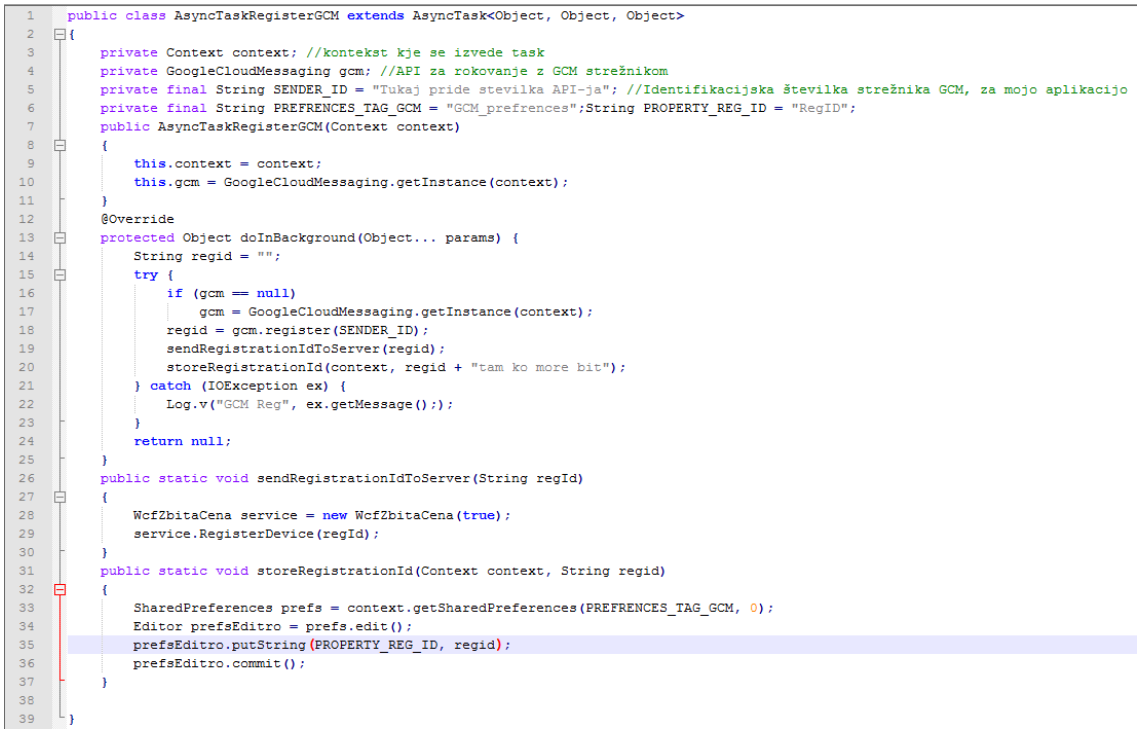
enkrat. Pri ostalih dostopih do nastavitev se nam prikaže seznam (`ListView`), ki na vrhu prikaže že definirane, privzete ključne besede. Spodaj pa ključne besede, ki smo jih dodali sami.

Slika 5.9: Koda SettingsActivity

```
public class SettingsActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.settings_activity);
        if(!GcmBroadcastReceiver.isGCMRegisterd(context))
        {
            final Dialog dialog = new Dialog(context);
            dialog.setContentView(R.layout.reg_dialog);
            dialog.setTitle("Registracija");
            Button dialogButton = (Button) dialog.findViewById(R.id.dialogBtnRegistracija);
            dialogButton.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                    if(!InternetChacker.isNetworkAvailable(context))
                        Toast.makeText(context, "Ni internetne povezave, vklopitejo in poskusite ponovno.", Toast.LENGTH_LONG).show();
                    else
                    {
                        AsyncTaskRegisterGCM registerGCM = new AsyncTaskRegisterGCM(context);
                        registerGCM.execute(null, null, null);
                        dialog.dismiss();
                    }
                }
            });
            dialog.show();
        }
        //nalaganje nastavitev
        AsyncTaskLoadSettings loadSettings = new AsyncTaskLoadSettings();
        loadSettings.execute(null, null, null);
    }
```

Registriramo se s pomočjo knjižnice, ki je namenjena storitvi GCM in ima tudi to ime (`GoogleCloudMessaging`). Pri klicu metoda `gcm.register()` je parameter `SENDER_ID` številka, ki jo dobimo v Google API konzoli [7]. Rezultat metode je identifikacijska številka naprave, ki jo nato shranimo v »`SharedPrefrences`« in na spletni strežnik. Registracija se izvrši v razredu `AsyncTaskRegisterGCM` (Slika 5.10), ki je podrazred že prej omenjenega razreda `AsyncTask`. Shranjevanje na strežnik se izvede s pomočjo razreda `WcfZbitaCena` katerega bomo predstavili kasneje.

