Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Andrej Belcijan

Mobilna aplikacija za študente fakultete za računalništvo in informatiko

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

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

Št. naloge: 00279/2012
Datum: 12.04.2012

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

Kandidat: **ANDREJ BELCIJAN**

Naslov: **MOBILNA APLIKACIJA ZA ŠTUDENTE FAKULTETE ZA RAČUNALNIŠTVO IN INFORMATIKO**

MOBILE APPLICATION FOR STUDENTS OF FACULTY OF COMPUTER AND INFORMATION SCIENCE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V okviru študijskega procesa na FRI se aktivno uporablja sistem Moodle, ki študentom omogoča dostop do različnih podatkov vezanih na študijske obveznosti. Za študente bi torej bila zelo uporabna mobilna aplikacija, ki jim omogoča dostop do podatkov na sistemu Moodle. Poleg podatkov iz sistema Moodle bi bile v okviru mobilne aplikacije verjetno študentom uporabne še druge storitve.

Med študenti FRI izvedite anketo o tem, katere storitve vezane na podatke iz sistema Moodle bi v okviru mobilne aplikacije potrebovali v stanju mobilnosti. Anketa naj pokrije tudi ostale storitve, ki bi jih študentje potrebovali v okviru mobilne aplikacije. Na podlagi ankete razvijte mobilno aplikacijo mFRI in vmesnik do sistema Moodle.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Andrej Belcijan, z vpisno številko **63070034**, sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za študente fakultete za računalništvo in informatiko

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Klepca in somentorstvom izr. prof. dr. Roka Rupnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 27. avgusta 2012 Podpis avtorja:

Kazalo

Povzetek

V diplomski nalogi bomo predstavili razvoj aplikacije mFRI za mobilne naprave z operacijskim sistemom Android. Cilj je olajšati življenje študentom Fakultete za računalništvo in informatiko. Glavna komponenta aplikacije je odjemalec Moodle, ki omogoča dostop do spletne učilnice. Stranska orodja so pregledovalnik restavracij, bralnik novic fakultete, pregledovalnik prihodov avtobusov in drugi pripomočki. Aplikacija je zasnovana razširljivo in tako primerna za nadgradnjo ob pojavi novih potreb uporabnikov. V okviru naloge smo predstavili postopke implementiranja različnih funkcionalnosti v Androidu, ki lahko neizkušenim razvijalcem zaradi nekoliko pomanjkljive dokumentacije včasih vzamejo veliko časa. Poleg tega smo opisali postopek nastavitve portala Moodle za pravilno delovanje z našo mobilno aplikacijo in razvili preprost strežnik oglasov.

Abstract

In the thesis we will present the development of the mFRI application for mobile devices based on the Android operating system. The goal is to facilitate the lives of students of the Faculty of Computer and Information Science. The main component of the application is a Moodle client, which enables access to the web classroom. Auxiliary tools include an overview of restaurants, a faculty news reader, a bus arrival time display and other utilities. The application is scalable and thus suitable for upgrades, were new user needs to arise. Within the thesis we presented the procedures for implementing various Android functionalities that could otherwise take a great amount of time from inexperienced developers due to somewhat insufficient documentation. We have also described the procedure for configuring Moodle to function correctly with our application and developed a simple ad server.

Poglavje 1

Uvod

Z razvojem mobilne telefonije in samih aparatov se njihova uporabnost z vsako novo generacijo povečuje. Od začetnih primarnih funkcij, kot je bilo osnovno klicanje, so se storitve postopoma razširile na kratka sporočila SMS[1] in kasneje z razvojem video zajemnih naprav na sporočila MMS[2]. S širokimi razvojnimi možnostmi in operacijskimi sistemi so mobilne naprave začele nadomeščati glasbene predvajalnike, preko njih smo lahko z dostopom do interneta obiskovali svoje najljubše spletne strani, v zadnjih letih pa so mobilni aparati vse bolj postajali večnamenske naprave, kljub ohranjanju svoje primarne funkcije, torej opravljanju telefonskih klicev.

Z razvojem mobilne naprave niso bile več omejene na aplikacije, katere je predhodno namestil proizvajalec, pač pa si je lahko lastnik naprave le-te prilagodil po svojih lastnih željah in preferencah. S pojavom spletnih trgovin, kot je Google Android market (Google Play), lahko lastniki pametnih telefonov sedaj izbirajo med aplikacijami, ki točno ustrezajo njihovim potrebam za opravljanje posameznih nalog oziroma funkcij, znotraj teh pa med najbolj priljubljenimi tudi njim všečen vizualni izgled posamezne aplikacije.

Na podlagi izkušenj s programskim jezikom Java, ki smo jih pridobili v času študija, smo se odločili, da razvijemo mobilno aplikacijo za operaci-

[1]Short Message Service
[2]Multimedia Messaging Service

jski sistem Android. Njen cilj bo olajšati življenje študentom Fakultete za računalništvo in informatiko. Nastala je aplikacija mFRI, ki trenutno vsebuje štiri glavne komponente. To so odjemalec za spletno učilnico FRI, ki temelji na portalu Moodle, pripomoček za pregled časov prihoda mestnih avtobusov v bližini fakultete, osnovne informacije o restavracijah in bralnik uradnih novic RSS fakultete.

Z vse hitrejšim razvojem tehnologije, ki poganja mobilne naprave bodo takšne in podobne aplikacije postajale vse bolj napredne in prijaznejše uporabnikom. S povezovanjem posameznih protokolov in funkcij preko interneta pa si lahko obetamo še bolj prilagodljive storitve, s katerimi bomo na svoje mobilne naprave dobili ažurne in točne podatke ter informacije v trenutku, ko jih bomo potrebovali.

Poglavje 2

Razvojna orodja

2.1 Linux

Tako diplomska naloga kot mobilna aplikacija sta bili v celoti izdelani v odprtokodnem operacijskem sistemu GNU/Linux, natančneje z distribucijo Ubuntu Linux različice 11.10 s 64-bitno arhitekturo. [1]

Razvoj aplikacije Android je bil na izvedljiv zato, ker je podjetje Google poskrbelo, da razvojna orodja za Android delujejo na treh najpopularnejših operacijskih sistemih. To so Microsoft Windows, Apple Mac OS X in sistemi, ki temeljijo na jedru Linux.

2.2 Eclipse IDE

Trenutno je za razvoj aplikacij Android najbolj razširjeno orodje Eclipse. Gre za odprtokodno razvojno okolje, ki omogoča programiranje v številnih programskih jezikih. Izbran je bil na podlagi široke podpore za razvoj Android aplikacij in odlične kompatibilnosti z razvojnimi orodji Android. Podjetje Google tudi uradno podpira razvojno okolje Eclipse.

2.3 Android SDK

Android SDK[1] je paket razvojnih orodij, ki jih potrebuje razvijalec mobilne aplikacije. Vsebuje knjižnice, dokumentacijo, vodiče, primere, emulator, razhroščevalnik, dodatke za okolje Eclipse in mnoga druga orodja. Temelji na programskem jeziku Java, vzdržuje pa podjetje Google. Ob vsaki novi različici sistema Android podjetje izda tudi novo različico razvojnih orodij. Novejše različice vsebujejo nekatere knjižnice z razredi in metodami, ki jih starejše različice Androida ne poznajo. To pomeni, da mora razvijalec pazljivo izbrati, katero različico orodij bo uporabljal. Google podpira vse stare različice, zato lahko izberemo katerokoli.

Za našo aplikacijo smo izbrali najnovejšo različico SDK, ki podpira Android API[2] 16 in hkrati zagotovili združljivost za nazaj vse do različice 8. API je vmesnik za programiranje aplikacij. Ta je namenjen operacijskemu sistemu Android Jelly Bean 4.1. Na sliki 2.1 si lahko ogledamo, katera različica API pripada določeni različici operacijskega sistema Android. Na sliki 2.2 vidimo da ima najnovejši Android zelo malo uporabnikov, zato bi bilo smiselno uporabljati starejšo različico. Vendar to ni potrebno, saj lahko svobodno uporabljamo najnovejšo različico ter se pri tem izogibamo uporabi razredov in metod, ki jih starejše različice ne poznajo. Na to nas med programiranjem samodejno opozori okolje Eclipse.

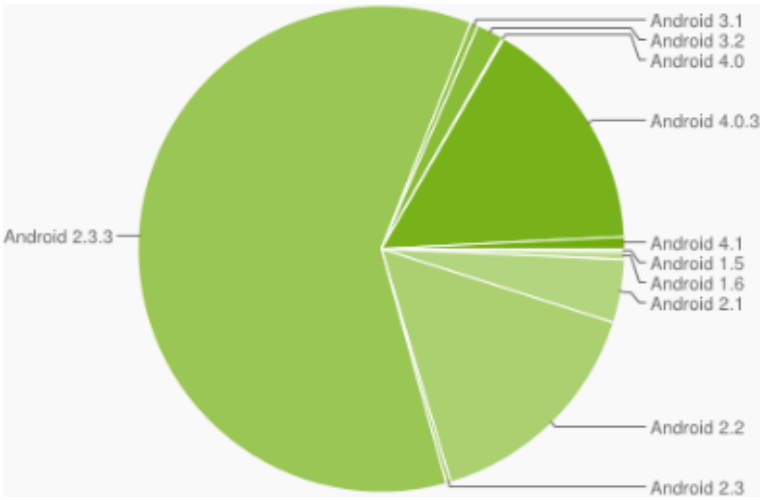
2.4 Razvoj na mobilni napravi

1. Poskrbeti moramo, da naša aplikacija dovoljuje razhroščevanje. To običajno storimo tako, da v datoteko `AndroidManifest.xml` v element *application* dodamo argument `android:debuggable="true"`. V primeru, da uporabljamo Eclipse nam tega ni potrebno storiti, saj razvojno okolje avtomatsko poskrbi za vklop razhroščevanja.

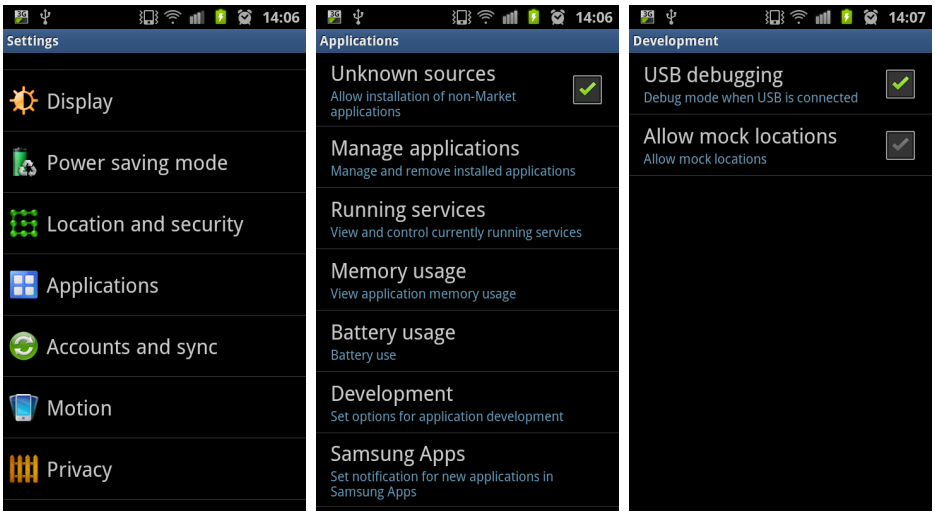
[1] Software Development Kit
[2] Applicaiton Programming Interface

Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.2%
2.2	Froyo	8	15.5%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	60.3%
3.1	Honeycomb	12	0.5%
3.2		13	1.8%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.1%
4.0.3 - 4.0.4		15	15.8%
4.1	Jelly Bean	16	0.8%

Slika 2.1: Različice Androida in nivoji API ter njihova razširjenost. [2]



Slika 2.2: Razširjenost različic Androida po deležu. [2]



Slika 2.3: Trije koraki do vklopa razhroščevanja USB na mobilni napravi.

2. Na napravi Android moramo vklopiti razhroščevanje USB. To storimo v nastavitvah na napravi kot je prikazano na sliki 2.3.
3. Na sistemu Ubuntu Linux moramo dodati pravilo udev, zato da bo naša mobilna naprava avtomatsko prepoznana. Udev je upravitelj naprav na jedru Linux. To storimo tako, da kot uporabnik root ustvarimo datoteko `/etc/udev/rules.d/51-android.rules`. Z ukazom poskrbimo, da lahko vsi uporabniki berejo njeno vsebino in nato vanjo zapišemo sledečo vrstico:

   ```
   SUBSYSTEM=="usb", ATTR{idVendor}=="04e8", MODE="0666",
   GROUP="plugdev"
   ```

   Argument `ATTR{idVendor}=="48e8"` določa identifikacijsko kodo proizvajalca mobilne naprave, ki jo najdemo v spletni dokumentaciji. [3] Koda "`04e8`"pripada proizvajalcu *Samsung*.

2.5 Android Debug Bridge

Android Debug Bridge (ADB) je večnamenski program za rabo v ukazni vrstici, ki je del kompleta programskih orodij Android. Je glavno orodje pri

stopnja	črka	ime	pomen
1.	`v`	Verbose	obširne informacije
2.	`d`	Debug	razhroščevalne informacije
3.	`i`	Information	splošne informacije
4.	`w`	Warning	opozorila
5.	`e`	Error	napaka

Tabela 2.1: Preglednica nivojev sistemskih sporočil v Androidu

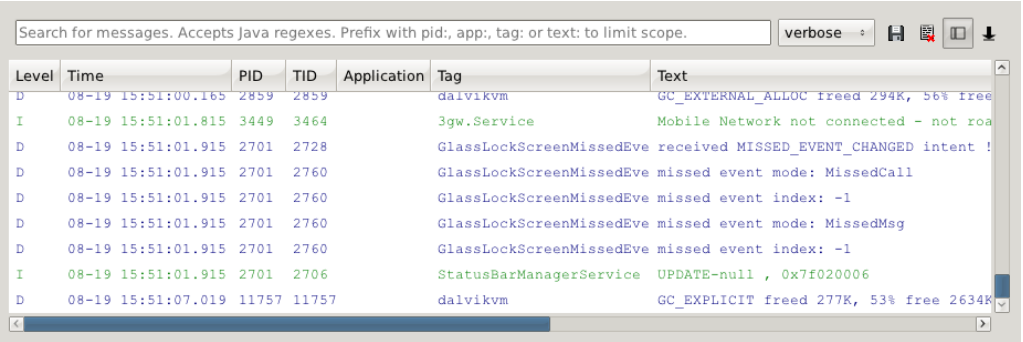
razvoju aplikacij Android in deluje na tri načine:

- kot odjemalec na razvojnem računalniku, ki sprejema ukaze
- kot strežnik na razvojnem računalniku, ki skrbi za povezavo z napravo
- kot proces na napravi ali emulatorju, ki se poveže s strežnikom

Pozna širok nabor ukazov za nadzor naprav Android. Omogoča prenos datotek med napravo in računalnikom v obe smeri, nameščanje ali odstranjevanje aplikacij na napravo iz računalnika, izvajanje ukazov na napravi, ustvarjanje varnostnih kopij podatkov naprave, izpis stanj naprave in še mnogo drugega. Ena njegovih bolj uporabnih funkcij je *LogCat*.

LogCat je Androidov mehanizem za zbiranje in pregledovanje sistemskih sporočil. Sistemsko sporočilo je vsako sporočilo, ki ga izpiše sistem ali aplikacija. Vsakič ko se na izhod izpiše sporočilo, ga *LogCat* zajame in po potrebi posreduje razvijalcu. Sistemska sporočila v Androidu imajo lahko 5 različnih stopenj. Od najmanj do najbolj kritičnih so razporejene sledeče:

Za njihov izpis mora načeloma poskrbeti razvijalec aplikacije. Če aplikacija naredi napako, se avtomatsko izpiše sporočilo stopnje *Error*. Tudi stopnjo posameznega sporočila določi razvijalec. Sporočilo izpiše tako, da kliče statično metodo razreda Log, ki ji poda dva niza: izvor sporočila in pa samo sporočilo. Ime metode je enako črki stopnje, ki jo razvijalec želi uporabiti. Za 3. stopnjo (*Information*) bi tako uporabil črko "`i`", kot prikazuje primer 2.1.



Slika 2.4: LogCat v okolju Eclipse.

Koda 2.1: Ukaz za izpis sporočila s stopnjo informacije.

```
Log.i("MainActivity", "Aplikacija aktivirana!");
```

Pri pregledovanju nato omejimo izpis na tisto stopnjo, ki nas zanima. Pogosto se zgodi, da je pri izpisovanju prve stopnje število sporočil preveliko. To otežuje iskanje informacij, ki nas zanimajo, zato je omejevanje izpisa na stopnjo zaželjeno. To na zelo preprost način omogoča dodatek za Eclipse, ki je vključen v paketu Android SDK. Prikazan je na sliki 2.4.

Poglavje 3

Aplikacija mFRI

Namen naše mobilne aplikacije je poleg integracije spletne učilnice Moodle poenostaviti vsakdan študentom Fakultete za računalništvo in informatiko. Sestavljena je iz 4 glavnih komponent:

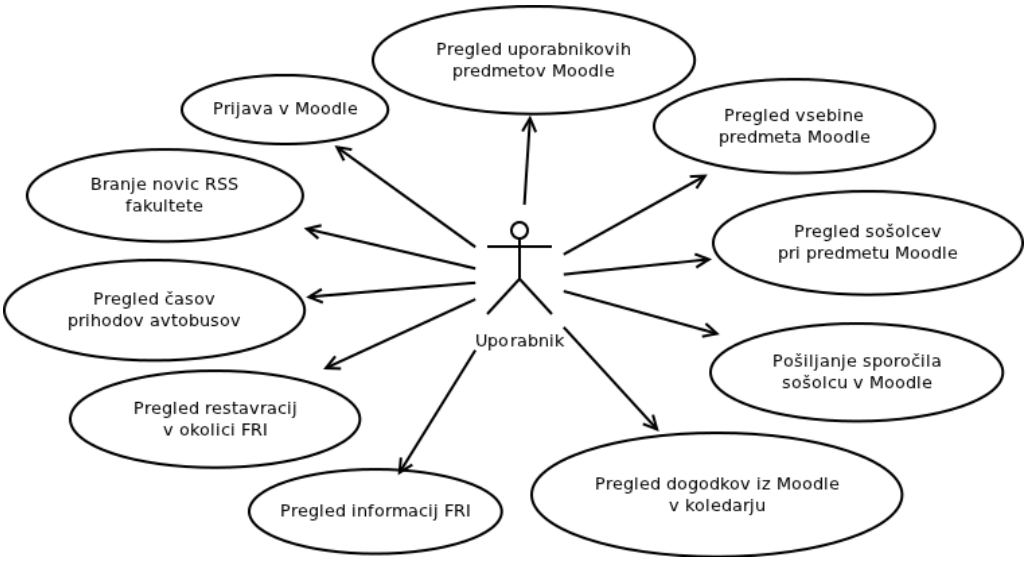
- Odjemalec za spletno učilnico FRI (Moodle).
- Pripomoček za pregled časov prihoda mestnih avtobusov v okolici. FRI
- Osnovne informacije o restavracijah v okolici FRI.
- Bralnik uradnih novic RSS[1] fakultete.

