PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET METROPOLITAN BEOGRAD

KI205 - JAVA 8: PROGRAMIRANJE U JAVI NA ANDROID PLATFORMI

Provajderi sadržaja

Lekcija 06

PRIRUČNIK ZA STUDENTE

KI205 - JAVA 8: PROGRAMIRANJE U JAVI NA ANDROID PLATFORMI

Lekcija 06

PROVAJDERI SADRŽAJA

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

UVOD

Android operativni sistem predstavlja koncept provajder sadržaja koji omogućava razmenu podataka između različitih paketa.

Android operativni sistem podržava *SQLite* sistem za upravljanje bazom podatak. Navedeni sistem podržava sve poznate operacije nad bazom podataka ali ima i jedan nedostatak. Bata podataka, kreirana SQLite, sistemom za neku mobilnu aplikaciju, nije dostupna izvan matične aplikacije. Otuda, bilo je neophodno naći načine da bi se podaci od značaja učinili dostupnim izvan paketa u kojem su kreirani.

Android operativni sistem predstavlja koncept *provajder sadržaja* koji omogućava razmenu podataka između različitih paketa. U ovoj lekciji akcenat će biti na sledećim temama:

- Šta su provajderi sadržaja?
- Kako se provajderi sadržaja koriste u Android operativnom sistemu?
- Kako se kreira i koristi vlastiti provajder sadržaja?

Savladavanjem ove lekcije studenti će biti osposobljeni da kreiraju Android aplikacije koje imaju mogućnost razmene podataka sa drugim Android aplikacijama, bilo da su to korisničke aplikacije ili već postojeće, integrisane u Android OS mobilnog uređaja. Posebno, studenti će razumeti i koristi poznate provajdere sadržaja koji su detaljno obrađeni u ovj lekciji.

Poglavlje 1

Deljenje podataka u Android OS

UVOD U ANDROID DELJENJE PODATAKA

U Android operativni sistem je ugrađeno nekoliko veoma korisnih provajdera sadržaja.

Android operativni sistem predlaže *provajdere sadržaja* za deljenje podataka između različitih Android paketa. Provajder sadržaja može biti shvaćen kao izvesno skladište podataka kojem paketi pristupaju primenom odgovarajućeg interfejsa. U mnogim situacijama, provajder sadržaja se ponaša slično bazi podataka. Moguće je postavljati upite, menjati podatke, dodavati nove i uklanjati stare podatke i tako dalje. Međutim, za razliku od baze podataka, provajder sadržaja može da koristi različite načine skladištenja podataka. Podaci mogu da budu smešteni u bazu podataka, datoteku ili dostupni preko mreže.

U Android operativni sistem je ugrađeno nekoliko veoma korisnih provajdera sadržaja:

- Browser – čuva podatke kao što se zabeležene web stranice, istorija pregleda stranica itd;
- CallLog – čuva podatke kao što su propušteni pozivi, detalji o pozivima itd;
- Contacts – čuva podatke o kontaktima;
- MediaStore – čuva multimedijalne datoteke;
- Settings – čuva podešavanja uređaja i preferencije korisnika.

Pored ugrađenih, Android podržava rad sa provajderima sadržaja koje su kreirali programeri tokom razvoja izvesnih Android aplikacija. U nastavku biće pokazano kako se vrše operacije nad provajderima sadržaja, kako se oni koriste i kako se vrši filtriranje informacija nad provajderima sadržaja.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

UPITI NAD PROVAJDERIMA SADRŽAJA

U Androidu, upit nad provajderom sadržaja koristi formu URI identifikatora.

U sledećem izlaganju neophodno je obraditi mehanizme za vršenje upita nad provajderima sadržaja, kao i formu u kojoj se upiti postavljaju nad provajderima. U Android operativnom sistemu, upit nad provajderom sadržaja koristi formu URI (Uniform Resource Identifier)

identifikatora sa opcionim specifikatorom koji se odnosi na konkretnu vrstu. Opšti oblik upita nad provajderom sadržaja izgleda ovako:

<standardni_prefiks>://<vlasnik>/<putanja_podataka>/<id>

Upit je izgrađen iz sledećih komponenata:

- standardni_prefiks- za provajdere sadržaja je uvek *content://.*
- vlasnik – predstavlja naziv provajdera sadržaja.
- putanja_podataka – specificira vrstu traženih podataka. Na primer, ukoliko su u aplikaciji neophodni kontakti iz *Contacts* provajdera sadržaja, putanja može da bude označena kao *people*, a URI identifikator da glasi: content://*contacts/people.*
- id – specificira zahtevani zapis. Na primer, ukoliko se zahteva peti kontakt u *Contacts* provajderu sadržaja, *URI* identifikator može da ima sledeći oblik: content://*contacts/ people/5.*

Sledećom slikom predstavljeni su često korišćeni stringovi upita nad provajderima sadržaja sa odgovarajućim opisom.

String upita	Opis
content://*media/internal/images*	Vraća listu svih slika iz interne memorije
content://*media/external/images*	Vraća listu svih slika iz eksterne memorije
content://*call_log/calls*	Vraća listu svih poziva koje je registrovao Call Log
content://*browser/bookmarks*	Vraća listu zapamćenih stranica web čitačem

Slika 1.1 Primeri stringova upita

PRIMER 1 - KORIŠĆENJE UGRAĐENIH PROVAJDERA SADRŽAJA

U AndroidManifest.xml datoteci neophodno je ugraditi odgovarajuće privilegije.