Slika 5.10: Koda AsyncTaskRegisterGCM

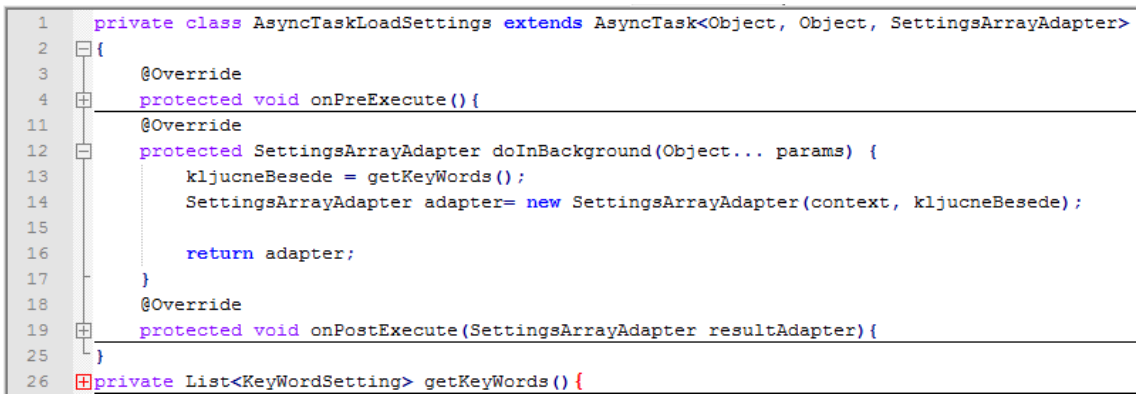


```
public class AsyncTaskRegisterGCM extends AsyncTask<Object, Object, Object>
{
    private Context context; //kontekst kje se izvede task
    private GoogleCloudMessaging gcm; //API za rokovanje z GCM strežnikom
    private final String SENDER_ID = "Tukaj pride stevilka API-ja"; //Identifikacijska številka strežnika GCM, za mojo aplikacijo
    private final String PREFRENCES_TAG_GCM = "GCM_prefrences";String PROPERTY_REG_ID = "RegID";
    public AsyncTaskRegisterGCM(Context context)
    {
        this.context = context;
        this.gcm = GoogleCloudMessaging.getInstance(context);
    }
    @Override
    protected Object doInBackground(Object... params) {
        String regid = "";
        try {
            if (gcm == null)
                gcm = GoogleCloudMessaging.getInstance(context);
            regid = gcm.register(SENDER_ID);
            sendRegistrationIdToServer(regid);
            storeRegistrationId(context, regid + "tam ko more bit");
        } catch (IOException ex) {
            Log.v("GCM Reg", ex.getMessage(););
        }
        return null;
    }
    public static void sendRegistrationIdToServer(String regId)
    {
        WcfZbitaCena service = new WcfZbitaCena(true);
        service.RegisterDevice(regId);
    }
    public static void storeRegistrationId(Context context, String regid)
    {
        SharedPreferences prefs = context.getSharedPreferences(PREFRENCES_TAG_GCM, 0);
        Editor prefsEditro = prefs.edit();
        prefsEditro.putString(PROPERTY_REG_ID, regid);
        prefsEditro.commit();
    }

}
```

Nalaganje ključnih besed se izvede v razredu `AsyncTaskLoadSettings`. Metodi `onPreExecute()` in `onPostExecute()` naredita iste stvari, kot pri razredu `AsyncTaskLoadList` (Slika 5.11) ter v razredu `StartUpActivity` (Slika 5.2). Metoda `doInBackGround()` pridobi seznam ključnih besed iz »`SharedPrefrences`«, ta seznam pa vrne metoda `getKeyWords()`, nato se ustvari nov adapter, kise vrne.

Slika 5.11: Koda AsyncTaskLoadSettings



```
private class AsyncTaskLoadSettings extends AsyncTask<Object, Object, SettingsArrayAdapter>
{
    @Override
    protected void onPreExecute(){
    @Override
    protected SettingsArrayAdapter doInBackground(Object... params) {
        kljucneBesede = getKeyWords();
        SettingsArrayAdapter adapter= new SettingsArrayAdapter(context, kljucneBesede);

        return adapter;
    }
    @Override
    protected void onPostExecute(SettingsArrayAdapter resultAdapter){
}
private List<KeyWordSetting> getKeyWords(){
```

`SettingsArrayAdapter` razširja nad razred `ArrayAdapter`. Njegova glavna metoda je `getView()`, ki vrne izgled posameznega objekta v seznamu (`ListView`).

Ta postopek je bil opisan že prej, v tem razredu pa je spremenjen samo tip seznama objektov. Ti objekti so tipa `KeyWordSetting` (Slika 5.12).

`KeyWordSetting` je pomožen razred, ki ima le štiri privatne spremenljivke, njihove set in get metode ter konstruktor, ki sprejme vse štiri lastnosti. Prva spremenljivka predstavlja ključno besedo, druga pove, ali gre za privzeto skupino besed, ki so že vnaprej določene. Tretja je pomembna, če je beseda privzeta in predstavlja vrednost, ko je ta izbrana za prejemanje obvestil. Četrta pa predstavlja le to, kako se imenuje ključna beseda v »`SharedPrefrences`« aplikaciji.

Slika 5.12: Koda KeyWordSettings

```
1   public class KeyWordSetting {
2       private String keyWord;
3       private boolean defaultKeyWord;
4       private boolean chkValue;
5       private String SharedPrefrencesPropertyName;
6       //{ razred ima ze štiri privatene spremenljivke le set in get metode, ter konstrutor  z vsemi lastnostmi.
51  }
```

Na dnu aktivnosti nastavitve imamo še tri gumbe: »Dodaj«, »Shrani« in »Prekliči«. Z gumbom »Dodaj« se nam odpre dialog za dodajanje nove ključne besede, v katerega vpišemo novo ključno besedo. Potrdimo jo s pritiskom na gumb »Dodaj«. Trajno shranjena je šele ob pritisku na gumb »Shrani«. Ob pritisku na gumb »Shrani« se naše trenutne nastavitve shranijo v »SharedPrefrences«, naše aplikacije in na naš strežnik. Shranjevanje na strežnik se izvede s pomočjo razreda `WcfZbitaCena`. Gumb »Prekliči« zaključi trenutno aktivnost, in sicer brez sprememb.

5.2.3 Aktivnost za prikaz podrobnosti oglasa

V tej aktivnosti `DetailsActivity` je prikazan oglas z vsemi podrobnostmi oglasa (Slika 5.14).