Vse možne uporabe aplikacije so prikazane z diagramom UML na sliki 3.1. Aplikacija je razširljiva, zato bo v prihodnosti mogoče dodajati nove komponente, ki bodo še dodatno obogatile nabor orodij na voljo študentom fakultete.

3.1 Glavni meni

Vstopna točka v aplikacijo je aktivnost z glavnim menijem. Aktivnost je posamezni uporabniški vmesnik aplikacije v katerem lahko uporabnik opravlja neko predvideno dejavnost. Temeljijo na razredu *Activity*, med njimi pa je mogoče prehajati s pomočjo objektov *Intent*. Naša aplikacija je zasnovana brez začasnega zaslona (angl. *splash screen*). Začetni zaslon je celozaslonska

[1] Rich Site Summary

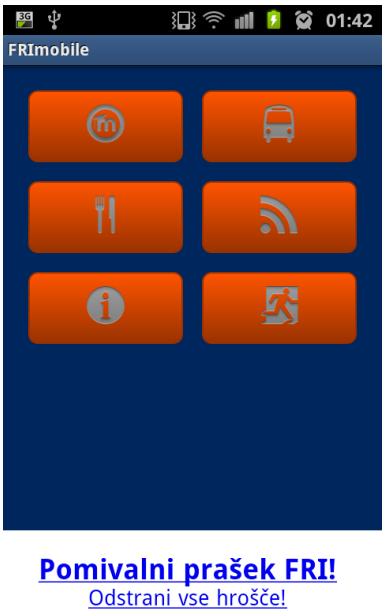


Slika 3.1: Diagram uporabe UML za aplikacijo mFRI.

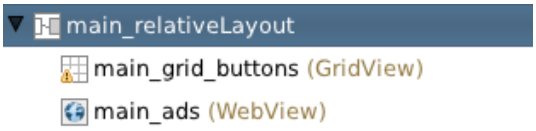
aktivnost, ki se za kratek čas (do nekaj sekund) prikaže ob zagonu aplikacije. Menimo, da takšna poteza otežuje uporabo in niža produktivnost. Vstop v našo aplikacijo je nemoten in takojšen. Uporabnik lahko nemudoma prične z uporabo. Kot vidimo na sliki 3.2 je glavni meni zelo preprost. Sprehajanje po aplikaciji mora biti čim bolj tekoče.

Glavna aktivnost je sestavljena iz treh elementov, ki so prikazani na sliki 3.3. Elementi so sledeči: *RelativeLayout*, *GridView* in *WebView*. Določeni so v datoteki XML[2], ki je priložena v imeniku `res/layout/`. Ob prvem zagonu aktivnosti moramo poklicati metodo `setContentView()`, ki ji kot parameter dodamo vir, datoteko XML. V našem primeru je to `R.layout.main`. To storimo v vsaki aktivnosti, ki smo ji postavitev elementov določili s pomočjo datoteke XML. V nasprotnem primeru bi morali za vsak element ročno ustvariti objekt in ga dodati aktivnosti. *RelativeLayout* je osnovni element v datoteki XML, ki skrbi za relativno postavitev svojih podelementov. *GridView* skrbi za izris gumbov glavnega menija, WebView pa prikaže oglas, kot nam ga posreduje strežnik oglasov. *RelativeLayout* je le eden izmed razre-

[2]Extensible Markup Language



Slika 3.2: Vstopna točka v aplikacijo mFRI.



Slika 3.3: Vstopna točka v aplikacijo mFRI.

dov postavitve *Layout*. Samo *RelativeLayout* omogoča postavitev oglasa na dno zaslona in hkratno zapolnitev preostalega prostora z drugimi elementi. To je pomembno, ker mora biti postavitev elementov enaka na napravah z različnimi ločljivostmi zaslona.

GridView je otrok Androidovega razreda *View* od katerega posredno deduje. Namenjen je prikazovanju dvodimenzionalne mreže elementov, ki mu jih posreduje objekt z vmesnikom *Adapter*. Uporabljen je bil zato, ker predstavlja edini način za enakomerno razporeditev nedoločenega števila elementov v mreži. *GridView* deluje tako, da od metode `getView()`, ki je prikazana s kodo 3.1, za vsako pozicijo, podano z argumentom `position`, pričakuje objekt *View*. Za osnovo smo uporabili razred *BaseAdapter* in ga razširili tako,

da posreduje gumbe, ki jih želimo imeti na zaslonu.

Koda 3.1: Metoda getView() razreda GridView.

```
public View getView(int position, View convertView,
    ViewGroup parent) {
  ImageButton imageButton;
  if (convertView == null) {
    imageButton = new ImageButton(context);
  } else {
    imageButton = (ImageButton) convertView;
  }
  imageButton.setOnClickListener(listeners[position]);
  imageButton.setImageResource(buttons[position]);
  imageButton.setBackgroundResource(R.layout.button_main);
  return imageButton;
}
```

Gumbi so objekti razreda *ImageButton*, ki jim določimo vir ozadja, vir ikone in objekt z vmesnikom *OnClickListener*. *OnClickListener* je preprost vmesnik, ki definira metodo `onClick(View v)`. Metoda v našem primeru vsebuje kodo, ki se izvrši ob pritisku na gumb. Določimo ga s pomočjo metode `setOnClickListener()`. Objekti *OnClickListener* so, tako kot viri ikon, definirani v polju znotraj razreda *BaseAdapter*. Primer 3.2 prikazuje inicializacijo objekta na podlagi vmesnika *OnClickListener*.

Koda 3.2: Koda za preusmeritev na izbrano aktivnost.

```
new OnClickListener() {
  public void onClick(View v) {
    Intent i = new
        Intent(context, MoodleCoursesActivity.class);
    startActivity(i);
  }
}
```

GridView je vsebovalnik, ki omogoča drsenje vsebine. Drsenje vsebine je smiselno v primeru, da na zaslonu ni dovolj prostora za njen prikaz v celoti. Ker se v našem primeru to ne more zgoditi, moramo poskrbeti, da je drsenje onemogočeno. To storimo tako, da mu nastavimo nov objekt vmesnika *OnTouchListener*. Ob zaznavi drsenja mora vrniti vrednost `true`, v nasprotnem primeru pa `false`. To storimo s kodo 3.3.

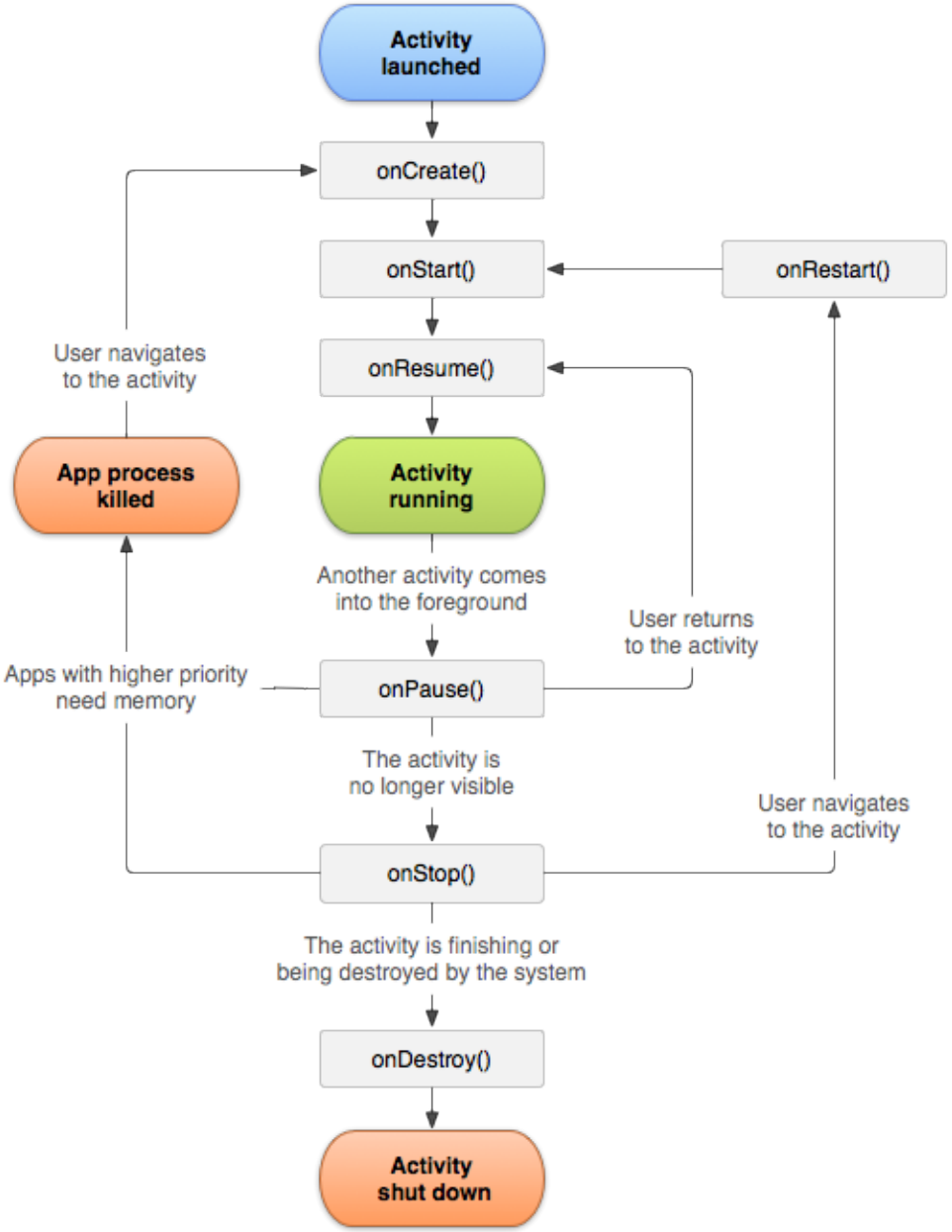
Koda 3.3: Koda za izklop drsenja na objektu GridView.

```
gridview.setOnTouchListener(new OnTouchListener() {
  public boolean onTouch(View v, MotionEvent event) {
    return (event.getAction() ==
        MotionEvent.ACTION_MOVE);
  }
});
```

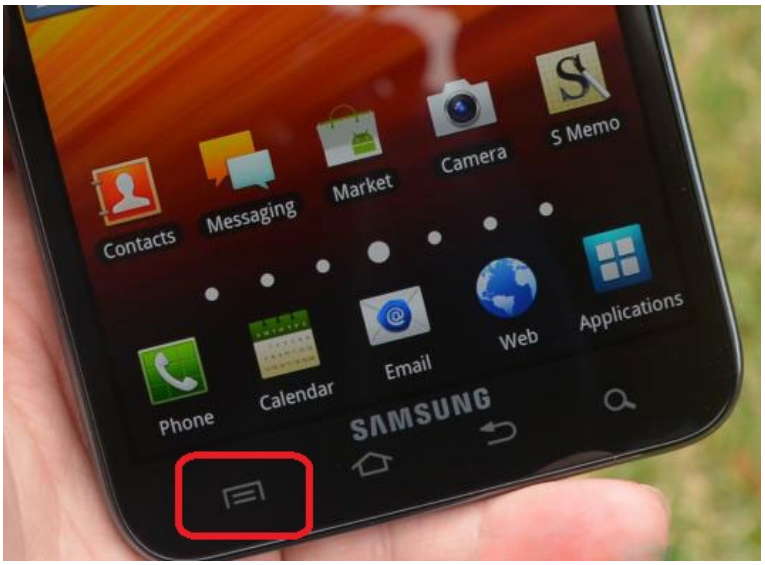
Izris vsebine *GridView* se izvede samo ob začetku aktivnosti znotraj metode `onStart()`. Izris oglasa se zgodi ob začetku in vsakič, ko se vrnemo na glavno aktivnost. Za to poskrbi metoda `onResume()`. To je zato, ker želimo, da se oglas ob vrnitvi na začetno stran zamenja. V nasprotnem primeru bi uporabnik videl vsakič enak oglas kot ob vstopu v aplikacijo. Diagram življenjskega cikla aktivnosti na sliki 3.4 nazorno prikazuje kdaj se izvedeta metodi `onStart()` in `onResume()`.

WebView je poseben vsebovalnik, ki je sposoben prikazovati spletno vsebino na enak način kot to stori spletni brskalnik. Oglas se vedno nahaja na enakem naslovu URL[3], zato je naloga *WebView* enostavna. Poklicati mora metodo `loadUrl()` z argumentom, ki predstavlja naslov reklame. Primer reklame je viden ob spodnjem robu na sliki 3.4.

[3]Uniform Resource Locator



Slika 3.4: Življenjski cikel aktivnosti v Androidu.



Slika 3.5: Gumb za opcijski meni na mobilni napravi Samsung.

Slika 3.6: Opcijski meni v glavni aktivnosti mFRI.

3.2 Opcijski meni

Aktivnosti, kjer so potrebne dodatne funkcionalnosti, imajo svoj lasten meni. Do njega se pride s pritiskom na fizično tipko na mobilni napravi. Tipko ima vsaka naprava, čeprav na različnih mestih. Na sliki 3.5 lahko vidimo primer tipke meni. V glavni aktivnosti ima meni štiri gumbe. Vsak izmed njih ima poleg ikone še besedilni opis svoje funkcije, kot je razvidno iz slike 3.6.

Na programskem nivoju se tovrstni meni ustvari z določitvijo dveh metod. V prvi, ki je prikazana s kodo 3.4, poskrbimo za prikaz menija, v drugi, ki je prikazana s kodo 3.5, pa določimo kaj se zgodi ob pritisku na posamezni gumb menija. V večini primerov samo preklopimo na neko drugo aktivnost.

Koda 3.4: Metoda onCreateOpsionsMenu().

```
public boolean onCreateOptionsMenu(Menu menu) {
  // Pridobi objekt, ki zna sestavit meni.
  MenuInflater inflater = getMenuInflater();
  // Sestavi meni po navodilu vira (datoteka XML).
  inflater.inflate(R.menu.main, menu);
  // Vrni vrednost "true" in s tem pove, da se je
     metoda uspela.
  return true;
}
```

Koda 3.5: Metoda onOptionsItemSelected().

```
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case R.id.preferences:
      // Pritisk na gumb "Nastavitve".
      startActivity(new
          Intent(this, SettingsActivity.class));
      return true;
    case R.id.about:
      // Pritisk na gumb "O programu".
      startActivity(new
          Intent(this, AboutActivity.class));
      return true;
    case R.id.help:
      // Pritisk na gumb "Pomoč".
      startActivity(new
          Intent(this, HelpActivity.class));
      return true;
    case R.id.exit:
      // Pritisk na gumb "Izhod".
      finish(); // Zapusti aktivnost (in s tem
```

```
            aplikacijo).
        return true;
      default: // Tega dela kode program praviloma
          nikoli ne izvede.
            // V primeru da ga, prepusti delo izvorni
               metodi.
      return super.onOptionsItemSelected(item);
    }
}
```

3.3 Razred Communicator

Metoda `loadUrl()`, ki jo vsebuje razred *WebView*, je zelo priročna pri branju vsebine iz določenega naslova. Vendar pa to ni vedno dovolj. V aplikaciji pogosto potrebujemo surove podatke z naslova, ki jih nameravamo obdelati in prikazati na poljuben način. V ta namen smo napisali razred *Communicator*. V osnovi omogoča izvedbo zahteve GET s parametri, odgovor strežnika pa vrne v željeni obliki.

Zahteve GET se pošiljajo s pomočjo objekta razreda *URLConnection*, ki je del Jave. Poda se mu naslov v obliki objekta URL in vklopi vse željene nastavitve. Nujno moramo vklopiti predpomnjenje (angl. *caching*), da se izognemo večkratnemu prenosu enakih podatkov. Prenos podatkov je lahko za uporabnika plačljiv, zato moramo poskrbeti, da je poraba čim manjša. Dobiti želimo tudi odgovor na zahtevo GET, zato metoda vrača tok vhodnih podatkov. Včasih to ne zadostuje, zato je metoda GET razdeljena na dve. Prva, prikazana s kodo 3.6, vrača tok podatkov, druga, prikazana s kodo 3.7, pa celoten odgovor v obliki niza, ki ga prebere iz toka.

Koda 3.6: Metoda GET() razreda Communicator (vrača InputStream).

```
public static InputStream GETis(String url,
   Map<String,String> data) throws IOException {
```

```
    // Sestavi objekt URL iz niza
    URL urlurl = new URL(url);
    // Odpri povezavo
    URLConnection connection = urlurl.openConnection();
    // Nastavi glavo "User-Agent"
    connection.setRequestProperty("User-Agent",
        userAgent);
    // Vklopi predpomnjenje (caching)
    connection.setUseCaches(true);
    // Vklopi izhod (radi bi prejeli odgovor)
    connection.setDoOutput(true);

    // Ustvari pisalni tok, ki bo povezavi posredoval
        podatke
    OutputStreamWriter osw = new
        OutputStreamWriter(connection.getOutputStream());

    // Podaj podatke pisalnemu toku
    osw.write(buildURLParameters(data));
    osw.flush();  // Poplavi pisalni tok
    osw.close();  // Zapri pisalni tok

    // Vrni povezavo do toka vhodnih podatkov
    return connection.getInputStream();
}
```

Koda 3.7: Metoda GET() razreda Communicator (vrača String).

```
public static String GET(String url,
    Map<String,String> data) throws IOException {
    // Tok podatkov odpri z bralnikom toka podatkov
    InputStreamReader isr = new
        InputStreamReader(GETis(url,data));
```

```
    // Preberi celoten tok podatkov in ga shrani v niz
    String rawText = readReader(isr);
    // Zapri bralnik toka podatkov
    isr.close();
    // Vrni surov niz
    return rawText;
}
```

3.4 Odjemalec Moodle

Glavna, najpomembnejša in najbolj zapletena komponenta je odjemalec za Moodle. Preden smo se sploh lahko lotili programiranja v Androidu, smo morali na testni strežnik namestiti portal Moodle in ga pravilno nastaviti. Moodle je zelo obsežen portal. Narejen je v programskem jeziku PHP in za delovanje potrebuje podatkovno bazo. V ta namen smo uporabili lastni strežnik Linux s spletnim strežnikom *Apache* in podatkovno bazo *MySQL*. Po namestitvi testnega portala Moodle je prišla na vrsto nastavitev. Uporabili smo avtentifikacija, dovoljenja in spletne storitve. Moodle budi ogromen nabor storitev in funkcionalnosti, ki so v večjem delu implementirane modularno. Avtentifikacija in spletne storitve sta dva primera modulov v sistemu Moodle (angl. *plugins*).