Aktivnost ob klicu metode `onCreate()` prebere dodatne lastnosti objekta Intent, s katerim je bil ustvarjen. V teh je shranjen niz, ki predstavlja Json notacijo objekta OglasCeloten. Ta niz, s pomočjo knjižnice Gson za serializacijo formata json, spremenimo v objekt tipa `OglasCeloten`. Nato pa v `WebView` nastavimo vsebino tega oglasa v obliki HTML. Izgled na sliki 5.13.

Slika 5.13: Izgled aktivnosti DetailsActivity

Za izgled samega oglasa v `WebView` skrbi CSS (angl. Cascading Style Sheets), ki je shranjen kot vir (angl. resource) v aplikaciji, v mapi assets. Ta se skupaj z HTML (angl. HyperText Markup Language) vsebino naloži v vrstici 18. Slika 5.14.

Slika 5.14: Koda DetailsActivity

```
1  public class DetailsActivity extends Activity {
2      protected void onCreate(Bundle savedInstanceState) {
3          super.onCreate(savedInstanceState);
4          this.requestWindowFeature(Window.FEATURE_NO_TITLE);
5          setContentView(R.layout.details_activity);
6
7          Intent intent = getIntent();
8          if(intent.getExtras() == null)
9              this.finish();
10         String jsonString = intent.getStringExtra("Oglas");
11         Gson manager = new Gson();
12         OglasCeloten oglas = manager.fromJson(jsonString, OglasCeloten.class);
13
14         oglas.cleanDescription();
15         String html = oglas.getOglasHtml();
16
17         WebView webOglas = (WebView) findViewById(R.id.webDetailsDescription);
18         webOglas.loadDataWithBaseURL("file:///android_asset/", html, "text/html", "UTF-8", null);
19
20         WebSettings settings = webOglas.getSettings();
21         settings.setDefaultTextEncodingName("utf-8");
22     }
23 }
```

5.2.4 Komunikacija s strežnikom

Obstajata dva različna načina komunikacije. Prvi način je komunikacija z našim strežnikom preko servisa `WcfZbitaCena` (Slika 5.15). Vsa komunikacija te vrste se

odvija v razredu `WcfZbitaCena`. Drugi način pa je komunikacija z GCM strežnikom. Ta pa se izvaja v razredu `GcmBroadcastReciver`.

Najprej bomo predstavili razred `WcfZbitaCena`. Kot smo že omenili, ta razred komunicira s servisom WcfZbitaCena na našem strežniku. Njegove funkcionalnosti so:

- pridobivanje vseh aktivnih oglasov,
- pridobivanje oglasa z določeno ključno besedo,
- pridobitev celotnega oglasa,
- pošiljanje registracijskega ID-ja naprave in
- pošiljanje uporabnikovih nastavitev ter ključnih besed strežniku.

Url naslove, na katerih so dostopne metode servisa WcfZbitaCena, ima razred shranjene v konstantah.

Slika 5.15 Konstante WcfZbitaCena

```
1  public class WcfZbitaCena {
2      private String BaseURL = "http://164.8.122.127:8780/";//"http://192.168.1.64:8780/"; // predstavlja konstanto z predpono za vse funkcije
3      private final String OglasByIdURL = BaseURL + "WcfZbitaCena/Service/OglasById/ID="; // predstavlja url servisa na katerem dobimo oglas z ID-jem
4      private final String OglasiByKeyWordURL = BaseURL + "WcfZbitaCena/Service/OglasiByKeyWord/Search=";// predstavlja url servisa na katerem dobimo oglase z ključno besedo
5      private final String VsiAktivniOglasiURL = BaseURL + "WcfZbitaCena/Service/VsiAktivniOglasi";// predstavlja url servisa na katerem dobimo vse aktivne oglase
6      private final String RegisterDeviceURL = BaseURL + "WcfZbitaCena/Service/RegisterDevice/DeviceID=";// predstavlja url servisa na katerem vnesemo deviceID v PB
7      private final String SaveUserSettingsURL = BaseURL + "WcfZbitaCena/Service/SaveUserSettings";// predstavlja url servisa na katerem vnesemo nastavitve uporabnika v PB
8      public WcfZbitaCena(){}
9      public static boolean NotStringNullOrEmpty(String string){return string != null && !string.isEmpty() && !string.trim().isEmpty();}
```

Ena njegovih glavnih metod, v kateri se zgodi dejanska komunikacija s strežnikom, je `getJsonStringFromURL()` (Slika 5.17). V 14 vrstici na sliki 5.17 se povežemo s servisom na naslovu, ki ga metoda pridobi kot parameter. Nato preverimo, da vsebina ni prazna. Če ni, dobimo odgovor. Odgovor preberemo s pomočjo razreda `StringBuffer` in vrnemo niz.

Izvajanje metod `getCelotenOglas()`, `getVsiAktivniOglasi()` in `getOglasiByKeyWord()` se ne razlikuje veliko (Slika 5.16). Na strežnik pošiljajo le argumente tipa `String`. Vsi najprej pridobijo »jsonString« oz. odgovor strežnika, ki je json notacija objekta. Nato pa pretvorijo ta niz, s pomočjo knjižnice Gson, v željen objekt. Razlika pri metodah je v kakšen objekt je pretvorjen »jsonString«. Pri `getVsiAktivniOglasi()` in `getOglasiByKeyWord()` je ta objekt tipa

seznam, ki vsebuje objekte razreda `OglasKratek` (`List<OglasKratek>`). V primeru `getCelotenOglas()`, pa je vrnjen objekt tipa `OglasCeloten`.