3.4.1 Avtentifikacija

Najprej moramo nastaviti samostojno ustvarjanje uporabniških računov. Moodle, ki je v uporabi na FRI, namenoma ne omogoča tovrstne avtentifikacije. Za ustvarjanje računov poskrbi mehanizem na univerzi. Samostojno ustvarjanje računov je uporabno izključno za razvoj mobilne aplikacije v testnem okolju Moodle. Moodle omogoča več avtentifikacijskih načinov. Administrator do njihovega seznama pride po sledeči poti:

Site administration → *Plugins* → *Authentication* → *Manage authentica-*

dovoljenje	za
`moodle/webservice:createmobiletoken`	vgrajeno mobilno storitev
`moodle/webservice:createtoken`	vse ostale storitve
`webservice/rest:use`	uporabo protokola REST
`moodle/course:view`	vpogled v predmet
`moodle/course:update`	spreminjanje predmeta
`moodle/course:viewhiddencourses`	pregled nevpisanih predmetov

Tabela 3.1: Pregled najpomembnejših dovoljen v Moodle in njihov pomen

tion

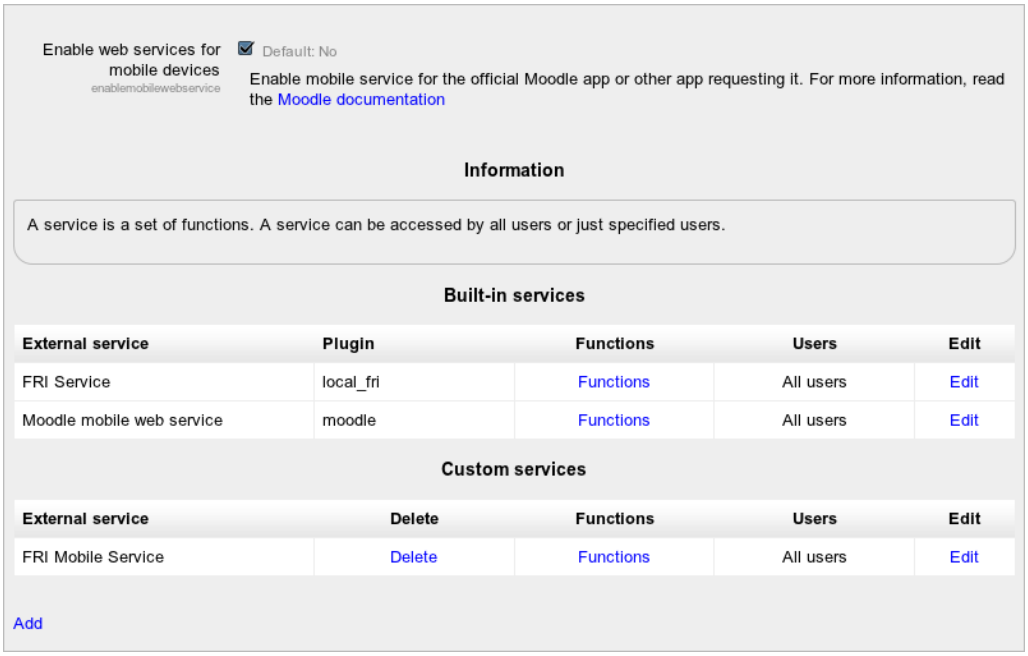
Za samostojno ustvarjanje uporabniških računov potrebujemo registracijo preko elektronske pošte. Na ta način se od novih uporabnikov zahteva potrditev registracije preko e-pošte. Pri seznamu avtentifikacijskih načinov moramo vklopiti ta način ter na isti strani spodaj v kategoriji *Common settings* izbrati *Self registration*. Izberemo *Email-based self-registration*.

3.4.2 Dovoljenja

Moodle določa dovoljenja uporabnikom na podlagi vlog. Za uporabo mobilne mora imeti vsak prijavljen uporabnik dovoljenje, da si sam ustvari avtentifikacijski žeton za spletne storitve. V nasprotnem primeru bi moral administrator vsakemu uporabniku posebej žeton določiti ročno. Vloga registriranega uporabnika se imenuje *Authenticated user*. Do strani za nastavljanje dovoljenj za vloge (npr. *Authenticated user*) pridemo po sledeči poti:

Site administration → *Users* → *Permissions* → *Define roles* → *Authenticated user* → *Edit*

Vloga *Authenticated user* mora imeti nastavljena vsaj dovoljenja, ki so navedena v tabeli 3.1.



Slika 3.7: Nastavitve storitev v portalu Moodle.

3.4.3 Spletne storitve

Spletne storitve so še eden izmed modulov, ki jih ponuja Moodle. Gre za relativno novost, dodano v verziji 2.0. Nastavljamo jih lahko v nadzornem sistemu portala Moodle, kot prikazuje slika 3.7.

Moodle pozna dve obliki spletnih storitev, vgrajene (angl. *Built-in services*) in po meri (angl. *Custom services*). Vsaka spletna storitev ima točno določen seznam funkcij, ki jih dovoljuje uporabljati. Razlika je v tem, da so funkcije pri vgrajenih storitvah programsko določene in jih ne moremo spreminjati. Storitve po meri naredi upravitelj portala Moodle in jim hkrati določi seznam funkcij.

Seznam vseh funkcij, ki jih nudi Moodle je trenutno še zelo omejen. Razvijalci portala imajo v načrtu svojo mobilno aplikacijo za Android, zato so v ta namen v Moodle dodali posebno vgrajeno storitev imenovano *Moodle mobile web service*. [4] Za našo aplikacijo bi bila idealna, a ima nekaj pomanjkljivosti. Funkcije, ki jih uporablja, so označene kot opuščene (angl. *deprecated*), kar

pomeni, da jih bodo v prihodnosti nadomestile druge. Poleg tega ne vsebuje nekaterih funkcij, ki jih nujno potrebujemo za normalno delovanje aplikacije. Ena izmed njih je na primer naša lastna funkcija za pridobitev dogodkov v obliki *iCalendar*. Do nadaljnjega bo zato naša aplikacija koristila storitev po meri. V portalu za sledenje hroščev v Moodle je objavljen okvirni načrt izdelave funkcij, ki bodo v prihodnosti na voljo.

Moodle kot ime storitve uporablja kratko ime, ki je določeno v podatkovni bazi v tabeli `external_services` pod stolpcem `shortname`. Težava je v tem, da v trenutni različici 2.3.1 ni dostopno preko nadzornega sistema Moodle. Potrebno ga je poiskati neposredno v podatkovni bazi. V primeru, da je polje v tabeli prazno, ga moramo nujno določiti. Privzeto kratko ime za vgrajeni *Moodle mobile web service* je `moodle_mobile_app`. Kratko ime, predvideno za mobilno aplikacijo FRI, je `fri_mobile_app`.

3.4.4 Avtentifikacija pri spletnih storitvah

Avtentifikacija poteka preko protokolov HTTP[4] ali HTTPS[5]. Predhodno registriran uporabnik portala Moodle pošlje zahtevo GET s svojim uporabniškim imenom in geslom ter kratkim imenom storitve, do katere želi imeti dostop na relativni naslov `/login/token.php`. Če je vse pravilno nastavljeno in ima uporabnik zadostne pravice, mu Moodle pošlje odgovor z žetonom v obliki JSON[6].

Uporabiti moramo enak naslov, kot je definiran v nastavitveni datoteki `config.php` na strežniku v korenskem imeniku portala Moodle. Če je naslov definiran na podlagi domene (na primer `https://ucilnica.fri.uni-lj.si/`), potem zahtev ne moremo pošiljati na naslov `https://212.235.188.24/login/token.php`, kljub temu da gre v obeh primerih za isti cilj. Razlog je v tem, da Moodle uporabnika vedno preusmeri tisto domeno, ki je določena v nastavitveni datoteki. V skripti `token.php` pa Moodle preusmeritev ne

[4]Hypertext Transfer Protocol
[5]Hypertext Transfer Protocol Secure
[6]JavaScript Object Notation

dovoljuje, zato pride do napake. Napaka je bila že prijavljena v portal za sledenje hroščev [13] (angl. *bug tracker*) z najvišjo prioriteto, a se ni vedelo zakaj do nje pride. Razvijalcem smo pojasnili, kako priti do napake in predlagali rešitev. Pravilno bi bilo, da bi bil uporabnik o rabi napačnega naslova obveščen. Popravilo hrošča je trenutno v izvajanju.

Primer pravilne zahteve:
`https://ucilnica.fri.uni-lj.si/login/token.php?service=fri_mobile_app&username=ab1234@student.uni-lj.si&password=MojeGeslo123!?`

Pojasnilo zahteve po delih:

- `https://ucilnica.fri.uni-lj.si` - Osnovni naslov učilnice FRI
- `/login/token.php` - Relativni naslov do prijavne skripte
- `service=fri_mobile_app` - Kratko ime storitve za katero se prijavljamo
- `username=ab1234@student.uni-lj.si` - Uporabniško ime
- `password=MojeGeslo123!?` - Uporabniško geslo

Žeton (angl. *token*) je naključno, edinstveno 512-bitno število v šestnajstiškem zapisu. Kot niz je dolg 32 znakov. Uporablja se pri vsakem klicu funkcije iz storitve. Pomembno je poudariti, da je žeton namenjen točno določenemu uporabniku za točno določeno storitev. Isti uporabnik potrebuje dva različna žetona, če želi koristiti funkcije dveh različnih storitev. Ob prvi avtentifikaciji Moodle uporabniku določi in pošlje nov žeton. Pri vseh nadaljnjih pa mu posreduje že obstoječega.

Primer odgovora na zahtevo:


```
{
  "token": "ba8800cb5d32848cf9778a7c1478877f"
}
```

3.4.5 Klici funkcij

Komunikacija s spletnimi storitvami lahko poteka na štiri različne načine: AMF[7], REST[8], SOAP[9] in XML-RPC[10]. Za mobilno aplikacijo je bil na podlagi popularnosti, enostavnosti in dobre kompatibilnosti z Javo izbran način REST. REST je način programske arhitekture, ki se vedno pogosteje uporablja za spletne storitve. Popolna implementacija arhitekture REST se imenuje *RESTful*, ki pa je Moodle ne pozna. Namesto tega ponuja relativni naslov `/webservice/rest/server.php` preko katerega z zahtevami GET kličemo funkcije spletnih storitev. Vse podatke podajamo izključno s parametri GET.

Sledeči parametri so obvezni pri vsakem klicu:

1. `wstoken` - Žeton.
2. `wsfunction` - Ime funkcije, ki jo kličemo.
3. `moodlewsrestformat` - Oblika, v kateri želimo odgovor.

Ostali parametri so odvisni od funkcije, ki jo kličemo. Oglejmo si primer klica funkcije `core_enrol_get_enrolled_users`, ki vrača seznam uporabnikov vpisanih pri predmetu. Predmet podamo z obveznim parametrom `courseid`. Pričakovan argument je celo število, ki predstavlja identifikacijsko številko predmeta. Odgovor vedno pričakujemo v obliki JSON. Če zahteva ne uspe potem kot odgovor dobimo ime napake in njeno sporočilo, prav tako v obliki JSON.

Primer klica funkcije `core_enrol_get_enrolled_users`:

```
https://ucilnica.uni-lj.si/webservice/rest/server.php?courseid=274
&wstoken=ba8800cb5d32848cf9778a7c1478877f&moodlewsrestformat=json
&wsfunction=core_enrol_get_enrolled_users
```

Koda 3.8: Primer odgovora uspešne zahteve

```
[
```

[7] Action Message Format
[8] Representational State Transfer
[9] Simple Object Access Protocol
[10] XML Remote Procedure Call

```
{
  "id": 3,
  "username": "ab9699@student.uni-lj.si",
  "fullname": "Andrej Belcijan",
  "email": "ab9699@student.uni-lj.si",
  "department": "",
  "firstaccess": 1340925516,
  "lastaccess": 1345285604,
  "description": "",
  "descriptionformat": 1,
  "city": "Ljubljana",
  "country": "SI",
  "profileimageurlsmall":
    "https:\/\/ucilnica.fri.uni-lj.si\/
     pluginfile.php\/20\/user\/icon\/f2",
  "profileimageurl":
    "https:\/\/ucilnica.fri.uni-lj.si\/
     pluginfile.php\/20\/user\/icon\/f1",
  "roles": [
    {
      "roleid": 5,
      "name": "Student",
      "shortname": "student",
      "sortorder": 0
    }
  ],
  "preferences": [
    {
      "name": "auth_forcepasswordchange",
      "value": "0"
    },
```

```
        {
          "name": "email_bounce_count",
          "value": "1"
        },
        {
          "name": "email_send_count",
          "value": "6"
        },
        {
          "name": "_lastloaded",
          "value": 1345285637
        }
      ],
      "enrolledcourses": [
        {
          "id": 274,
          "fullname": "Diplomsko delo",
          "shortname": "DD1"
        }
      ]
    }
]
```


Koda 3.9: Primer odgovora neuspešne zahteve


```
{
  "exception": "webservice_access_exception",
  "errorcode": "accessexception",
  "message": "Access control exception"
}
```


Seznam funkcij spletnih storitev, ki jih uporabljamo je sledeč:

- core_course_get_contents

Parametri: `courseid = int`

- core_enrol_get_users_courses
  Parametri: `userid = int`
- core_enrol_get_enrolled_users
  Parametri: `courseid = int`
- core_webservice_get_site_info
  Parametri: `/`
- core_user_get_users_by_id
  Parametri: `userids[0] = int`
- core_message_send_instant_messages
  Parametri:
  - `messages[0][touserid] = int`
  - `messages[0][text] = string`
  - `messages[0][textformat] = int`
  - `messages[0][clientmsgid] = string`
- local_fri_export_ical
  Parametri: `/`

Za komunikacijo s spletnimi storitvami Moodle smo razširili poprej opisan razred *Communicator* v novega z imenom *MoodleCommunicator*. Njegov namen je avtentifikacija (pridobitev žetona), pridobitev osnovnih podatkov o uporabniškem računu in klici vseh podprtih funkcij, ki jih omogoča storitev. Vsebuje tudi nize, ki predstavljajo vse potrebne naslove URL, ime storitve in druge pomembnejše podatke. Zaradi varnosti so določeni programsko.

Funkcije storitev v razredu *MoodleCommunicator* imajo enaka imena in parametre kot jih predvideva Moodle. Za klice funkcij se *MoodleCommunicator* poslužuje metode GET, definirane v nad-razredu *Communicator*. Pridobljen odgovor s strežnika Moodle je sicer v obliki JSON a za naš program le navaden niz. Zato uporabimo Javina razreda *JSONArray* in *JSONObject*, ki zanesljivo iz niza ustvarita objekte. Prvi ustvari polje JSON, drugi pa objekt JSON. Problem pri njuni uporabi je, da moramo vnaprej vedeti ali niz predstavlja polje ali objekt. To najlažje ugotovimo tako, da preverimo

prvi znak niza. Če je to oglati oklepaj ("["), gre najverjetneje za polje, če je zaviti oklepaj ("{") potem imamo opravka z objektom. Moodle na klice funkcij spletnih storitev vedno odgovori s poljem JSON. Z objektom JSON odgovori samo v primeru, da je prišlo do napake. Tako lahko hitro preverimo ali je zahteva uspela. Za vse to skrbi metoda `execute()`, ki jo vsaka s svojimi argumenti kličejo metode za izvajanje funkcij spletnih storitev. Prikazana je s kodo 3.10.

Koda 3.10: Metoda execute() razreda MoodleCommunicator.

```
private JSONArray execute(String function,
    Map<String,String> data) {
  if (data == null) { // Če nimamo strukture s
      podatki, ustvari nov objekt HashMap.
    data = new HashMap<String,String>();
  }
  data.put("wstoken",this.token); // Žeton
  data.put("wsfunction",function); // Ime funkcije
  data.put("moodlewsrestformat",this.moodlewsrestformat);
      // Format odgovora
  try {
    String response = GET(this.url+this.url_rest,data);
    if (response.length() <= 0) {
      // Vsebina je prazna.
      Integer rmsg =
          R.string.server_returned_blank_response;
      String msg = context.getString(rmsg);
      throw new RuntimeException(msg);
    } else if (response.charAt(0) == '{') {
      // Dobili smo objekt JSON. Verjetno je Moodlov
          odgovor napaka.
      JSONObject jo = new JSONObject(response);
      // Kadar objekt vsebuje "key_errorcode", je
```

```
            Moodlov odgovor napaka.
        if (jo.has(key_errorcode)) {
          String errorcode = jo.getString(key_errorcode);
          String message = jo.has(key_message) ?
              jo.getString(key_message) : "";
          if (errorcode.equals("accessexception") ||
              errorcode.equals("nopermissions"))
            throw new AccessControlException(message);
          else
            throw new RuntimeException(message);
        }
        // Pretvori JSONObject v JSONArray
        JSONArray ja = new JSONArray();
        ja.put(jo);
        return ja;
      } else if (response.charAt(0) != '[') {
        // Če odgovor ni ne objekt ne polje JSON, je
            nekaj narobe.
        Integer rmsg =
            R.string.returned_content_not_json;
        String msg = context.getString(rmsg);
        throw new RuntimeException(msg);
      }
      return new JSONArray(response);
    } catch (JSONException e) {
      e.printStackTrace();
      return null;
    } catch (IOException e) {
      e.printStackTrace();
      return null;
    }
```

}

3.4.6 MoodleActivity

Medtem ko sinhronizacija koledarja v ozadju poteka nemoteno, je uporabniški vmesnik Moodle vidni del naše aplikacije. Sestavljen je iz štirih aktivnosti:

- *MoodleCourses* - Seznam predmetov.
- *MoodleCourse* - Vpogled v predmet.
- *MoodleClassmates* - Seznam sošolcev pri predmetu.
- *MoodleClassmate* - Vpogled v profil sošolca.