Slika 5.16 Koda WcfZbitaCena getMetode

```
1  public List<OglasKratek> getVsiAktivniOglasi() throws MalformedURLException{
2      String requestURL = VsiAktivniOglasiURL;
3      String jsonString = null;
4
5      jsonString = getJsonStringFromURL(requestURL);
6
7      List<OglasKratek> vsiOglasi = null;
8      if(NotStringNullOrEmpty(jsonString))
9      {
10         Gson jsonConverter = new Gson();
11         Type type = new TypeToken<List<OglasKratek>>(){}.getType();
12         vsiOglasi = jsonConverter.fromJson(jsonString, type);
13     }
14     return vsiOglasi;
15 }
16 public List<OglasKratek> getOglasiWithKeyWord(String keyWord) throws MalformedURLException{
33 public OglasCeloten getOglasById(int id) throws MalformedURLException{
```

Slika 5.17: Koda WcfZbitaCena GetJsonStringFromURL

```
1  private String getJsonStringFromURL(String url)
2  {
3      StringBuffer JsonString;
4      InputStreamReader in = null;
5      URL page = null;
6      try {
7          page = new URL(url);
8      } catch (MalformedURLException e) {
9          e.printStackTrace();
10     }
11     HttpURLConnection conn;
12     try {
13         conn = (HttpURLConnection) page.openConnection();
14         conn.connect();
15         if(conn.getContent() != null)
16         {
17             in = new InputStreamReader((InputStream) conn.getContent());
18         }
19     } catch (IOException e) {
20         e.printStackTrace();
21     }
22     String line;
23     JsonString = new StringBuffer("");
24     if(in != null){
25         try {
26             buff = new BufferedReader(in);
27             do {
28                 line = buff.readLine();
29                 if(line != null)
30                     JsonString.append(line);
31             } while (line != null);
32         } catch (IOException e) {}
33     }
34     return JsonString.toString();
35 }
```

Omenjali smo tipe, ki jih vračajo get metode. To sta tipa `OglasCeloten` in `OglasKratek` (Slika 5.18). Eden predstavlja celoten oglas z vsemi podatki. Drugi pa predstavlja skrčen oglas, ki je predstavljen kot element v seznamu (`ListView`) v `StartUpActivity`.

Slika 5.18. Koda OglasKratek, OglasCeloten

```
1  public class OglasCeloten {
2      private int Id;
3      private String Title;
4      private String Region;
5      private double OriginalValue;
6      private double DiscountedValue;
7      private int Discount;
8      private String ImageUrl;
9      private String Deeplink;
10     private String ExpiresOn;
11     private int Bought;
12     private String Description;
13     /*razred ima še set in get metode za vse zgornje lastnosti
14      *dodatne lastnosti so Deeplink, Boutgh in Description
15      */
16     public void cleanDescription(){
33     public String getOglasHtml() {
58 }
```

```
1  public class OglasKratek {
2      private int Id;
3      private String Title;
4      private String Region;
5      private double OriginalValue;
6      private double DiscountedValue;
7      private int Discount;
8      private String ImageUrl;
9      private String ExpiresOn;
10     //Razred ima še set in get metode za vse lastnosti
11 }
12
13
14
15
16
17
18
```

Ostaneta še metodi `RegisterDevice()` in `SaveUserSettings()`. Odgovor strežnika na obe metodi je tipa boolean in nam pove, ali je shranjevanje uspelo.

`RegisterDevice()`, sprejme parameter `deviceID`, ki predstavlja GCM ID, katerega je naprava pridobila ob registraciji (Slika 5.19). Metoda samo pošlje ta ID servisu.

Slika 5.19: Koda WcfZbitaCena registerDevice

```
1  public void RegisterDevice(String deviceID)
2  {
3      String requestURL = RegisterDeviceURL + deviceID;
4      String response = getJsonStringFromURL(requestURL);
5      if(NotStringNullOrEmpty(response))
6      {
7          Log.v("WCF Register device", "odgovror:" + response);
8      }
9  }
```

Metoda `SaveUserSettings()`, pa kot parameter dobi objekt tipa UserSettings (Slika 5.20). Objekt se najprej pretvori v »jsonString«. Te nastavitve s pomočjo razredov `HttpPost` in `DefaultHttpClient` pošlje na strežnik. To se zgodi v vrstici 16 na sliki 5.20.

Slika 5.20: Koda WcfZbitaCena SaveUserSettings

```
public boolean SaveUserSettings(UserSettings settings)
{
    Gson gson = new Gson();
    String jsonString = gson.toJson(settings);
    jsonString = "{\"settings\":" + jsonString + "}";
    try {
        StringEntity entity = new StringEntity(jsonString,"UTF-8");
        HttpPost request = new HttpPost(SaveUserSettingsURL);
        request.setHeader("Accept", "application/json");
        request.setHeader("Content-type", "application/json");
        entity.setContentEncoding(new BasicHeader(HTTP.CONTENT_TYPE, "application/json"));
        entity.setContentType("application/json");
        request.setEntity(entity);
        DefaultHttpClient httpClient = new DefaultHttpClient();
        HttpConnectionParams.setConnectionTimeout(httpClient.getParams(), 10000);
        HttpResponse response = httpClient.execute(request);
        if(response.getStatusLine() == null)
            return false;
        else
            return true;
    } catch (ClientProtocolException e) {
        return false;
    } catch (IOException e) {
        return false;
    }
}
```

`UserSettings` je razred, ki vsebuje lastnosti `deviceID` in `kljucneBesede`. Prva je za identifikacijo naprave, druga pa vsebuje ključne besede na katere je uporabnik naročen.

Drugi način komunikacije s strežnikom je komunikacija z GCM strežnikom. Vse v povezavi s to komunikacijo se odvija v razredu `GcmBroadcastReciver`, ki je podrazred razreda `BroadcastReciver`.

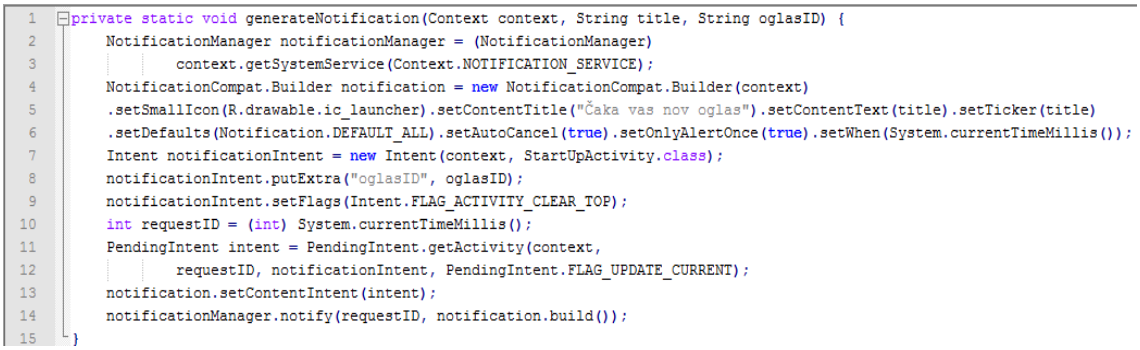
Njegova glavna metoda je `onReceive()` (Slika 5.21). Klicana je takrat ko prične GCM servis pošiljati sporočilo. Najprej se preveri, če je sporočilo tipa sporočilo in ne napaka ali odgovor, saj nas sporočila takšnih tipov ne zanimajo. Če gre za tip sporočilo, se sporočilo prebere , nakar se ustvari obvestilo s pomočjo metode `generateNotification()`. V nadaljevanju so vrstice, ki opravijo isto nalogo kot `AsyncTaskGCMRegistration`, saj v nekaterih primerih pride do napake s servisom GCM, čeprav je registracija uspela. V tem primeru se naprava registrira v tem delu.

Slika 5.21: Koda GcmBroadcastReceiver

```
public class GcmBroadcastReceiver extends BroadcastReceiver {
public void onReceive(Context context, Intent intent) {
    GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(context);
    if(isGCMRegisterd(context))
    {
        if(gcm.getMessageType(intent).equals(GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE))
        {
            String title = intent.getExtras().getString("Title");
            String oglasID = intent.getExtras().getString("Oglas_Id");
            Log.v("GCM message", "Title: " + title + " ID: " + oglasID);
            generateNotification(context, title, oglasID);
        }
    }
    String regId = intent.getExtras().getString("registration_id");
    if(regId != null && !regId.equals("")) {
        Log.v("GCM Reg", "BroadcastReciver registration");
        Object[] stringContext = new Object[] {regId, context};
        new AsyncTask<Object, Object, Object>() {
            @Override
            protected Object doInBackground(Object... params) {
                String registrationID = (String) params[0];
                Context anonContext = (Context) params[1];
                AsyncTaskRegisterGCM.storeRegistrationId(anonContext, registrationID);
                AsyncTaskRegisterGCM.sendRegistrationIdToServer(registrationID);
                return null;
            }
        }.execute(stringContext);
    }
}
```

Tudi prej omenjena metoda `generateNotification()` je metoda tega razreda (Slika 5.22). Ta metoda je zelo pomembna, saj ustvari obvestilo, o katerem smo obveščeni v statusni vrstici.

Slika 5.22 : Koda GcmBroadcastReceiver generateNotification



```
private static void generateNotification(Context context, String title, String oglasID) {
    NotificationManager notificationManager = (NotificationManager)
            context.getSystemService(Context.NOTIFICATION_SERVICE);
    NotificationCompat.Builder notification = new NotificationCompat.Builder(context)
    .setSmallIcon(R.drawable.ic_launcher).setContentTitle("Čaka vas nov oglas").setContentText(title).setTicker(title)
    .setDefaults(Notification.DEFAULT_ALL).setAutoCancel(true).setOnlyAlertOnce(true).setWhen(System.currentTimeMillis());
    Intent notificationIntent = new Intent(context, StartUpActivity.class);
    notificationIntent.putExtra("oglasID", oglasID);
    notificationIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    int requestID = (int) System.currentTimeMillis();
    PendingIntent intent = PendingIntent.getActivity(context,
            requestID, notificationIntent, PendingIntent.FLAG_UPDATE_CURRENT);
    notification.setContentIntent(intent);
    notificationManager.notify(requestID, notification.build());
}
```

5.3 Strežnik

Strežniški del aplikacije je implementiran v jeziku c# z uporabo orodja Visul Studio 2010. Prav tako ima strežnik svojo podatkovno bazo. Za upravljanje podatkovne baze smo uporabljali program SQL Server 2008.

Na strežniku teče program, ki ob zagonu zažene servis, imenovan WcfZbitaCena. Program skrbi, da se vsako uro pridobi nov XML z oglasi. Novi oglasi se shranijo v podatkovno bazo, nato pa se pošljejo obvestila uporabnikom, ki so na obvestila naročeni.

Servis WcfZbitaCena skrbi za izvrševanje zahtev, katere pošlje oddaljena aplikacija (odjemalec). To pomeni, da je dostopna točka strežnika. Servis ob zahtevi odjemalca izvrši metodo, ki pridobi oglase iz podatkovne baze ali pa vanjo shrani podatke.

Pomemben del strežniške aplikacije je tudi razred `DataManager`, ki skrbi za rokovanje s podatkovno bazo strežnika.