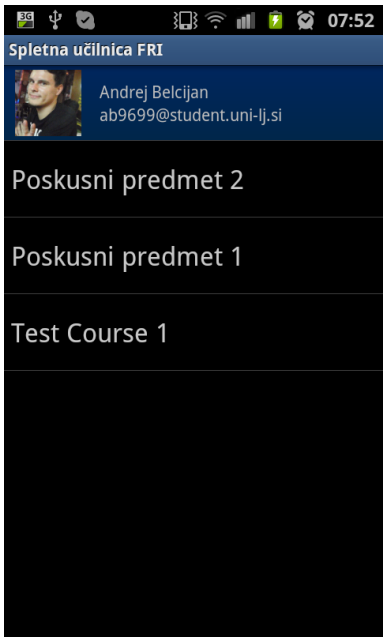
Vse so razširjene iz osnovne aktivnosti *MoodleActivity*, ki je sama razširitev razreda *Activity*. To smo storili zaradi konsistence in poenostavitve nekaterih ponavljajočih se operacij. Želeli smo, da je profil uporabnika, ki je prijavljen, viden na vrhu vsake aktivnosti. Za izris profila skrbi kar razred *MoodleActivity*. Razred ima deklarirano abstraktno metodo `getContent()`, ki jo mora obvezno implementirati vsaka izmed naštetih aktivnosti. Poleg izrisa profila in vsebine poskrbi, da se vedno, ko čakamo, da nam strežnik posreduje podatke, na zaslonu prikaže obvestilo "*Prosimo počakajte...*". Vse to zagotovimo s pomočjo razreda *AsyncTask*. Njegova značilnost so tri metode. Metoda `onPreExecute()` vsebuje kodo, ki se izvrši pred začetkom izvajanja niti. Najpomembnejša `onDoInBackground()`, vsebuje kodo, ki se izvrši v vzporedni niti (asinhrono, v ozadju). Metoda `onPostExecute()` se izvede, ko se vzporedna nit zaključi. Pomembno je vedeti, da v metodi `onDoInBackground()` nimamo dostopa do uporabniškega vmesnika. Tudi če pride do napake, jo moramo shraniti v privatno spremenljivko. Opozorilo o napaki lahko uporabniku prikažemo šele v `onPostExecute()`. Oglejmo si kako aktivnost *MoodleActivity* ustvari objekt *AsyncTask*, prikazan s kodo 3.11.

Koda 3.11: Objekt AsyncTask in njegova izvršitev.

```
new AsyncTask<Void, Void, Void>() {
  private Exception exception = null;
```

```
  protected void onPreExecute() {
    // Prikaži obvestilo "Prosimo počakajte..."
    content_loading_dialog.show();
  }
  protected Void doInBackground(Void... params) {
    try {
      mc = new MoodleCommunicator(self,a);
      content = getContent(); // Pridobi vsebino
          aktivnosti
    } catch (Exception e) {
      exception = e; // Shrani napako
      e.printStackTrace();
    }
    return null;
  }
  protected void onPostExecute(Void result) {
    content_loading_dialog.dismiss();
    if (exception == null) { // Če ni bilo napake
      renderHeader(mc); // Izriši profil uporabnika
      ll.addView(content); // Pod profil dodaj vsebino
      onReady(); // Kliči neobvezno metodo onReady();
    } else { // Prikaži napako in zapusti aktivnost
      Toast.makeText(getBaseContext(),
          exception.getLocalizedMessage(),
          Toast.LENGTH_LONG).show();
      finish();
    }
  }
}.execute();
```

Aktivnost *MoodleCourses* seznam predmetov pridobi od storitvene funkcije `core_enrol_get_users_courses(userid)`, ki vrne podatke o predmetih, v

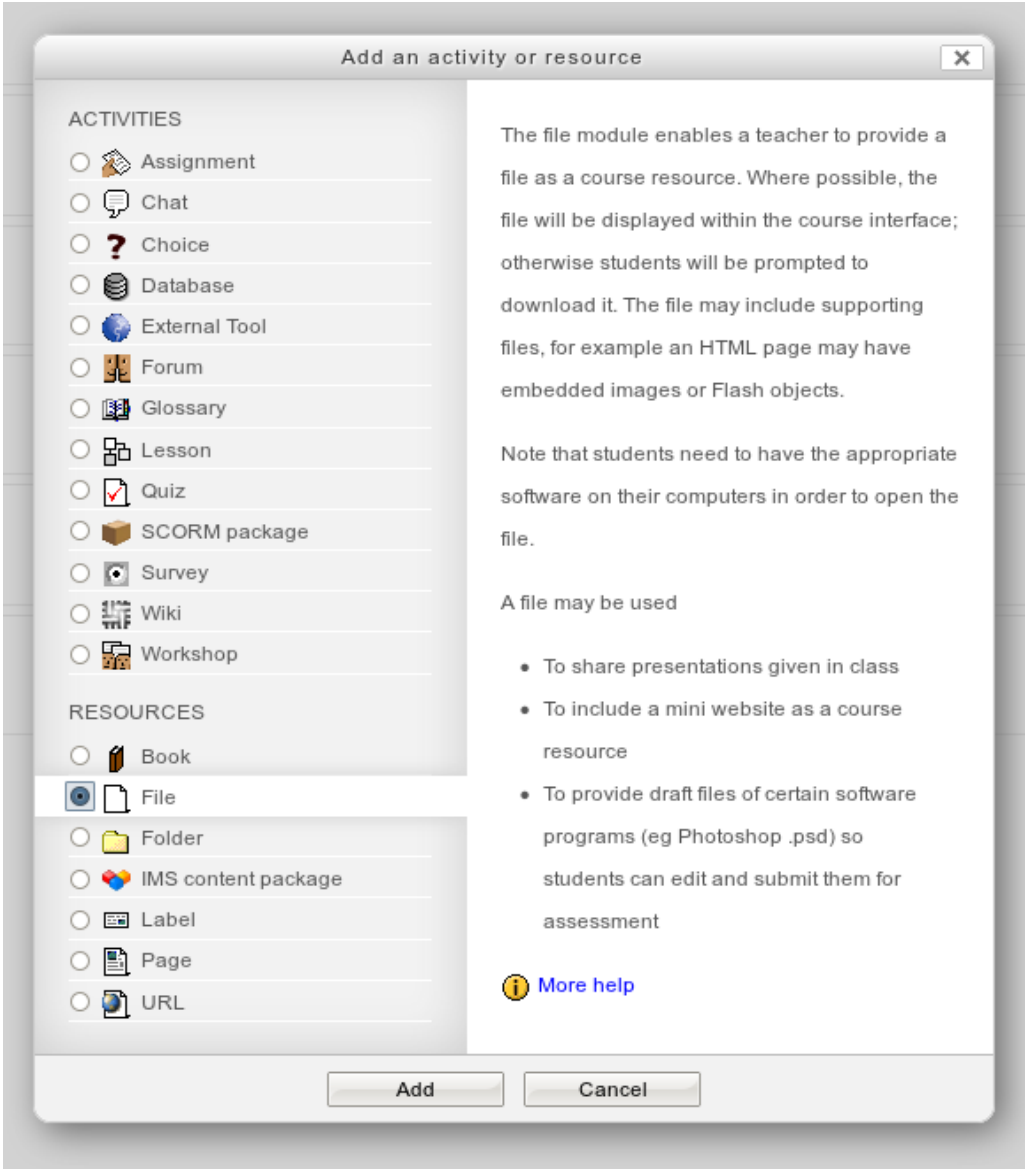


Slika 3.8: Aktivnost MoodleCoursesActivity s seznamom predmetov v katere smo vpisani.

katere je vpisan uporabnik s številko podano z argumentom `userid`. Vsebino beremo s pomočjo razredov *JSONArray* in *JSONObject*. Sproti shranjujemo tudi številke predmetov, ki jih posredujemo naslednji aktivnosti, ko uporabnik izbere enega izmed njih. Izgled aktivnosti *MoodleCourses* je zelo preprost, kar je razvidno iz slike 3.8.

Ko uporabnik izbere predmet aplikacija, preklopi na aktivnost *MoodleCourse*, ki je zadolžena za prikaz vsebine predmeta. Za pridobitev podatkov uporabi funkcijo `core_course_get_contents(courseid)`, ki vrne vsebino predmeta združeno po tednih od začetka do konca semestra. V vsakem tednu so lahko različni moduli, ki jih aktivnost prikaže. Module nastavi učitelj na portalu Moodle. Na izbiro ima vse, ki so prikazani na sliki 3.9.

Za osnovo seznama smo uporabili razred *ListView*. Gre za vsebovalnik, ki prikazuje poljuben seznam elementov. Prikazati smo želeli dve vrsti elementov: naslove tednov in module. *ListView* vsebino črpa iz objekta tipa *Adapter*. Razširili smo razred *ArrayAdapter* in ga poimenovali *EntryAdapter*,



Slika 3.9: Okno za dodajanje modulov v portalu Moodle.

ki mu ob inicializaciji posredujemo seznam objektov Item, pridobljen iz podatkov JSON. Metoda `getView()` v razredu *EntryAdapter* ustvari posamezni objekt *View* glede na tip. Ustvarili smo dva razreda, ki implementirata vmesnik *Item*: *EntryItem* in *SectionItem*. Prvi prikazuje module drugi pa imena tednov. Obema smo morali najprej definirati obliko (angl. layout) uporabo datotek XML.

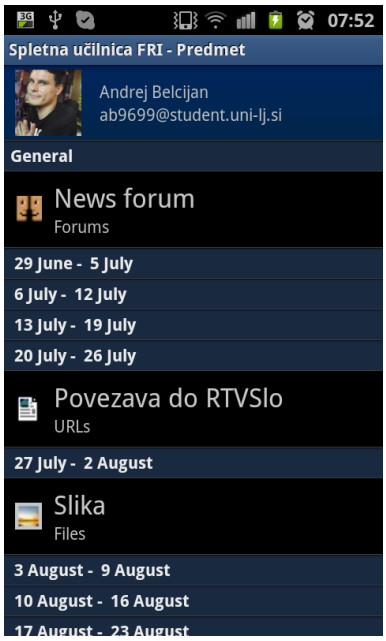
Na seznamu je možno izbrati katerikoli modul. Ob izbiri aktivnost uporabnika preusmeri na spletno stran učilnice, kjer se nahaja vsebina modula. Modul lahko vsebuje tudi povezavo URL do naložene datoteke ali neke določene spletne strani. V takšnem primeru aktivnost uporabnika namesto na učilnico preusmeri neposredno do navedene povezave. V objektu JSON, ki predstavlja teden, se podatek o povezavi skriva v polju `contents` pod ključem `fileurl`, kot je to razvidno iz primera:

```
{
  "id": 6,
  "name": " 20 July - 26 July",
  "visible": 1,
  "summary": "",
  "summaryformat": 1,
  "modules": [
    {
      "id": 8,
      "url": "https:\/\/ucilnica.fri.uni-lj.si\/
              moodle\/mod\/url\/view.php?id=8",
      "name": "Povezava do RTVSlo",
      "visible": 1,
      "modicon": "http:\/\/ucilnica.fri.uni-lj.si\/
                  moodle\/theme\/image.php\/
                  standard\/core\/1345117490\/f\/html",
      "modname": "url",
      "modplural": "URLs",
```

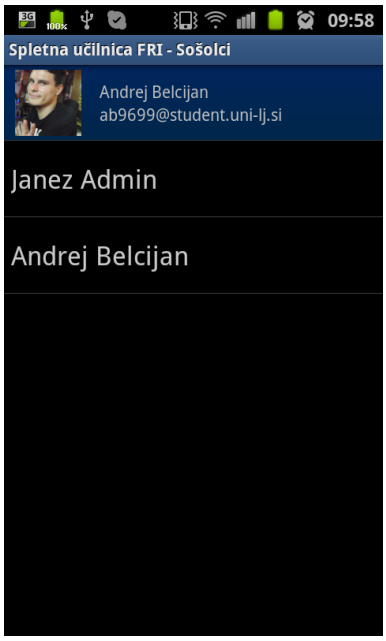
```
      "availablefrom": 0,
      "availableuntil": 0,
      "indent": 0,
      "contents": [
        {
          "type": "url",
          "filename": null,
          "filepath": null,
          "filesize": 0,
          "fileurl": "http:\/\/www.rtvslo.si\/",
          "timecreated": null,
          "timemodified": null,
          "sortorder": null,
          "userid": null,
          "author": null,
          "license": null
        }
      ]
    }
  ]
}
```


Primer končnega izgleda aktivnosti *CourseActivity* je viden na sliki 3.10. Dodatna storitev aktivnosti je preklop na seznam sošolcev, ki so vpisani v izbran predmet. Seznam je dostopen preko menija. Ob izbiri mu aktivnost posreduje številko predmeta.

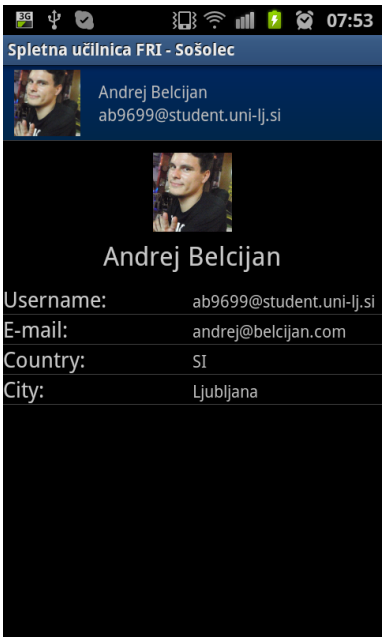
Seznam sošolcev je spet zelo preprosta aktivnost, ki v element tipa *ListView* napolni polna imena uporabnikov. Podatke iz Moodla pridobi po funkciji `core_enrol_get_enrolled_users(courseid)`. Primer aktivnosti je razviden iz slike 3.11. Ob izbiri sošolca aktivnost njegovo številko posreduje aktivnosti *ClassmateActivity*.



Slika 3.10: Aktivnost CourseActivity.



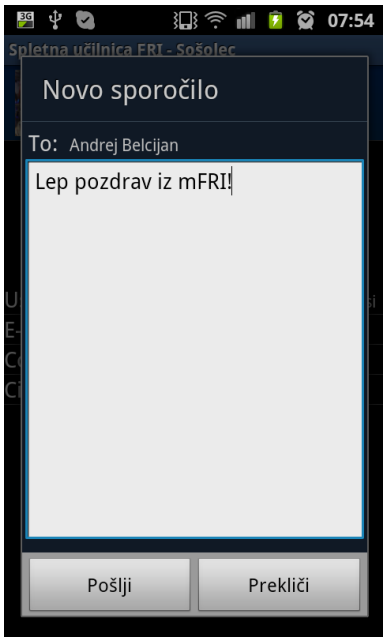
Slika 3.11: Aktivnost ClassmatesActivity s seznamom sošolcev.



Slika 3.12: Aktivnost ClassmateActivity s podatki sošolca.

Zadnja aktivnost je namenjena natančnejšemu prikazu profila sošolca, ki smo ga prejeli od prejšnje aktivnosti. Zasnovana je tako, da lahko prikaže profil kateregakoli uporabnika iz strežnika Moodle, če imamo do njegovih podatkov dostop. To je razvidno iz slike 3.12. Podatke iz Moodla pridobi s klicem funkcije `core_user_get_users_by_id(userids[])`. Funkcija namesto ene številke sprejme polje številk uporabnikov in nato vrne seznam vseh. V aktivnosti *ClassmateActivity* potrebujemo le enega uporabnika, zato kot argument podamo polje števil z enim elementom.

Dodatna storitev aktivnosti *ClassmateActivity* je možnost pošiljanja sporočila sošolcu. Storitev je dostopna preko menija. Namesto nove aktivnosti tokrat odpremo manjše okno v katerega je mogoče napisati sporočilo, razvidno iz slike 3.13. Po pritisku na tipko pošlji aktivnost s pomočjo razreda *AsyncTask* v ozadju kliče Moodlovo funkcijo `core_message_send_instant_messages()`. Ko se pošiljanje zaključi, dobimo obvestilo o uspehu, okno s sporočilom pa izgine. Na portalu Moodle lahko uporabnik, kateremu je bilo poslano sporočilo, to sporočilo takoj tudi prebere. Žal Moodle še ne omogoča preje-



Slika 3.13: Aktivnost ClassmateActivity z dialogom za novo sporočilo.

manja sporočil preko spletnih storitev,zato moramo odgovor prebrati preko spletnega vmesnika.

3.4.7 Uporabniški računi

Žeton in druge podatke o uporabniškem računu je najlažje hraniti s pomočjo razreda *SharedPreferences*, a obstaja boljša rešitev. Android omogoča razvijalcem mobilnih aplikacij dostop do sistemskega nadzornika uporabniških računov. Ta način poleg shranjevanja vseh potrebnih podatkov o uporabniškem računu prinaša mnoge koristi. Zato smo se odločili ustvariti globalni račun za učilnico FRI. Postopek je nekoliko zapleten, a kljub temu razumljiv. Obliko računa definiramo s pomočjo datoteke XML, ki jo shranimo v imenik `xml/`. V našem primeru smo jo poimenovali `moodle_authenticator.xml`. Njena vsebina določa podrobnosti o računu kot: ime, ikone različnih velikosti, dodatne nastavitve in pa najpomembneje unikatno ime vrste računa (`accountType`). Vsebina datoteke je sledeča:

```
<account-authenticator
    xmlns:android="http://schemas.android.com/apk/res/android"
   android:accountType="org.unilj.fri.moodle"
   android:icon="@drawable/ic_launcher"
   android:smallIcon="@drawable/ic_launcher"
   android:label="@string/authenticator_name_moodle"
   android:accountPreferences=
     "@xml/moodle_account_preferences" />
```

Nato moramo ustvariti posebno avtentifikacijsko storitev. Pred tem jo najavimo v datoteki `AndroidManifest.xml`. Hkrati podamo tudi vir do pravkar ustvarjene datoteke s podatki o obliki našega računa. Vse skupaj izgleda sledeče:

```
<service android:name=
  "org.unilj.fri.moodle.MoodleAuthenticatorService">
<intent-filter>
  <action
      android:name="android.accounts.AccountAuthenticator"
      />
</intent-filter>
<meta-data
    android:name="android.accounts.AccountAuthenticator"
    android:resource="@xml/moodle_authenticator" />
</service>
```

Naša aplikacija bo imela dostop do upravljalnika z uporabniškimi računi. Za to potrebujemo štiri različna dovoljenja s strani uporabnika Androida. Napovemo jih v glavnem elementu `manifest` v datoteki `AndroidManifest.xml` na sledeč način:

```
<uses-permission
    android:name="android.permission.AUTHENTICATE_ACCOUNTS"
    />
```

```
<uses-permission
    android:name="android.permission.USE_CREDENTIALS" />
<uses-permission
    android:name="android.permission.GET_ACCOUNTS" />
<uses-permission
    android:name="android.permission.MANAGE_ACCOUNTS" />
```

Sedaj je vse pripravljeno za našo avtentifikacijsko storitev. Ta je zelo preprosta, a se mora držati nekaj strogih pravil. Biti mora vezana (angl. *bound*) storitev tipa *Service*. Imeti mora obvezno redefinirano metodo `onBind()`. Metoda 3.12 vrne *IBinder* objekta *MoodleAccountAuthenticator*, ki je razširitev razreda *AbstractAccountAuthenticator*.

Koda 3.12: Metoda onBind() v razredu Service.

```
public IBinder onBind(Intent intent) {
  IBinder ret = null;
  String action =
     android.accounts.AccountManager.ACTION_AUTHENTICATOR_INTENT;
  if (intent.getAction().equals(action))
    ret = getAuthenticator().getIBinder();
  return ret;
}
```

V naslednjem koraku ustvarimo naš razred *MoodleAccountAuthenticator*. Vsebuje metode, ki jih kliče sistem Android ob dogodkih kot je dodajanje novega računa. Ta se imenuje `addAccount()` in je edina, ki jo potrebujemo. Prikazana je s kodo 3.13. Ostale metode definiramo po želji ali pa z njimi samo vrnemo vrednost `null`.

Koda 3.13: Metoda addAccount razreda MoodleAccountAuthenticator.

```
public Bundle addAccount(AccountAuthenticatorResponse
    response, String accountType, String authTokenType,
    String[] requiredFeatures, Bundle options) throws
    NetworkErrorException {
```

```
    if (!accountType.equals(ACCOUNT_TYPE)) return null;
    final Intent intent = new Intent(this.mContext,
        MoodleLoginActivity.class);
    intent.putExtra(KEY_AUTHTOKEN_TYPE, authTokenType);
    intent.putExtra(AccountManager.KEY_ACCOUNT_TYPE,
        ACCOUNT_TYPE);
    intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE,
        response);
    final Bundle result = new Bundle();
    result.putParcelable(AccountManager.KEY_INTENT,
        intent);
    return result;
}
```

Kot je razvidno iz programske kode, metoda požene aktivnost *MoodleLogin* od katere pričakuje odgovor v obliki objekta *Bundle*, ki ga aktivnost tudi vrne. Pred tem aktivnosti posredujemo 3 podatke, in sicer:

- `authTokenType` - Unikaten niz, ki predstavlja tip žetona.
- `ACCOUNT_TYPE` - Unikaten niz, ki predstavlja tip. uporabniškega računa.
- `response` - Odgovor sistemskega avtentikatorja računov.

MoodleLogin je posebna aktivnost namenjena avtentifikaciji uporabniških računov. Je razširitev razreda *AccountAuthenticatorActivity*. V osnovi predstavlja le zaslon za vnos uporabniškega imena in gesla, a ima veliko odgovornost. Poskrbeti mora za preverjanje računa na strežniku Moodle. To stori s pomočjo razreda *MoodleCommunicator*. Če strežnik avtentifikacijo potrdi in nam vrne žeton, mora nato aktivnost v Androidov upravljalnik z uporabniškimi računi dodati nov račun ter vanj shraniti žeton in vse ostale pridobljene podatke. Operacija je asinhrona, saj moramo počakati na odgovor strežnika, zato smo pri tem uporabili razred *AsyncTask*. Del kode brez uporabe *AsyncTask* bi izgledal takole:

```
accMgr = AccountManager.get(MoodleLoginActivity.this);
account = new Account(username, accountType);
```

```
MoodleCommunicator mc = new
    MoodleCommunicator(MoodleLoginActivity.this, account, username, pass
authToken = mc.getToken(); // Od strežnika Moodle
    pridobi žeton
userData = mc.acquireUserData(); // Od strežnika
    Moodle pridobi podatke o uporabniškem računu
Boolean accountCreated =
    accMgr.addAccountExplicitly(account, password,
    userData); // Dodaj novi račun v Android
accMgr.setAuthToken(account,
    MoodleAccountAuthenticator.AUTHTOKEN_TYPE,
    authToken); // Shrani žeton!

if (accountCreated) {
  final Intent intent = new Intent();
  intent.putExtra(AccountManager.KEY_ACCOUNT_NAME,
     username); // Vrni uporabniško ime
  intent.putExtra(AccountManager.KEY_ACCOUNT_TYPE,
     MoodleAccountAuthenticator.ACCOUNT_TYPE); // Vrni
     tip uporabniškega računa
  intent.putExtra(MoodleAccountAuthenticator.KEY_AUTHTOKEN_TYPE,
     MoodleAccountAuthenticator.AUTHTOKEN_TYPE); //
     Vrni tip žetona
  intent.putExtra(AccountManager.KEY_AUTHTOKEN,
     authToken); // Vrni žeton
  setAccountAuthenticatorResult(intent.getExtras());
     // Pošlji odgovor sistemskemu avtentikatorju
  setResult(RESULT_OK, intent); // Sporoči, da je
     rezultat operacije uspešen
  Toast.makeText(getBaseContext(),
     R.string.moodle_account_add_successful,
```

```
        Toast.LENGTH_LONG).show();
    finish();
}
```

Sedaj lahko v nastavitvah Androida dodajamo in odstranjujemo uporabniške račune učilnice FRI. Vsak izmed njih hrani svoj žeton in svoje podatke. Po prvi uspešni avtentifikaciji bo objekt našega razreda *MoodleCommunicator* žeton in podatke o uporabniškem računu pridobil od Androidovega upravljalnika z uporabniškimi računi.

3.4.8 Sinhronizacija

Upravljalnik z uporabniškimi računi omogoča tudi sinhronizacijo. Sinhroniziramo lahko praktično karkoli, kontakte, sporočila, koledarje, datoteke... Za naše potrebe je to več kot idealno. Želeli bi enosmerno sinhronizirati koledar dogodkov na portalu Moodle s koledarjem v mobilni napravi. V ta namen moramo implementirati še storitev za sinhronizacijo. Tako kot avtentifikacijsko storitev moramo tudi to najprej napovedati v aplikacijski datoteki `AndroidManifest.xml`. V našem primeru izgleda sledeče:

```
<service
    android:name="org.unilj.fri.moodle.MoodleSyncAdapterService"
    android:exported="true">
  <intent-filter>
    <action android:name="android.content.SyncAdapter"
        />
  </intent-filter>
  <meta-data
      android:name="android.content.SyncAdapter"
      android:resource="@xml/moodle_syncadapter_calendar"
      />
</service>
```

Tudi tokrat potrebujemo štiri dovoljenja s strani uporabnika. Dovoljenja predvidevajo pisanje in branje koledarjev ter pisanje in branje sinhronizacijskih nastavitev. Zahteve za dovoljenja v `AndroidManifest.xml` so sledeče:

```
<uses-permission
    android:name="android.permission.WRITE_CALENDAR" />
<uses-permission
    android:name="android.permission.READ_CALENDAR" />
<uses-permission
    android:name="android.permission.WRITE_SYNC_SETTINGS"
    />
<uses-permission
    android:name="android.permission.READ_SYNC_SETTINGS"
    />
```

Nato ustvarimo našo storitev, ki je razširitev razreda *Service*. Poimenovali smo jo *MoodleSyncAdapterService*. Njena obvezna in edina metoda, prikazana s kodo 3.14, `onBind()` vrne *IBinder* razreda *MoodleThreadedSyncAdapter*.

Koda 3.14: Metoda onBind() razreda MoodleSyncAdapterService.

```
public IBinder onBind(Intent intent) {
  this.sCalendarBuilder = new CalendarBuilder(); //
     Ustvari objekt CalendarBuilder
  return getSyncAdapter().getSyncAdapterBinder(); //
     Vrni IBinder razreda MoodleThreadedSyncAdapter
}
```

Metoda `getSyncAdapter()` enostavno vrne statični objekt razreda *MoodleThreadedSyncAdapter*. To je razred, ki je razširitev razreda *AbstractThreadedSyncAdapter*. Ima le eno obvezno metodo `onPerformSync()`, ki jo kliče sistem Android vsakič, ko se odloči, da je čas za sinhronizacijo. Metoda 3.15 opravi več pomembnih korakov.

Koda 3.15: Metoda onPerformSync razreda MoodleThreadedSyncAdapter.

```
public void onPerformSync(Account account, Bundle
    extras, String authority, ContentProviderClient
    provider, SyncResult syncResult) {
  mContentResolver = mContext.getContentResolver();

  // Poročaj o sinhronizaciji
  Log.i(TAG, "Synchronizing calendars: " +
      account.toString());

  // Pridobi naslov podatkovne baze
  URI_CALENDAR = getCalendarUriBase();
  URI_CALENDARS = Uri.parse(URI_CALENDAR +
      "calendars");
  URI_EVENTS = Uri.parse(URI_CALENDAR + "events");

  try {
    // Pridobi številko lokalnega koledarja
    Integer calId = getLocalCalendar(account);
    // Odstrani vse dogodke v koledarju
    Integer wipedEntries = wipeCalendar(calId);

    Log.i(TAG, "Cleared " + wipedEntries + " calendar
        entries.");

    MoodleCommunicator mc = new
        MoodleCommunicator(mContext, account);
    // Pridobi koledar iz Moodla
    InputStream is = mc.local_fri_export_ical();
    // Sestavi model koledarja
```

```
    Calendar cal =
        MoodleSyncAdapterService.sCalendarBuilder.build(is);
    // Iz modela koledarja pridobi seznam komponent
    ComponentList cl = cal.getComponents();

    Iterator<Component> ic = cl.iterator();
    // Sprehodi se po komponentah koledarja
    while (ic.hasNext()) {
      Component c = (Component) ic.next();
          if (c instanceof VEvent) { // Če je
              komponenta tipa VEvent,
        addEvent(calId,(VEvent)c); // dodaj dogodek.
      }
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

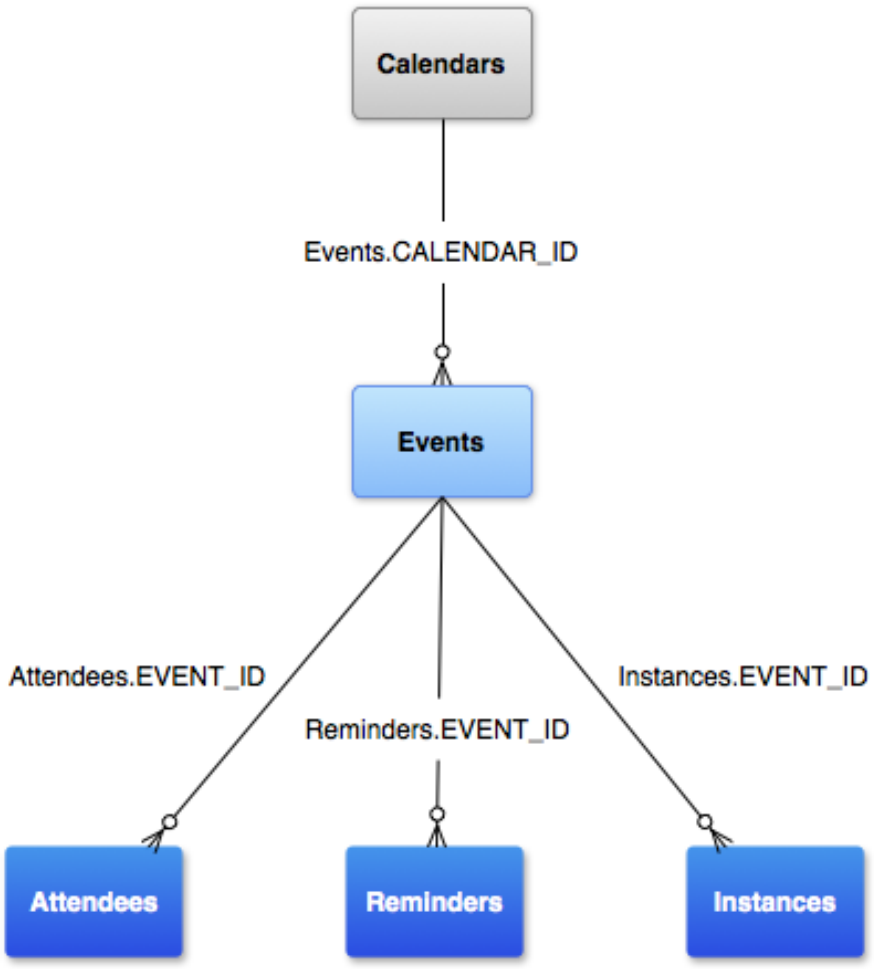
Prvi korak je pridobitev naslovov *Uri*. To so Androidovi notranji naslovi, ki jih uporablja za dostop do notranjih vsebin, v našem primeru podatkovne baze. Podatkovni model baze koledarjev in dogodkov v Androidu je prikazan na sliki 3.14. Težava je v tem, da se naslova pred in po različici Androida 2.2 razlikujeta. Če želimo, da naša mobilna aplikacija deluje na vseh različicah moramo ugotoviti kateri naslov je pravi. Razlika je sledeča:

- Android 2.1 in prej: "`content://calendar/`"
- Android 2.2 in kasneje: "`content://com.android.calendar/`"

Metodo 3.16 za ugotavljanje pravega naslova smo našli na spletni strani StackOverflow in jo nekoliko izboljšali. [6] Pomembno je, da jo kličemo šele ob sinhronizaciji, saj v konstruktorju razreda ne bo delovala pravilno.

Koda 3.16: Metoda getCalendarUriBase().

```
private String getCalendarUriBase() {
```



Slika 3.14: Podatkovni model Androidove notranje podatkovne baze koledarjev in dogodkov. [5]

```
  String calendarUriBase = null;
  Cursor managedCursor = null;
  Uri calendars;
  calendars = Uri.parse(
     "content://com.android.calendar/calendars" );
  try {
       managedCursor =
          mContentResolver.query(calendars, null, null,
          null, null);
       if (managedCursor != null) return
          "content://com.android.calendar/";
  } catch (Exception e) { }
  calendars =
     Uri.parse("content://calendar/calendars");
  try {
       managedCursor =
          mContentResolver.query(calendars, null, null,
          null, null);
       if (managedCursor != null) return
          "content://calendar/";
  } catch (Exception e) { }
  return calendarUriBase;
}
```

Ob sinhronizaciji preverimo ali koledar, v katerega bi lahko vpisovali dogodke, že obstaja. Če še ne, ga moramo dodati. Gre za lokalni koledar, do katerega ima dostop samo naša aplikacija. Vidijo ga lahko morebitne koledarske aplikacije, ki so nameščene na napravi. Poudariti je treba, da nekatere namestitve sistema Android ne vključujejo koledarske aplikacije. Nov koledar v bazo dodamo na način prikazan s kodo 3.17.

Koda 3.17: Koda za dodajanje novega koledarja.

```
ContentValues values = new ContentValues();
```

```
values.put( "displayName",
    mContext.getString(R.string.calendar_name_fri) );
values.put( "_sync_account", username );
values.put( "_sync_account_type",
    MoodleAccountAuthenticator.ACCOUNT_TYPE );
mContentResolver.insert(
    asSyncAdapter(URI_CALENDARS, username), values );
```

V vsakem primeru moramo pridobiti številko koledarja, kot je zapisana v sistemski bazi podatkov. Poenostavljen postopek pridobitve številke koledarja (angl. *calendar identification*) glede na uporabniško ime (stolpec "_sync_account") in tip uporabniškega računa (stolpec "_sync_account_name") je prikazan s kodo 3.18.

Koda 3.18: Koda za iskanje številke koledarja.

```
String selection = "(("+"_sync_account" + "=?) AND
    ("+"_sync_account_type"+" =?))";
String[] CALENDAR_PROJECTION = new String[] { "_id",
    "_sync_account", "_sync_account_type",
    "displayName" };
String[] selectionArgs = new String[] { username,
    MoodleAccountAuthenticator.ACCOUNT_TYPE };
Cursor cur = mContentResolver.query( URI_CALENDARS,
    CALENDAR_PROJECTION, selection, selectionArgs, null
    );
cur.moveToFirst();
return cur.getInt(0);
```

3.4.9 Format iCalendar

Preden lahko v sistemski koledar vnesemo vse dogodke, jih moramo seveda pridobiti od strežnika Moodle. To storimo z metodo razreda *MoodleCommunicator* `local_fri_export_ical()`. Od ostalih se razlikuje po tem, da

komponenta	pomen
`VEVENT`	dogodek
`VTODO`	opravilo
`VJOURNAL`	dnevnik
`VFREEBUSY`	zahteva ali odgovor za prost ali zaseden čas
`VTIMEZONE`	časovni pas
`VALARM`	opomnik

Tabela 3.2: Pregled komponent formata iCalendar in njihovih pomenov

namesto v obliki JSON odgovor pošlje v obliki *iCalendar*.

Internet Calendaring and Scheduling Core Object Specification, krajše iCalendar, je odprt standard za prenos in izmenjavo koledarskih podatkov. Napisan je bil leta 2009 na organizaciji IETF[11]. [8] Za uporabo z našo mobilno aplikacijo je zelo primeren, ker ne vsebuje nobenih lastniških omejitev in ker sistem Moodle že vsebuje vsa potrebna orodja za delo z njim. V osnovi je standard dokaj preprost in se uporablja podobno kot XHTML[12], le z drugačno sintakso. Sestavljen je iz dveh vrst objektov, lastnosti in komponent. Lastnosti opisujejo komponente katerim pripadajo. Komponente torej vsebujejo lastnosti ali druge komponente. Vsaka izmed komponent ima glavo in rep. Glava ima predpono `BEGIN:`, ki ji sledi ime komponente. Rep ima predpono `END:`, ki ji prav tako sledi ime pripadajoče komponente. Vsa imena komponent se začnejo z veliko črko "`V`". Glavna komponenta se imenuje `VCALENDAR`. Njena vsebina se imenuje "*icalbody*". Komponente, ki jih lahko vsebuje, so razvidne iz tabele 3.2.

Za našo mobilno aplikacijo so predvsem pomembne komponente tipa `VEVENT`. Predstavljajo vsak dogodek (npr. kolokvij), ki je vnešen v koledar Moodle. Ima lahko različne lastnosti. Najpomembnejše so `SUMMARY` (ime dogodka), `DESCRIPTION` (opis dogodka), `DTSTART` (čas pričetka dogodka) in

[11] Organizacija Internet Engineering Task Force
[12] Extensible HyperText Markup Language

`DTEND` (čas konca dogodka). Primer 3.19 prikazuje vsebino iCalendar s komentarji, ki jo pošlje funkcija Moodle.

Koda 3.19: Primer vsebine iCalendar.