5.3.1 Strežniški program

Ob pogonu programa se izvrši metoda static void `Main()`, ki kliče konstruktor tega razreda (Slika 5.23). Glavno delo se izvrši v tem konstruktorju. Ena izmed nalog programa je, da zažene servis WcfZbitaCena. To se zgodi s pomočjo razreda `ServiceHost`, kar je prikazano v 12. in 13. vrstici na sliki 5.26.

V drugem delu se inicializirata spremenljivki tipa `TaskTimer` in `TaskWork`, ter izvede metodo`StartNewTaskWorkThread()`, ki opravi delo pridobivanja novih oglasov in pošiljanja obvestil GCM strežniku, ki jih kasneje posreduje uporabniku. Prav tako se v razredu TaskTimer pokliče metoda, ki se izvrši, ko poteče določen čas. V tej metodi se prav tako kliče metoda `StartNewTaskWorkThread()` razreda `TaskWork`.

Slika 5.23: Koda Program

```
class Program
{
    private TaskWork work;
    private const string URL_ZBITA_CENA_XML = "http://zbitacena.si/ljubljana/xmls/ceneje.xml";
    static void Main(string[] args)
    {
        Program program = new Program();
    }

    public Program()
    {
        System.ServiceModel.ServiceHost host = new System.ServiceModel.ServiceHost(typeof(WcfZbitaCena.TestService), new Uri("http://localhost:8780/WcfZbitaCena"));
        host.Open();
        Console.WriteLine("Service host running.....");

        work = new TaskWork(URL_ZBITA_CENA_XML);
        work.StartNewTaskWorkThread();

        TaskTimer timer = new TaskTimer(60);
        timer.Start();
        timer.TaskTimerElapesd(new ElapsedEventHandler(timer_Elapsed));

        Console.ReadLine()
        host.Close();
    }

    private void timer_Elapsed(Object sender, ElapsedEventArgs args)
    {
        work.StartNewTaskWorkThread();
    }
}
```

Z razredom `TaskTimer` upravljamo z razredom `Timer` (Slika 5.24). Njegov namen je izvršitev določenega opravila ob izteku predpisanega časa. Razred v konstruktorju sprejme čas v minutah, nato izračuna koliko je to milisekund in inicializira `Timer`, da se sproži vsakih nekaj minut. Odštevalnik prične odštevati čas ob klicu metode `Start()`. Metoda `TaskTimerElapsed()`, sprejme funkcijo, `eventHandler`, ki se izvrši ko čas poteče.

Glavni del programa je poleg zagona servisa tudi pridobivanje novih oglasov in pošiljanje obvestil. Ta dejavnost se izvršujev razredu `TaskWork` (Slika 5.25). Ob klicu get metode za nit se vedno vrne nova nit. Ostale operacije se izvršijo znotraj te niti v funkciji `taskWorkThreadExecution`().

Slika 5.24: Koda TaskTimer

```
public class TaskTimer {
    private Timer taskTimer;
    public TaskTimer(double elapsedMinutes){
        taskTimer = new Timer(elapsedMinutes * 60 * 1000);
        taskTimer.AutoReset = true;
    }
    public void Start(){
        taskTimer.Start();
    }
    public void TaskTimerElapesd(ElapsedEventHandler elapsedHandler){
        taskTimer.Elapsed += elapsedHandler;
    }
}
```

Slika 5.25: Koda TaskWork

```
public class TaskWork{
    private string xmlUrl;
    private Thread workerTherad;
    private DataManager manager;
    public TaskWork(string xmlUrl){

    public Thread WorkerTherad{
        get
        {
            workerTherad = new Thread(new ThreadStart(TaskWorkThreadExecution));
            return workerTherad;
        }
    }
    public string XmlUrl{

    public void TaskWorkThreadExecution()
    {
        Console.WriteLine("Event start " + DateTime.Now);
        List<OglasXML> vsiOglasi = GetXmlFromUrl();
        if (vsiOglasi != null)
        {
            //1. Pridobimo samo nove oglase
            List<OglasXML> noviOglasi = GetOnlyNewOffers(vsiOglasi);
            //2. nove oglase se doda v podatkovno bazo
            manager.InsertOffers(noviOglasi);
            //3. Dodelimo naprej definirane ključne besede uporabnika
            AddKeyWordsToNewOffers(noviOglasi);
            //4. v podatkovni bazi se vsakemu uporabniko pošljejo pushi za oglase na katere so naročeni
            SendNotoficatons(noviOglasi);
        }
    }
```

Najprej se izvrši klic metode `GetXmlFromUrl()` (Slika 5.26), ki prebere XML datoteko in jo s pomočjo XML deserializacije spremeni v seznam objektov `OglasXML`.

Slika 5.26: Koda TaskWork getXmlFromUrl

```
private List<OglasXML> GetXmlFromUrl(){
    XmlReader reader = new XmlTextReader(xmlUrl);

    XmlSerializer serializer = new XmlSerializer(typeof(OglasListXML));
    OglasListXML vsiOglasi;
    vsiOglasi = (OglasListXML)serializer.Deserialize(reader);

    return vsiOglasi.Oglasi;
}
```

Nato z metodo `GetOnlyNewOffers()` iz seznama z vsemi oglasi pridobimo le oglase, ki v naši podatkovni bazi še ne obstajajo.

V metodi `AddKeyWordsToNewOffers()` se preverijo vnaprej definirane skupine ključnih besed (potovanja, izlet, oblačila, naprave itd.). To se stori tako, da se uporabi SQL poizvedba. V tej poizvedbi imamo ključno besedo in uporabimo ključno besedo LIKE. Če objekt DataTable ni prazen, to pomeni, da je ključna beseda vsebovana.

Zadnja metoda `SendNotification()` pošlje obvestila, da so prisotne nove ponudbe.