```
BEGIN:VCALENDAR // Začetek komponente VCALENDAR
METHOD:PUBLISH
PRODID:-//John Papaioannou/NONSGML Bennu 0.1//EN //
    Lastnost "PRODID", ki določa opis komponente
    VCALENDAR
VERSION:2.0
BEGIN:VEVENT  // Začetek komponente VEVENT
UID:6@
SUMMARY:Predavanje
DESCRIPTION:Predavanje o Androidu // Lastnost
    "DESCRIPTION", ki določa opis komponente VEVENT
CLASS:PUBLIC
LAST-MODIFIED:20120815T004154Z
DTSTAMP:20120816T224202Z
DTSTART:20120818T004000Z
DTEND:20120818T014000Z
CATEGORIES:AN1
END:VEVENT  // Konec komponente VEVENT
BEGIN:VEVENT // Začetek komponente VEVENT
UID:7@
SUMMARY:Vaje
DESCRIPTION:Programiranje Android
CLASS:PUBLIC
LAST-MODIFIED:20120815T005633Z
DTSTAMP:20120816T224202Z
DTSTART:20120831T005500Z
DTEND:20120831T023500Z
CATEGORIES:AN1
```

```
END:VEVENT   // Konec komponente VEVENT
END:VCALENDAR // Konec komponente VCALENDAR
```

Za ustvarjanje novega dogodka v Moodle moramo določiti najmanj tri obvezne parametre. To so vrsta dogodka (*Type of event*), ime dogodka (*Name*) in čas pričetka dogodka (*Date*), ki je sestavljen iz leta, meseca, dneva, ure in minute.

Moodle pozna tri vrste dogodkov: *User*, *Course* in *Site*. Razlikujejo se v vidljivosti. Prvi so vidni samo uporabniku, ki jih je ustvaril. Drugi so vidni vsem, ki so vpisani v predmet pri katerem so dogodki objavljeni. Tretji so globalni in jih vidijo vsi. Za našo mobilno aplikacijo je vidljivost tako rekoč nepomembna, saj nam Moodle dogodkov, ki nam niso namenjeni, ne posreduje.

Ime dogodka se v obliko *iCalendar* preslika pod lastnost `SUMMARY`, čas pričetka pa pod lastnost `DTSTART`. Čas konca dogodka se preslika pod lastnost `DTEND`. Njen pripadajoč parameter, ki določa konec dogodka ni obvezen. V primeru, da ga ne vnesemo, dobi `DTEND` enako vrednost kot lastnost `DTSTART`.

Vsi ostali parametri so neobvezni tako v Moodle kot v obliki iCalendar, zato se v primeru nedoločenosti preprosto izpustijo. iCalendar podpira tudi nekatere lastnosti, ki jih Moodle ne uporablja. To so na primer lastnosti za določitev lokacije (`LOCATION`) in geografske pozicije (`GEO`) dogodka.

Moodle pri ustvarjanju dogodka omogoča tudi določitev tedenskega ponavljanja dogodka, poleg katerega je potrebno vpisati še število ponovitev. Če izberemo 10 ponovitev, bo Moodle v interni podatkovni bazi ustvaril 10 zaporednih dogodkov z razmikom enega tedna. Zato tudi funkcija za izvoz koledarja prikaže 10 ločenih dogodkov, ki se v osnovi razlikujejo le po času začetka in konca. Moodle bi lahko izvoz optimiziral z uporabo lastnosti `RDATE`, ki v standardu *iCalendar* določa ponavljajoči se dogodek.

Koda PHP, ki koledar Moodle izvozi v obliko iCalendar se nahaja v podmapi `calendar/` in se imenuje `export_execute.php`. Bila je vzeta kot osnova za izdelavo funkcije za izvoz koledarja.

3.4.10 Knjižnica ical4j

Knjižnica *ical4j* je namenjena delu s podatki oblike *iCalendar* v programskem jeziku Java. Omogoča programsko grajenje (angl. *building*) ali razčlenjevanje (angl. *parsing*) zapisa *iCalendar*.

V naši mobilni aplikaciji se uporablja za interpretiranje koledarskih podatkov, ki jih na zahtevo posreduje spletna storitev portala Moodle. Glavni razlog za uporabo *ical4j* je zanesljivost. Standard iCalendar je v celoti zelo obsežen, kljub temu, da beremo samo komponente `VEVENT`.

Pozor, *ical4j* vsebuje razrede, ki niso varni za delo z nitmi (niso *thread-safe*). [7] Primer je uporabljen razred *CalendarBuilder*, ki iz vhodnega toka podatkov zgradi model *iCalendar*. Pri njegovi uporabi moramo biti posebno previdni. Problematičen je zato, ker med izvajanjem prebere vsebino datoteke `ical4j.properties`, ki jo pričakuje v imeniku `src/` projekta Android. Datoteka lahko vsebuje natančnejše nastavitve za delovanje knjižnice *ical4j*. V ta namen objekt CalendarBuilder namesto ob vsaki uporabi statično ustvarimo v razredu *MoodleSyncAdapterService*. Pri tem poskrbimo, da je javno dostopen (angl. *public*), uporabimo pa ga ob vsaki sinhronizaciji koledarja.

Da se dogodki ne bi podvajali ob vsaki sinhronizaciji, iz našega lokalnega koledarja najprej odstranimo vse dogodke. To storimo s preprostim klicem metode delete, ki je del razreda *ContentResolver*, prikazanim s kodo 3.20.

Koda 3.20: Ukaz za izbris vseh dogodkov iz koledarja.

```
cr.delete(URI_EVENTS, "(calendar_id=?)", new
    String[]{calendarId.toString()});
```

Kot smo videli se v metodi `onSync()` sprehodimo preko vseh komponent. Vsako za katero ugotovimo, da je tipa *VEvent* s pomočjo metode `addEvent()` dodamo v sistemski koledar. Pri tem moramo biti posebno pozorni na manjkajoče podatke. Ni na primer nujno, da bo imel vsak dogodek podan opis, zato v takem primeru opisa ne shranjujemo. Postopek dodajanja objekta *VEvent* v sistemski koledar je prikazan s kodo 3.21.

Koda 3.21: Postopek dodajanja objekta VEvent v koledar.

```
// Poišči začetek dogodka
DtStart veStart = event.getStartDate();
// Poišči konec dogodka
DtEnd veEnd = event.getEndDate();
// Poišči naslov dogodka
Summary veSummary = event.getSummary();
// Poišči opis dogodka
Description veDescription = event.getDescription();
// Poišči kraj dogodka
Location veLocation = event.getLocation();

// Ustvari objekt ContentValues, kamor shranimo
   podatke o koledarju
ContentValues values = new ContentValues();
// Dodaj številko koledarja
values.put("calendar_id", calendarId);
// Preveri ali imamo čas začetka dogodka
if (veStart != null) {
  java.util.Calendar cStartDate =
     java.util.Calendar.getInstance();
  cStartDate.setTime(event.getStartDate().getDate());
  // Dodaj čas začetka dogodka
  values.put("dtstart", cStartDate.getTimeInMillis());
}
// Preveri ali imamo čas konca dogodka
if (veEnd != null) {
  java.util.Calendar cEndDate =
     java.util.Calendar.getInstance();
  cEndDate.setTime(event.getEndDate().getDate());
  // Dodaj čas konca dogodka
```

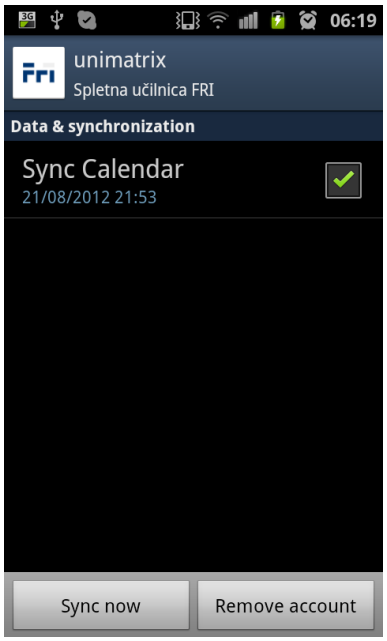
```
    values.put("dtend", cEndDate.getTimeInMillis());
}
// Dodaj naslov dogodka
if (veSummary != null)values.put("title",
    veSummary.getValue());
// Dodaj opis dogodka
if (veDescription != null) values.put("description",
    veDescription.getValue());
// Dodaj kraj dogodka
if (veLocation != null) values.put("eventLocation",
    veLocation.getValue());
// Dodaj podatek o tem, ali je dogodek celodneven
values.put("allDay", isEventAllDay(event) ? 1 : 0);

// Dodaj dogodek v lokalni koledar aplikacije
return mContentResolver.insert(URI_EVENTS, values);
```

Sinhronizacijo moramo za vsak račun posebej vklopiti kot prikazuje slika 3.15. Po privzetih nastavitvah je izklopljena, saj menimo, da te storitve ne bo koristil vsak študent. Če uporabniški račun iz sistema odstranimo, z njim samodejno izbrišemo tudi vse sinhronizirane dogodke v koledarju mobilne naprave.

3.5 Avtobusi

Velik del študentov vsakodnevno na fakulteto prispe z mestnim avtobusom. Radi bi vedeli, kdaj bo na postaji naslednji avtobus. Storitev, ki napoveduje prihode avtobusov na postaje obstaja v obliki spletne strani. Vzdržuje jo podjetje *Telargo d.o.o.*. Stran je vsem dostopna brezplačno. Narejena v dveh izvedbah, ki sta dosegljivi na sledečih naslovih:

1. `http://bus.talktrack.com`
2. `http://wbus.talktrack.com`



Slika 3.15: Nastavitve uporabniškega računa Moodle na Androidu.

Prva je namenjena navadnim spletnim brskalnikom in podatke prikaže na estetski način. Druga je namenjena starejšim mobilnim napravam, z brskalniki WAP[13]. To pomeni, da je izdelana zelo minimalistično, kar je za nas dobro.Potrebujemo namreč le čiste podatke, ki jih bomo prikazali po svoje. Podatke bomo zato črpali iz naslova številke 2. Uporaba v spletnem brskalniku je enostavna. V vnosno polje vpišemo ime avtobusne postaje, ki nas zanima in pritisnemo gumb `Prikaži`. Stran nam kot odgovor pošlje napovedi prihodov avtobusov na izbrani postaji. Kljub vsemu, tudi v tem primeru podatki niso povsem čisti. Poslani so v obliki HTML[14]. V surovi obliki posredovani podatki izgledajo kot prikazuje koda 3.22.

Koda 3.22: Podatki o napovedih prihodov avtobusov v obliki HTML.

```
<html><body>
<form id="Form1" name="Form1" method="post"
    action="wap.aspx?__ufps=080735">
```

[13]Wireless Application Protocol
[14]HyperText Markup Language

```
<input type="hidden" name="__EVENTTARGET" value="">
<input type="hidden" name="__EVENTARGUMENT" value="">
<script language=javascript><!--
function __doPostBack(target, argument){
  var theform = document.Form1
  theform.__EVENTTARGET.value = target
  theform.__EVENTARGUMENT.value = argument
  theform.submit()
}
// -->
</script>
NAPOVED PRIHODOV ZA Jadranska (603012)<br>
1 VIŽMARJE - MESTNI LOG: 14:04, n 14:18, 14:30<br>
NAPOVED PRIHODOV ZA Jadranska (603011)<br>
1 MESTNI LOG - GARAŽA: 22:33<br>
1 N MESTNI LOG - BAVARSKI DVOR - GAMELJNE: 22:15,
   22:46<br>
1 MESTNI LOG - VIŽMARJE: n 14:00, n 14:15, 14:30<br>
Napovedi so bile izračunane ob 14:00.<br>
Časi, označeni s črko n, pomenijo prihode nizkopodnih
   avtobusov, na katere je lažje vstopiti.<br>
<a href="wap.aspx?culture=sl-SI">Nazaj</a><br>
</form></body></html>
```

Podatki, ki nas zanimajo se nahajajo med elementi HTML, ki nas ne zanimajo, zato jih moramo najprej izločiti. To najlažje in najbolj zanesljivo storimo z uporabo knjižnice *jsoup* [10]. Gre za odprtokodni razčlenjevalnik oblike HTML. Deluje na principu DOM[15], ki v pomnilniku zgradi model spletne strani. Za to porabi nekoliko več prostora. To v našem primeru ni tako pomembno, saj je vsebina, ki jo bomo brali, zelo kratka in preprosta. Prebrati moramo le posamezne vrstice podatkov. To storimo z zanko prikazano

[15]Document Object Model

s kodo 3.23.

Koda 3.23: Zanka za prebiranje napovedi prihodov avtobusov iz kode HTML.

```
// Poišči element, ki ima id enak "Form1"
Element form1 = doc.getElementById("Form1");
List<TextNode> ltn = form1.textNodes();
Iterator<TextNode> i = ltn.iterator();
while (i.hasNext()) { // Sprehodi se skozi vrstice.
  TextNode tn = (TextNode) i.next();
  if (tn.isBlank()) continue; // Preskoči prazne
     vrstice.
  String txt = tn.text(); // Besedilo vrstice zapiši v
     niz txt
  // Sledi koda za razčlenjevanje posamezne vrstice.
}
```

Če posamezno vrstico izpišemo, dobimo sledeče čiste podatke:

```
NAPOVED PRIHODOV ZA Jadranska (603012)
1 VIŽMARJE - MESTNI LOG: 14:04, n 14:18, 14:30
NAPOVED PRIHODOV ZA Jadranska (603011)
1 MESTNI LOG - GARAŽA: 22:33
1 N MESTNI LOG - BAVARSKI DVOR - GAMELJNE: 22:15, 22:46
1 MESTNI LOG - VIŽMARJE: n 14:00, n 14:15, 14:30
```

Naša naslednja naloga je razbiranje posameznih informacij iz vrstice. Kot vidimo, imamo v enem sklopu napovedi za dve postaji, obe z istim imenom (Jadranska) a različnima številkama (`603012` in `603011`). Vsaka predstavlja svojo stran ceste, zato jih moramo interpretirati ločeno. Za branje vrstic uporabimo dva regularna izraza (angl. *regular expressions*):

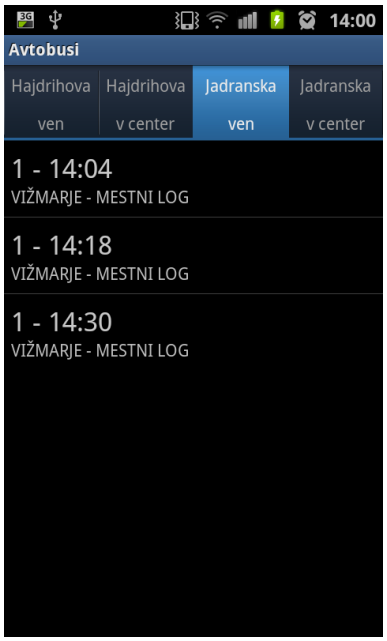
1. `NAPOVED PRIHODOV ZA (.+) \\(([0-9]+)\\)`
2. `([0-9]+)[ ]?([A-Z]?) ([^:]+): (.*)`

Prvi zajame vrstice, ki določajo stran ceste za katero bodo veljale napovedi, ki sledijo, drugi pa napovedi same. Na ta način v enem koraku iz niza izločimo

samo podatke, ki nas zanimajo. Če se bo oblika, v kateri dobivamo podatke s spletne strani kdaj spremenila, bo posodobitev aplikacije sicer nujna, a dokaj enostavna. S podatki, ki jih najde prvi regularni izraz ustvarimo objekte tipa *BusStop*, s podatki, ki jih najde drugi pa objekte tipa BusRoute. Objekti *BusStop* predstavljajo avtobusne postaje, objekti *BusRoute* pa relacije avtobusov. *BusStop* lahko vsebuje seznam relacij *BusRoute*, *BusRoute* pa vsebuje seznam časov, ob katerih avtobus prispe na postajo. Čase vseh relacij moramo nato razporediti po vrsti od najbližjega do najbolj oddaljenega. To storimo tako, da se sprehodimo preko vseh časov v vseh relacijah *BusRoute*, ki pripadajo določeni avtobusni postaji. Za vsak čas posebej v drevesno strukturo *TreeMap* dodamo relacijo, ki mu pripada. Kot ključ uporabimo objekt *Calendar*, vrednost pa je *BusRoute*. Ker je seznam drevesna struktura, bodo v njem vse relacije samodejno urejene padajoče po času. Opisan postopek je razviden iz kode 3.24.

Koda 3.24: Postopek za ustvarjanje drevesnega seznama z objekti BusRoute urejenimi po objektih Calendar.

```
TreeMap<Calendar, BusRoute> timeTable = new
    TreeMap<Calendar, BusRoute>();
Iterator<BusRoute> ir = routes.iterator();
while (ir.hasNext()) { // Sprehodi se čez vse relacije
    v trenutni avtobusni postaji
  BusRoute br = ir.next();
  List<Calendar> times = br.getTimes();
  Iterator<Calendar> it = times.iterator();
  while (it.hasNext()) { // Sprehodi se čez vse čase v
      relaciji
    Calendar time = it.next();
    timeTable.put(time, br); // V seznam dodaj
        relacijo s ključem time
  }
}
```



Slika 3.16: Prikaz časov prihodov avtobusov na postajo Jadranska v smeri izven centra Ljubljane.

Preostane nam le, da seznam relacij, urejen po časih, na zaslonu prikažemo na čim bolj pregleden način kot prikazuje slika 3.16. Imamo dve avtobusni postaji (Hajdrihova in Jadranska), ki imata vsaka po dve postajališči (ena na vsaki strani ceste). Skupaj so to štiri postajališča, katerih čase prihodov avtobusov prikažemo v ločenih jezičkih (angl. *tabs*).

3.6 Restavracije

V okolici Fakultete za računalništvo in informatiko je kar nekaj restavracij, ki nudijo prehrano na bone. Priročno bi bilo imeti nekaj splošnih informacij, ki so dostopne preko mobilne naprave. Trenutno obstajata dve konkurenčni spletni strani, ki zbirata in objavljata informacije o restavracijah na bone po Sloveniji:

1. `http://www.studentska-prehrana.si/`
2. `http://www.student-info.net/index.php/bonapetit/`

Slika 3.17: Aktivnost s seznamom restavracij.

Kot vir informacij za našo aplikacijo smo izbrali spletno stran *Student Info* [9]. Glavni razlog za izbiro je bila večja baza restavracij. Želeli smo pridobiti tudi podatke o dnevnih menijih restavracij. Izkazalo se je, da nobena izmed navedenih spletnih strani podatkov ne posodablja redno, ali pa jih sploh nima. Zato bomo prikazovali samo osnovne podatke o restavracijah. Slaba stran izbire spletne strani je, da imajo očitno nekaj težav s kodiranjem znakov. Na to naša aplikacija ne more vplivati.

Komponenta restavracij je najpreprostejša izmed vseh. Ima le dve aktivnosti. Prva aktivnost je enostaven seznam restavracij v okolici FRI. Izbrane so bile izključno po oddaljenosti od fakultete. Razdalja naj bi bila manjša od približno 500 metrov. Seznam je shranjen kot del aplikacije v obliki vira v datoteki XML. Vsakršna sprememba seznama torej zahteva posodobitev aplikacije. Namesto tega bi seznam pridobili iz namenskega strežnika, a menimo, da spremembe restavracij ne bodo zelo pogoste. Na ta način bi le porabili več podatkovnega prenosa. Aktivnost si lahko ogledamo na sliki 3.17.

Ob izbiri restavracije preklopimo na aktivnost, ki prikaže podrobnosti o restavraciji. Ker spletna stran ne ponuja nobene storitve, ki bi nam lahko podatke posredovala v surovem formatu, smo jih primorani razbrati neposredno iz spletne strani. Po eni strani je to nekoliko tvegano, saj ne vemo ali bo struktura spletne strani ostala enaka. Po drugi strani pa nam ni treba skrbeti glede vsebine. Če se vsebina spremeni, jo bo takšno prikazala tudi naša mobilna aplikacija.

Za pridobivanje podatkov iz spletne strani, kot že pri avtobusih, tudi tukaj uporabljamo knjižnico *jsoup*. Razlog je v tem, da je njena uporaba bistveno enostavnejša od katerekoli alternative. Oglejmo si del spletne strani, ki nas zanima:

```
<table cellpadding="3" width="100%"><tr>
  <td valign="top" width="50%">
    <img
        src="http://www.student-info.net/sis-mapa/prehrana/
             sis-mapa/slika/slika\_600.jpg"
        alt="Študentska restavracija FE in FRI"
        title="Študentska restavracija FE in FRI"
        style="max-width:250px;">
  </td>
  <td valign="top">
    Naslov: Tržaška 25, 1000, Ljubljana<br>
    Kontakt: 01-476-84-04, 030 313 602,
       kulinar@fe.uni-lj.si<br>
    Spletna stran: <br>
    Tip restavracije: samopostrezna<br>
    <br>
    <b>Boni v prodaji? </b><b>Normalno.</b><br>
    <b>Cena bona: 1.61 e </b><br>
    <br>
```

```
    <b>Opis: </b>Samopostrežna restavracija in
        pizzerija namenjena predvsem študentski
        populaciji,za prehrano zaposlenih in pripravo
        hladno toplih bifejev ter razli?nih vrst
        catering storitev.<br><br>
  </td>
</tr></table>
```

Iščemo torej element z imenom `table`, ki vsebuje osnovne podatke o restavraciji. Element je vedno šesti element z imenom `table` na spletni strani, kar je takorekoč edini kriterij po katerem ga lahko najdemo. Znotraj elementa sta dve celici (elementa HTML z imenom `td`). Prvi vsebuje sliko, drugi pa podatke in opis restavracije. Iz slike izločimo atribut `src`, ki predstavlja pot do slike restavracije. Opis in podatke izluščimo tako, da iz elementa `table` vzamemo vsebino zadnje celice (element `td`). Podatke s kodo 3.25 pridobimo in prikažemo v naši aktivnosti.

Koda 3.25: Koda za razbiranje podatkov iz oblike HTML.

```
// Najde vse elemente "table"
Elements tables = doc.getElementsByTag("table");
// Iz seznama elementov "table" izberi 6. zaporednega
Element table = tables.get(6);
// V izbranem elementu "table" najdi vse elemente "td"
Elements tds = table.getElementsByTag("td");
// V izbranem elementu "table" najdi prvi element "img"
Element img = table.getElementsByTag("img").first();
// Iz seznama elementov "td" izberi zadnjega
Element el = tds.last();
// Prenesi sliko in izvedi prikaz v ImageView
imageViewRestaurant.setImageBitmap(
    Utils.downloadBitmap(img.attr("src")) );
// Prikaži opis restavracije v WebView
```

Slika 3.18: Aktivnost s sliko in podatki študentske restavracije FE in FRI.

```
webViewRestaurant.loadData(header+el.toString(),
    "text/html", null);
```

Rezultat je slika 3.18.

3.7 Bralnik novic RSS

Uradna spletna stran Fakultete za računalništvo in informatiko ima objavljenih kar nekaj povezav do različnih novic v obliki RSS [11]. Ideja je bila preprosta: naredimo komponento za našo mobilno aplikacijo, ki bo znala prebrati in prikazati novice RSS s spletne strani fakultete. Izkazalo se je, da je realizacija nekoliko bolj zapletena. Težava je v tem, da ne Java, ne SDK Android ne vsebujeta nobenih razredov ali metod za razčlenjevanje zapisa RSS. Razvijalcu tako preostane samo lastna implementacija tovrstnega orodja.

Po dolgotrajnem iskanju rešitve smo na spletni strani *IBM developerWorks* [12] odkrili implementacijo bralnika RSS za sistem Android. Avtor kode je Michael Galpin, programski arhitekt pri podjetju *eBay Inc.*. Koda

predstavlja odlično osnovo za našo aplikacijo, a je kljub temu potrebovala številne prilagoditve in poenostavitve.

Naša izvedba sestoji iz dveh razredov: *FeedMessage* in *FeedParser*. V osnovi prvi predstavlja element, ki se lahko pojavi v vsebini RSS, drugi pa skrbi za pridobitev posameznih elementov. Njuna uporaba je dokaj preprosta. Najprej moramo ustvariti objekt razreda *FeedParser*, pri čemer podamo niz, ki predstavlja naslov URL do vsebine RSS, ki jo želimo prebrati. Nato na pravkar ustvarjenem objektu pokličemo metodo `parse()`, ki nam vrne seznam objektov tipa *FeedMessage*. Posamezni objekt *FeedMessage* tako vsebuje štiri osnovne podatke o vsakem elementu. To so naslov, opis, povezavo URL in datum objave.

Android sicer ne pozna funkcij za računjevanje vsebine RSS ima pa zato vgrajen razčlenjevalnik vsebin XML. Vemo namreč, da RSS ni nič drugega kot XML s točno določenimi elementi in strukturo. Zato razred FeedParser za svojo nalogo uporablja prav razred Xml, ki ga najdemo v paketu `android.util`. Poznamo dve vrsti razčlenjevalnikov: DOM[16] in SAX[17]. Prvi je orientiran objektno, drugi dogodkovno. Razlik je precej. DOM prebere vsebino XML in v spomin shrani celoten model. Z vidika programerja je to bistveno lažje in pregledneje za uporabo. Slaba stran je visoka poraba pomnilniškega prostora. SAX po drugi strani potrebuje zelo majhno količino pomnilnika. Deluje tako, da se preko vsebine XML sprehodi enkrat in ob dogodkih (na primer odkritje elementa z določenim imenom) kliče metode, katerih vsebino napiše programer. To je nekoliko manj pregledno, a pri tem pridobimo na nizki porabi pomnilnika, kar je za mobilne naprave izjemno pomembno. Metode razreda *Xml* uporabljajo dogodkovni način SAX.

Metoda `parse()` razreda *FeedParser* lovi pet različnih dogodkov pri sprehodu preko elementa `item` v vsebini RSS. Prvi štirje so odkritje elementov z imeni `title`, `link`, `description` in `pubDate`. Ob vsakem dogodku se prebere vsebina odkritega elementa, ki se shrani v trenuten objekt *FeedMes-*

[16]Document Object Model
[17]Simple API for XML

sage. Zadnji dogodek je konec elementa `item`. V tem primeru trenutni objekt *FeedMessage*, ki predstavlja element `item`, dodamo na konec seznama. Po zaključeni operaciji metoda 3.26 vrne seznam vseh dodanih objektov.

Koda 3.26: Metoda parse() razreda FeedParser.

```
public List<FeedMessage> parse() throws Exception {
  final FeedMessage currentMessage = new FeedMessage();
  RootElement root = new RootElement(RSS);
  final List<FeedMessage> messages = new
     ArrayList<FeedMessage>();
  Element channel = root.getChild(CHANNEL);
  Element item = channel.getChild(ITEM);
  item.setEndElementListener(new EndElementListener() {
    public void end() { // Konec elementa ITEM
      messages.add(new FeedMessage(currentMessage));
    }
  });
  item.getChild(TITLE).setEndTextElementListener( new
     EndTextElementListener() {
    public void end(String body) {  // Konec elementa
       TITLE
      currentMessage.setTitle(body);
    }
  });
  item.getChild(LINK).setEndTextElementListener( new
     EndTextElementListener() {
    public void end(String body) {  // Konec elementa
       LINK
        currentMessage.setLink(body);
    }
  });
  item.getChild(DESCRIPTION).setEndTextElementListener(
```

```
        new EndTextElementListener() {
      public void end(String body) {  // Konec elementa
          DESCRIPTION
              currentMessage.setDescription(body);
      }
    });
    item.getChild(PUB_DATE).setEndTextElementListener(
        new EndTextElementListener() {
      public void end(String body) {  // Konec elementa
          PUB_DATE
        currentMessage.setDate(body);
      }
    });
    Xml.parse(this.getInputStream(), Xml.Encoding.UTF_8,
        root.getContentHandler());
    return messages;
}
```

Preostal del naloge je vizualni prikaz prebranih elementov na zaslon uporabnika. Komponenta za branje novic RSS je sestavljena iz treh aktivnosti. Prva prikaže seznam vseh novic (glej sliko 3.19), kot so objavljene na uradni spletni strani FRI. Njihove povezave so zapisane statično v viru, ki je priložen aplikaciji. V primeru spremembe katerega od naslovov bo potrebna posodobitev aplikacije. Alternativna rešitev bi bila branje naslovov direktno iz spletne strani, a bi za to porabili dodaten prenos podatkov. Prav tako bi morali posodobiti aplikacijo v primeru, da bi se spremenila struktura spletne strani.

Ob izbiri novice moramo najprej preveriti ali imamo dostop do interneta. Če ga nimamo, na to opozorimo uporabnika, in se vrnemo na prejšnjo aktivnost. Potem z naslovom izbranega vira ustvarimo objekt razreda *FeedParser* in kličemo metodo `parse()`. Če ne pride do napak, prebrane elemente izpišemo na zaslon. Napake, do katerih lahko pride, so zelo raznovrstne. Poskrbeti moramo, da aplikacija ne preneha delovati. To naredimo s pomočjo

Slika 3.19: Aktivnost s seznamom virov novic RSS fakultete.

izjem (angl. *exception handling*). Primer izjeme je nedosegljivost vira na danem naslovu. V takšnem primeru uporabniku sporočimo napako in se vrnemo nazaj na seznam virov RSS.

Poseben primer je napaka pri interpretiranju prebranih podatkov. Največji problem nam je predstavljal seznam zagovorov diplomskih nalog. Zgodi se namreč, da so imena dni v datumih včasih navedena v angleškem, včasih pa v slovenskem jeziku (npr. *Wed* ali *sre* za sredo). Razlog za to je neznan. Izbira jezika je vsaj na videz popolnoma naključna. Za to smo morali poskrbeti, da ob napaki med interpretiranjem datuma poskusimo še z drugim jezikom. To je urejeno v metodi `setDate()` v razredu *ParseMessage*. Po uspešni interpretaciji vsebino RSS prikažemo na zaslon, kot kaže slika 3.20.

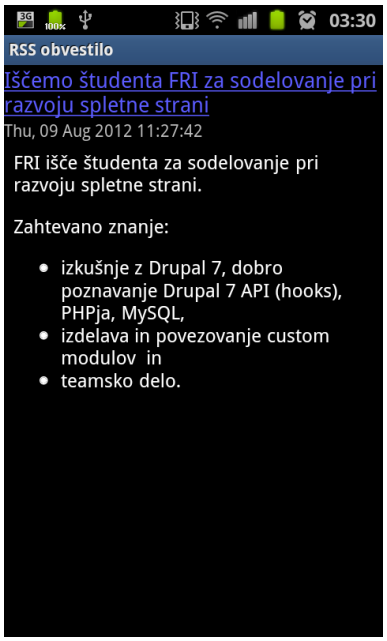
Koda 3.27: Metoda za interpretacijo datuma.

```
public void setDate(String date) {
  while (!date.endsWith("00"))
    date += "0"; // Kadar datum na koncu nima dveh 0,
       eno dodaj
```

Slika 3.20: Aktivnost s seznamom novic.

```
    try {
      this.date = dateFormatter.parse(date.trim());
    } catch (ParseException e) {
      nextLocale(); // Interpretacija spodletela,
          nastavi drug jezik
      try {
        this.date = dateFormatter.parse(date.trim());
      } catch (ParseException e2) {
        throw new RuntimeException(e2);
      }
    }
}
```

Zadnja aktivnost prikaže novico, ki jo je izbral uporabnik. Podatke ji v obliki nizov posreduje prejšnja aktivnost, zato dodatnega prenosa podatkov preko interneta ne potrebuje. Ker je opis novice podan v obliki HTML ga prikažemo v vsebovalniku *WebView*. Pri tem moramo upoštevati kodiranje.



Slika 3.21: Aktivnost, ki prikazuje izbrano novico.

Privzeto kodiranje je UTF-8[18]. Da *WebView* uporabi pravilno kodiranje, poskrbimo s trikom tako, da vsebini opisa na začetek pripnemo niz:

```
<?xml version="1.0"encoding="UTF-8"?>
```

Želimo tudi, da se ob pritisku s prstom na naslov novice RSS le-ta odpre v spletnem brskalniku. To dosežemo tako, da objektu, ki predstavlja naslov (v našem primeru `feedTitle`) s kodo 3.28 s pomočjo razreda *Html* vnesemo vsebino HTML.

Koda 3.28: Koda za nastavitev povezave URL objektu TextView.

```
feedTitle.setText(Html.fromHtml("<a
    href=\""+feed_link+"\">"+feed_title+"</a>"));
```

Primer rezultata je prikazan na sliki 3.21.

[18]UCS Transformation Format 8-bit

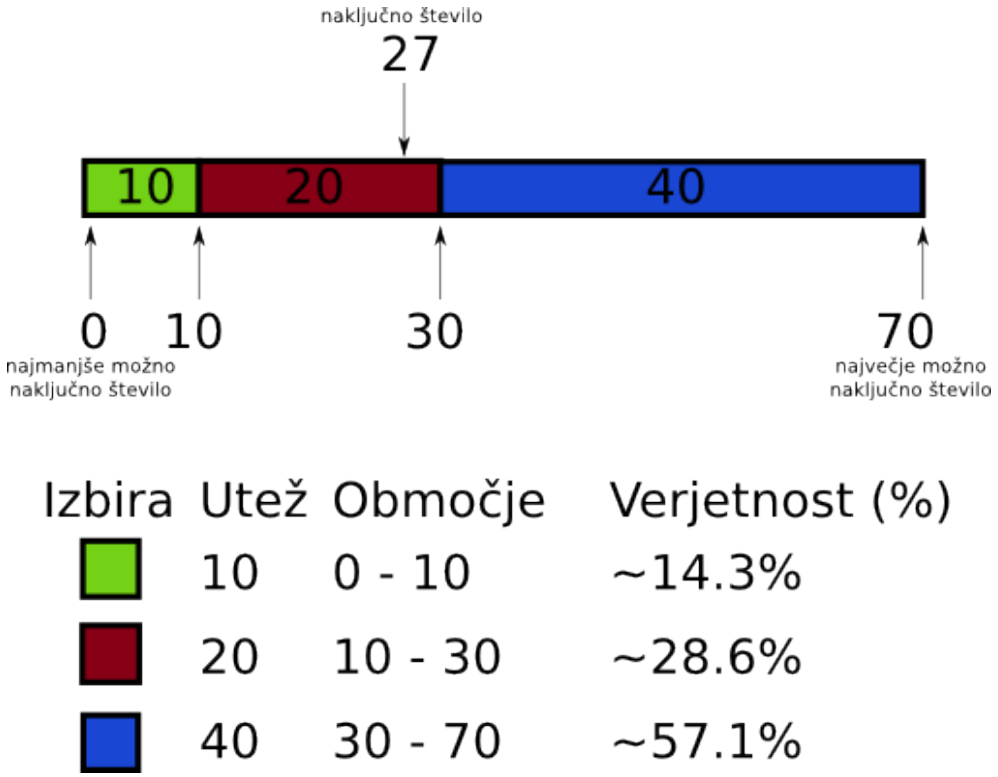
3.8 Oglasi

Prikazovanje oglasov v mobilni aplikaciji Android je običajno realizirano s pomočjo Googlove storitve *AdMob*. V našem primeru bomo za posredovanje oglasov odjemalcem skrbeli sami. Strežnik oglasov je preprost program, napisan v programskem jeziku PHP, ki teče na oddaljenem spletnem strežniku. Ima dve osnovni nalogi, izbiro oglasa in njegov prikaz.

Skripta omogoča posredovanje praktično neomejenega števila različnih oglasov. Popolnoma razumljivo je, da ne morejo biti vsi enakovredni. Različni oglaševalci so za oglaševanje pripravljeni nameniti različne vsote denarja. To pomeni, da morajo nekateri oglasi imeti prednost pred drugimi. V ta namen smo pri izbiri oglasa za prikaz izbrali algoritem obteženih naključnih števil. Njegova implementacija 3.29 je vizualno prikazana na sliki 3.22. Vsak oglas, ki ga dodamo na seznam za izbiro, ima določeno svojo utež. Utež je število, ki določa verjetnost izbire oglasa. Verjetnost izbire oglasa z določeno utežjo se izračuna po formuli (3.1).

Koda 3.29: Implementacija algoritma obteženih naključnih števil.