Najprej se izvede primerjava ključnih besed in če so skupne besede najdene,se pošlje obvestilo uporabniku. Psevdo kod funkcije `SendNotification()` imamo prikazan na sliki5.27.

Slika 5.27: Pseudo kod SendNotification

```
1  Foreach( uporabnika vpisanega v podatkovni bazi)
2      var ključneBesede = pridobimo_uporabnikove_kljucne_besede()
3      Foreach(nov oglas)
4          var ključneBesedeOglasa = pridobi_ključne_besede_oglasa()
5          if(najdene_skupne_ključne_besede)
6              pošlji_obevestilo()
7      endForeach
8  endForeach
```

Pošiljanje obvestil uporabniku se stori s pomočjo razreda GCM. Razred GCM ima samo eno metodo, ki se imenuje `sendGCMNotification()` (Slika 5.28). Le-ta sprejme parameter sporočilo. S pomočjo razreda `HttpWebRequest` ustvarimo POST zahtevo, v tem razredu pošljemo sporočilo strežniku GCM v json formatu. API ključ, ki ga pri tem uporabimo smo pridobili v že prej omenjeni Google konzoli, kjer smo registrirali lasten Google API.

Slika 5.28: Koda GCM SendNotification

```
public static string API_KEY = "ključ apija";
public string SendGCMNotification(string postData, string postDataContentType = "application/json")
{
    ServicePointManager.ServerCertificateValidationCallback += new RemoteCertificateValidationCallback(ValidateServerCertificate);
    byte[] byteArray = Encoding.UTF8.GetBytes(postData);
    HttpWebRequest Request = (HttpWebRequest)WebRequest.Create("https://android.googleapis.com/gcm/send");
    Request.Method = "POST";
    Request.KeepAlive = false;
    Request.ContentType = postDataContentType;
    Request.Headers.Add(string.Format("Authorization: key={0}", GCM.API_KEY));
    Request.ContentLength = byteArray.Length;
    Stream dataStream = Request.GetRequestStream();
    dataStream.Write(byteArray, 0, byteArray.Length);
    dataStream.Close();
    try
    {
        WebResponse Response = Request.GetResponse();
        return "succesful";
    }
    catch (Exception e)
    {
    }
    return "error";
}
```

Sporočilo, ki ga pošljemo, se sestavi v metodi `CreatePostData()` v razredu `TaskWork` (Slika 5.29). Metoda prejeme parameter `deviceID`, ki GCM strežniku pove, kateri napravi mora posredovati sporočilo, parameter `oglasID`, ki predstavlja ID oglasa nekega obvestila, da ga lahko nato odjemalec pridobi in zadnji parameter naslov, ki predstavlja naslov, katerega ima oglas. Sporočilo sestavimo ročno v json formatu. `Data` je jedro našega sporočila in predstavlja naše podatke.

Slika 5.29: Koda TaskWork CreatePostData

```
private string CreatePostData(string deviceID, string oglasID, string title)
{
    string postData =
    "{ \"registration_ids\": [ \"" + deviceID + "\" ], " +
        "\"data\": {\"Title\":\"" + title + "\", " +
                    "\"Oglas_Id\": \"" + oglasID + "\"}}";
    return postData;
}
```

5.3.2 Servis WcfZbitaCena

WCF servis WcfZbitaCena predstavlja vstopno točko do našega strežnika (Slika 5.30). Odjemalčeva aplikacija lahko dostopa le do metod servisa. Ob klicu metode se ta izvede in nato vrne rezultat.

V našem primeru je rezultat vrnjen v formatu json.

Definirane so naslednje metode:

- `SaveUserSettings()`

Metoda sprejema objekt v formatu json. Nato pa objekt shrani v podatkovno bazo.

Slika 5.30 : Koda vemsnik IServiceOglasi

```
[ServiceContract]
public interface IServiceOglasi
{
    [OperationContract]
    [WebInvoke(Method = "GET",
               ResponseFormat = WebMessageFormat.Json,
               UriTemplate = "OglasById/ID={id}")]
    OglasCeloten GetOglasWithId(string id);

    [OperationContract]
    [WebInvoke(Method = "GET",
               ResponseFormat = WebMessageFormat.Json,
               UriTemplate = "OglasiByKeyWord/Search={keyWord}")]
    List<OglasKratek> GetOglasiWithKeyWord(string keyWord);

    [OperationContract]
    [WebInvoke(Method = "GET",
               ResponseFormat = WebMessageFormat.Json,
               UriTemplate = "VsiAktivniOglasi")]
    List<OglasKratek> GetAllOglasi();

    [OperationContract]
    [WebInvoke(Method = "GET",
               ResponseFormat = WebMessageFormat.Json,
               UriTemplate = "/RegisterDevice/DeviceID={deviceID}")]
    bool SetUserDeviceId(string deviceID);

    [WebInvoke(Method = "POST", UriTemplate = "/SaveUserSettings",
               BodyStyle = WebMessageBodyStyle.WrappedRequest,
               RequestFormat = WebMessageFormat.Json, ResponseFormat = WebMessageFormat.Json)]
    [OperationContract]
    string SaveUserSettings(UserSettings settings);
}
```

- `GetOglasWithId()`

  Metoda sprejme parameter id, ki predstavlja id oglasa, ki ga želimo pridobiti. Ob izvedbi se izvede SQL (angl. Structured Query Language) poizvedba, s katero podatkovni bazi poišče oglas z enakim id-jem in ga tudi vrne.

- `.GetAllOglasi()`

  Metoda nam vrne vse aktivne oglase. Aktivni oglasi so tisti, katerih datum poteka se še ni iztekel.