```
$totalWeight = 0;
foreach ($ads as &$ad) {
  // Sestej utezi v skupno vsoto
  $totalWeight += $ad->getWeight();
}
// Izberi nakljucno st. med 0 in skupno vsoto utezi
$threshold = rand(0,$totalWeight);
foreach ($ads as &$ad) {
  $weight = $ad->getWeight();
  // Ko trenutna utez presega nakljucno stevilo, imamo
     izbiro
  if ($weight > $threshold)
    return $ad;
  // Odstej trenutno utez
```



Slika 3.22: Vizualizacija algoritma za izbiro naključnega obteženega oglasa.

```
    $threshold -= $weight;
}
```

$$P = W_{oglasa}/W_{skupna} \tag{3.1}$$

Ko je oglas izbran, ga je potrebno samo še izpisati. Vsak oglas se nahaja v svoji datoteki HTML, kjer je definirana njena vsebina. Strežnik oglasov vsebino datoteke prebere in izpiše. Vsebina oglasa je lahko poljubna spletna vsebina. Pomembno je, da oglas ne spreminja poslane glave HTTP protokola. V primeru, da oglas naredi preusmeritev na podlagi glave HTTP, bo aplikacija v Androidu uporabnika preusmerila na spletni brskalnik in oglas prikazala v njem. Želimo, da aplikacija vsebino prikaže v posebnem pogledu imenovanem *WebView*. Za pravilen izris na vseh možnih ločljivostih mora poskrbeti avtor oglasa. Za to lahko uporabi katerikoli element HTML ali celo

kodo CSS[19].

3.9 Ikone

Android predvideva 7 osnovnih vrst ikon.

- *Launcher icons* zaganjalne ikone
- *Menu icons* menijske ikone
- *Action bar icons* ikone vrstice dejanj
- *Status bar icons* ikone vrstice stanj
- *Tab icons* ikone zavihkov
- *Dialog icons* ikone pogovorov

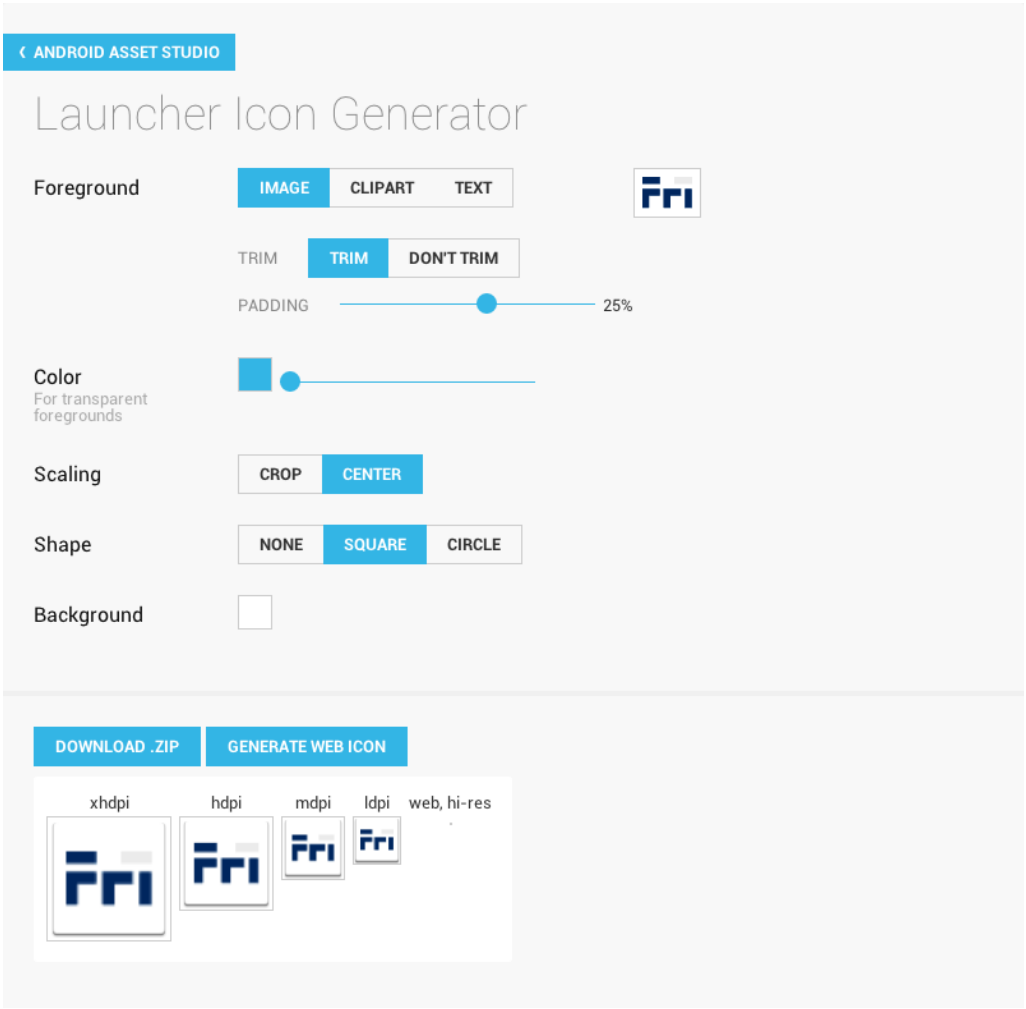
Android ima vgrajen nabor privzetih ikon v paketu `drawables`. Njihova uporaba je močno zaželjena, saj poskrbijo za bolj standarden izgled aplikacije. Poleg tega različne verzije Androida uporabljajo različne ikone. To pomeni, da bo naša aplikacija vedno uporabljala najnovejše ikone. Slaba stran vgrajenih ikon je, da jih je premalo. Velik del jih je celo rezerviranih samo za sistem. Programerju so dostopne samo javne, s katerimi so pokriti najbolj osnovni koncepti. Če želimo na primer ikono s sliko avtobusa, moramo izdelati svojo.

Za izdelavo aplikacije smo potrebovali tri vrste ikon: zaganjalne, menijske in pogovorne. Tako kot vse grafike morajo biti tudi ikone, ki se pojavljajo v projektu Android, narejene v štirih izvedbah:smo se zato

- HDPI (High DPI[20])
- LDPI (Low DPI)
- MDPI (Medium DPI)
- XHDPI (eXtra High DPI)

Te predstavljajo različne točkovne gostote. V projektu Android vsaki pripada podimenik v imeniku `res/` s predpono `drawable-`. Glavna ikona aplikacije se tako nahaja na naslednjih lokacijah:

[19]Cascading Style Sheets
[20]Dots Per Inch (število točk na palec)



Slika 3.23: Ikona mFRI v orodju Android Asset Studio.

- `src/drawable-hdpi/ic_launcher.png`
- `src/drawable-ldpi/ic_launcher.png`
- `src/drawable-mdpi/ic_launcher.png`
- `src/drawable-xhdpi/ic_launcher.png`

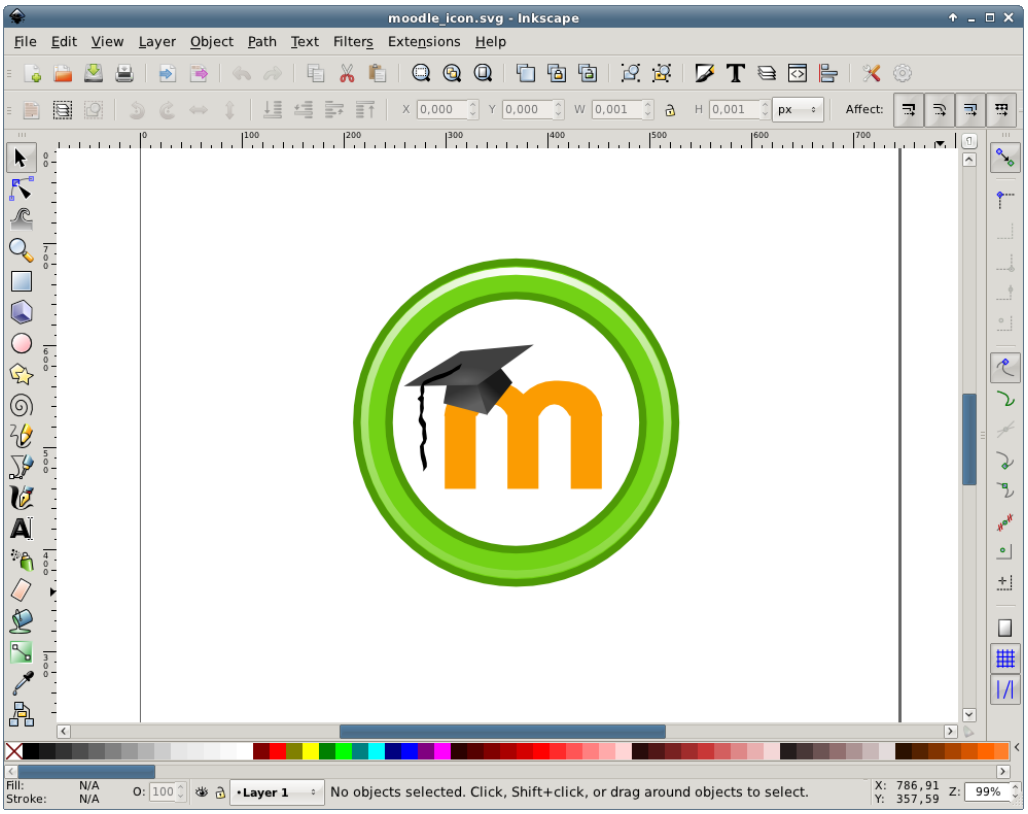
Najlažje in najhitreje je ikone izdelati s pomočjo spletnega orodja *Android Asset Studio* [14], kot ga prikazuje slika 3.23.

Ostale ikone so pridobljene s spletne strani *The Noun Project* [15]. Gre za največjo zbirko prosto dostopnih ikon, ki predstavljajo po en samostalnik. Dela na spletni strani so objavljena pod različnimi variacijami licence

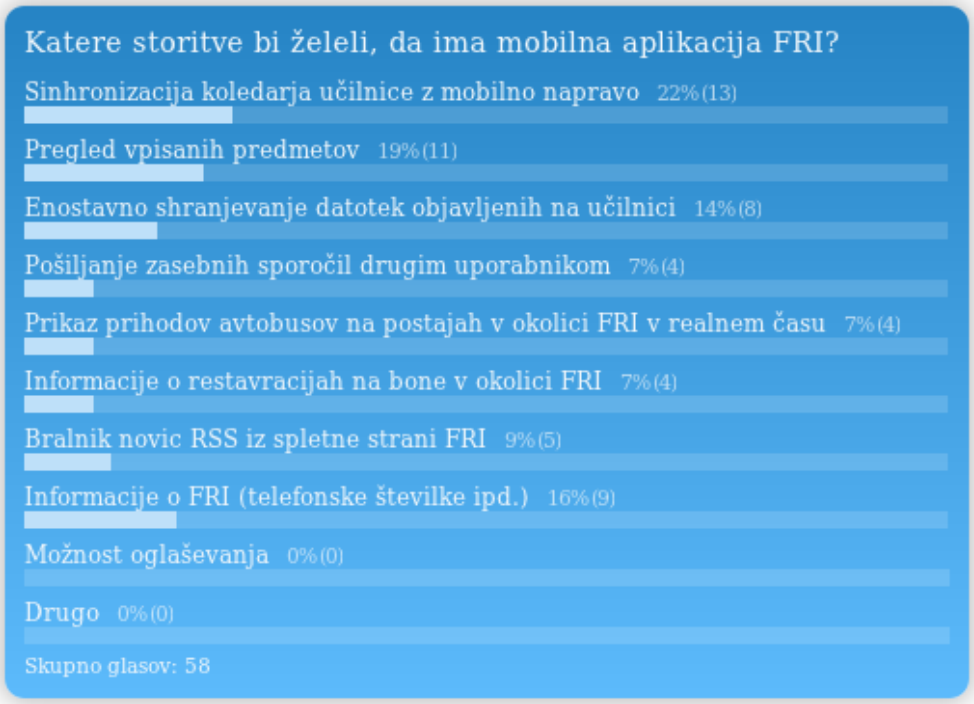
Creative Commons. Najbolj osnovna je *No Rights Reserved (CC0)*, ki nima nobenih omejitev. Najpogosteje se uporablja *Creative Commons - Attribution (CC BY 3.0)*, ki dovoljuje kopiranje, distribuiranje, prilagajanje in komercialno rabo pod pogojem, da navedemo kdo je avtor izvirnega dela. Če bodo v prihodnjem razvoju aplikacije uporabljene grafike, ki zahtevajo navedbo avtorjev, bo to mogoče narediti v trenutno onemogočeni aktivnosti z imenom "*Avtorstvo*", ki bo preko menija dostopna iz aktivnosti "*O programu*". Do sedaj so bile uporabljene samo grafike, ki so v javni lasti ali pa imajo licenco *CC0*, zato navedba avtorjev ni bila potrebna.

Posebnost je ikona Moodle, ki je bila narejena na osnovi logotipa Moodle in ikone *Emblem*, ki je del projekta *GNOME*. Obe sta objavljeni pod licenco GPL. Obdelani sta bili v vektorski obliki SVG[21] s pomočjo odprtokodnega programa *Inkscape*. Rezultat je viden na sliki 3.24.

[21]Scalable Vector Graphics



Slika 3.24: Ikona Moodle v programu Inkscape.



Slika 3.25: Rezultati ankete o storitvah mobilne aplikacije FRI.

Poglavje 4

Sklepne ugotovitve

V času pisanja diplomske naloge smo razvili aplikacijo za mobilne naprave z operacijskim sistemom Android. Opravili smo anketo o storitvah, ki bi jih študentje radi imeli v mobilni aplikaciji je. Kot je razvidno iz slike 3.25, je pokazala, da je največ zanimanja za storitve, ki se navezujejo neposredno na fakulteto. Kljub temu smo implementirali vse navedene izbire. To so odjemalec Moodle, pripomoček za pregled voznega reda mestnih avtobusov v okolici fakultete, osnovne informacije o bližnjih restavracijah, bralnik uradnih novic RSS in informacije o fakulteti. Tako je nastala razvejana aplikacija, ki že sama po sebi pokriva ogromno področij, z nadaljnjim razvojem pa omogoča še dodatne razširitve.

Tekom nastajanja aplikacije smo ugotovili, da je uradna dokumentacija za operacijski sistem Android nekoliko pomanjkljiva. Zaradi tega implementacija na videz preprostih stvari od neizkušenega razvijalca zahteva veliko vloženega truda in znanja, za kar porabi ogromno časa. Po drugi strani je programski jezik Java odlično dokumentiran, a je njegova uporaba vse prej kot programerju prijazna. Za zelo osnovne operacije je nemalokrat potrebno napisati precej programske kode. Kljub pomanjkljivi dokumentaciji za Android je to zelo dobro zasnovan operacijski sistem. Na žalost so starejše različice še vedno preveč razširjene med uporabniki, zato se ne moremo posluževati elegantnih novosti, ki jih prinašajo najnovejše različice.

Pri delu s spletnimi storitvami Moodle smo spoznali, da so te še v zgodnejši fazi razvoja, saj manjka velik del funkcij, ki bi nam prišle zelo prav. Razvijalci sistema Moodle imajo implementacijo manjkajočih funkcij v načrtu brez ciljnega datuma. Zato smo se v okviru diplomske naloge odločili uporabiti predvsem obstoječe funkcije. Hkrati smo predvideli kasnejšo implementacijo novih, ki bodo omogočile nove načine interakcije s spletno učilnico. Funkcijo, ki skrbi za posredovanje koledarja odjemalcu smo zaradi njene nujnosti napisali. Ko bodo v prihodnosti razvijalci v Moodle dodajali nove funkcije, bomo lahko sproti nadgrajevali tudi našo mobilno aplikacijo.

Slike

Literatura

[1] (2012) Ubuntu Linux. Dostopno na: http://www.ubuntu.com/.

[2] (2012) Dashboards, Android Developers. Dostopno na: http://developer.android.com/about/dashboards/index.html.

[3] (2012) Using Hardware Devices, Android Developers. Dostopno na: http://developer.android.com/tools/device.html#VendorIds.

[4] (2012) Web service API Roadmap, Moodle Tracker. Dostopno na: http://tracker.moodle.org/browse/MDL-29934.

[5] (2012) Calendar Provider, Android Developers. Dostopno na: http://developer.android.com/guide/topics/providers/calendar-provider.html.

[6] (2012) StackOverflow, funkcija *getCalendarUriBase()*. Dostopno na: http://stackoverflow.com/a/6846964/1132363/.

[7] (2012) iCal4j 1.0.2 API. Dostopno na: http://m2.modularity.net.au/projects/ical4j/apidocs/.

[8] (2012) RFC 5545, Internet Calendaring and Scheduling Core Object Specification (iCalendar). Dostopno na: http://tools.ietf.org/html/rfc5545.

[9] (2012) Student Info Bonapetit. Dostopno na: http://www.student-info.net/index.php/bonapetit/.

[10] (2012) jsoup: Java HTML Parser. Dostopno na: http://jsoup.org/.

[11] (2012) Obvestila / RSS, Fakulteta za računalništvo in informatiko. Dostopno na: http://www.fri.uni-lj.si/si/fakulteta/rss/.

[12] (2012) Michael Galpin, Working with XML on Android. Dostopno na: http://www.ibm.com/developerworks/opensource/library/x-android/index.html.

[13] (2012) token.php does not return the token sometimes, Moodle Tracker. Dostopno na: http://tracker.moodle.org/browse/MDL-29805.

[14] (2012) Android Asset Studio. Dostopno na: http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html.

[15] (2012) The Noun Project. Dostopno na: http://thenounproject.com/.