- `GetOglasiWithKeyWord()`

  Podobno se zgodi tudi v metodi `GetOglasiWithKeyWord()`, le da pri tem vrnemo oglase, katerih opis vsebuje podano ključno besedo »keyWord«.

- `SetUserDeviceId()`

V registraciji SetUserDeviceId() se dobljen parameter vpiše v podatkovno bazo in predstavlja napravo, ki ga je poslala.

V zadnji metodi, kjer dobimo uporabnikove nastavitve, se najprej preveri, če ima uporabnik nastavitve že shranjene. Če te nastavitve obstajajo, se jih izbriše in doda nove. Če jih še ni, pa se dodajo nove.

6 SKLEP

Aplikacija deluje v skladu z zastavljenimi cilji. Uporabnik je o novi ponudbi obveščen v trenutku, ko se poveže s svetovnim spletom. Procesorsko zahtevna opravila se opravijo na strežniku, s čemer prihranimo porabo baterije na telefonu. Uporabniku je omogočeno dodajati lastne ključne besede, tako da se lahko sam odloči, kateri kuponi so mu v interesu. Spoznali smo, da imam Google zelo dobro urejeno dostavo obvestil. Dostop do funkcionalnosti je zelo enostaven. Prav tako je enostavna uporaba Google GCM vmesnika.

Aplikacija ima še vseeno veliko prostora za nadgradnjo. Ena njenih večjih slabosti je, da si lahko v aplikaciji le ogledujemo oglase oz. kupone, za njihov nakup pa potrebujemo brskalnik, s katerim obiščemo spletno mesto, od koder smo kupon dobili. S tem uporabnik zapusti našo aplikacijo.

Prav tako bi lahko omogočili ocenjevanje in komentiranje oglasov. Tako bi lahko uporabniki, ki so že uporabili kupon, zapisali svoje mnenje. A večina kuponov je naprodaj le kakšen teden, zato bi morda bilo bolj smiselno, da bi se ocenjevali ponudniki kuponov. Prav tako se veliko kuponov uporabi šele po pretečeni ponudbi.

Aplikacijo bi lahko razširili tudi z možnostjo delitve oglasa, in sicer preko vedno bolj popularnih socialnih omrežij.

Če smo naročeni na prejemanje elektronske pošte ponudnika kuponov, velikokrat ne moremo izbirati obveščanja in smo tako obveščeni o vseh oglasih. Zato bi lahko uvedli dodatno storitev, ki bi posredovala na elektronsko pošto samo želene oglase.Vsekakor ima aplikacija še veliko možnosti za nadgradnjo.

7 VIRI

[1] Activity. (b.d.). Pridobljeno 11. 09. 2013, iz http://developer.android.com/reference/android/app/Activity.html

[2] Android (operating system). (b.d.). Pridobljeno 11. 09. 2013, iz http://en.wikipedia.org/wiki/Android_%28operating_system%29

[3] Chappell, D. (2007). *Introducing Windows Communication Foundation*. (b.d.). 12. 9. 2013 http://msdn.microsoft.com/sv-se/library/dd943056%28en-us%29.aspx

[4] Eclipse (b.d.). Pridobljeno 12 . 09. 2013, iz http://www.eclipse.org/downloads/packages/eclipse-standard-43/keplerr

[5] Gingerbread. (b.d.). Pridobljeno 11. 09. 2013, iz http://developer.android.com/about/versions/android-2.3-highlights.html

[6] GCM Overview. (b.d.). Pridobljeno 11. 09. 2013, iz http://developer.android.com/google/gcm/gcm.html

[7] Google console (b.d.). Pridobljeno 12. 09. 2013 iz https://code.google.com/apis/console/

[8] JSON. (b.d.). Pridobljeno 11. 09. 2013, iz http://en.wikipedia.org/wiki/JSON

[9] Microsoft visual studio 12. 09. 2013, iz http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express

[10] Microsoft SQL server 12.9.2013 iz http://www.microsoft.com/en-us/sqlserver/editions/2012-editions/express.aspx

[11] Sharp, J. 2010. *Windows® Communication Foundation 4 Step by Step*. ZDA, Kalifornija: O'Reilly Media

[12] What Is Windows Communication Foundation. (b.d.). 2013, iz http://msdn.microsoft.com/en-us/library/ms731082.aspx

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

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Bojan Orter,

z vpisno številko E1041547,

sem avtor/-ica diplomskega dela z naslovom:

MOBILNA APLIKACIJA ZA OBVEŠČANJE O DNEVNIH PONUDBAH

S svojim podpisom zagotavljam, da:

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

  doc. dr. Tomaž Kosar

  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 zbirki »Dela FRI«.

V Mariboru, dne 13.6.2013 Podpis avtorja/-ice: Bojan Orter

Sprejeto na Komisiji za študijske zadeve Senata FERI 14. 12. 2009

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

IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Podpisani mentor Doc. dr. Tomaž Kosar izjavljam, da je
(ime in priimek mentorja)
študent Bojan Orter izdelal diplomsko
(ime in priimek študenta-tke)

delo z naslovom: MOBILNA APLIKACIJA ZA OBVEŠČANJE O DNEVNIH PONUDBAH

(naslov diplomskega dela)

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

Datum in kraj: 13.9.2013, Maribor

Podpis mentorja:

Tomaž Kosar

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: Bojan Orter

Vpisna številka: E1041547

Študijski program: Računalništvo in informacijske tehnologije

Naslov diplomskega dela: MOBILNA APLIKACIJA ZA OBVEŠČANJE O DNEVNIH PONUDBAH

Mentor: doc. dr.Tomaž Kosar

Somentor: _____

Podpisani-a Bojan Orter 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: Maribor, 13.9.2013

Podpis diplomanta-tke:

Bojan