Za razumevanje koncepta provajdera sadržaja biće uveden odgovarajući primer. U Android StudioIDE kreira se projekat pod nazivom ProvajderDemo. Za njegovo kreiranje biće korišćen jedna od najnovijih API verzija koja odgovara Android 6.0 operativnom sistemu.

Sledećim listingom priložen je XML kod datoteke korisničkog interfejsa.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/android:list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:stackFromBottom="false"
```

```
        android:transcriptMode="normal" />

    <TextView
        android:id="@+id/contactName"
        android:textStyle="bold"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/contactID"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Da bi kreirana Android aplikacija imala mogućnost korišćenja provajdera sadržaja, moraju da joj se odobre izvesne privilegije. Navedene privilegije se ugrađuju u datoteku AndroidManifest.xml projekta koji odgovara kreiranoj aplikaciji. U konkretnom slučaju, aplikaciji će biti dozvoljeno da čita kontakte, a to je realizovano instrukcijom: *<uses-permission android:name="android.permission.READ_CONTACTS"/>* .

AndroidManifest.xml datoteka sa ugrađenom dozvolom pristupa odgovarajućem sadržaju, data je sledećim kodom.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.metropolitan.provajderdemo">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

KORIŠĆENJE UGRAĐENIH PROVAJDERA SADRŽAJA – JAVA KLASA

URI objekat je ugrađen u klasu aktivnosti aplikacije.

JAVA klasom aktivnosti projekta obavljaju se sve aktivnosti vezane za pristup i manipulaciju sadržajem kojeg obezbeđuje određeni provajder sadržaja. URI objekat, koji je odgovoran za izvršavanje stringa upita, ugrađen je u klasu aktivnosti aplikacije. U nastavku izlaganja biće prikazan programski kod koji odgovara kreiranoj klasi aktivnosti projekta ProvajderDemo.

```
package com.metropolitan.provajderdemo;

import android.app.ListActivity;
import android.content.CursorLoader;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.widget.CursorAdapter;
import android.widget.SimpleCursorAdapter;
import android.util.Log;
public class MainActivity extends ListActivity {


    /** Poziva se kada se aktivnost kreira. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Uri allContacts = ContactsContract.Contacts.CONTENT_URI;
       Uri allContacts = Uri.parse("content://contacts/people");
        Cursor c;
         String[] columns = new String[] {
                ContactsContract.Contacts.DISPLAY_NAME,
                ContactsContract.Contacts._ID,
                ContactsContract.Contacts.HAS_PHONE_NUMBER,

        };
        if (android.os.Build.VERSION.SDK_INT <11) {
            //---pre verzije Honeycomb---
            c = managedQuery(allContacts, columns,
                    ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?",
                    new String[] {"V%"},
                    ContactsContract.Contacts.DISPLAY_NAME + " ASC");
            //---Dozvoljava da aktivnost upravlja kursorom
            startManagingCursor(c);

        } else {
            //---Honeycomb i novije verzije---
            CursorLoader cursorLoader = new CursorLoader(
```

```
                    this,allContacts,columns,ContactsContract.Contacts.DISPLAY_NAME
+
                    " LIKE ?",
                    new String[] {"V%"},
                    ContactsContract.Contacts.DISPLAY_NAME + " ASC");
            c = cursorLoader.loadInBackground();
        }



        int[] views = new int[] {R.id.contactName, R.id.contactID };

        SimpleCursorAdapter adapter;

        if (android.os.Build.VERSION.SDK_INT <11) {
            //---pre Honeycomb---
            adapter = new SimpleCursorAdapter(
                    this, R.layout.activity_main, c, columns, views);
        } else {
            //---Honeycomb i novije verzije---
            adapter = new SimpleCursorAdapter(
                    this, R.layout.activity_main, c, columns, views,
                    CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
        }
        this.setListAdapter(adapter);
        PrintContacts(c);

    }

    private void PrintContacts(Cursor c){
        if (c.moveToFirst()) {
            do{
                String contactID = c.getString(c.getColumnIndex(
                        ContactsContract.Contacts._ID));
                String contactDisplayName =
                        c.getString(c.getColumnIndex(
                                ContactsContract.Contacts.DISPLAY_NAME));
                Log.v("Provajderi sadržaja", contactID + ", " +
                        contactDisplayName);
                //učitavanje broja telefona
                int hasPhone=
                        c.getInt(c.getColumnIndex(
                                ContactsContract.Contacts.HAS_PHONE_NUMBER));
                if (hasPhone==1){
                    Cursor phoneCursor =
                            getContentResolver().query(

ContactsContract.CommonDataKinds.Phone.CONTENT_URI,null,

ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " +
                                                contactID, null, null);
                    while (phoneCursor.moveToNext()){
                        Log.v("Provajderi sadržaja ", phoneCursor.getString(
```

```
                                phoneCursor.getColumnIndex(

ContactsContract.CommonDataKinds.Phone.NUMBER)));
                    }
                    phoneCursor.close();
                }
            } while (c.moveToNext());
        }
    }

}
```

KORIŠĆENJE UGRAĐENIH PROVAJDERA SADRŽAJA – FUNKCIONISANJE

Pristup provajderu sadržaja promenio se sa pojavom Android vezije Honeycomb.

Postojanje podataka o kontaktima, u mobilnom telefonu ili emulatoru, uslov je da bi ti podaci mogli da budu prikazani. Iz tog razloga, neophodno je, ukoliko je lista kontakata prazna, dodati nekoliko kontakata u provajder sadržaja Contacts. Aplikacija učitava sve kontakte iz aplikacije *Contacts*, a zatim ih prikazuje primenom *ListView* pogleda.

Za pristup aplikaciji *Contacts,* kreiran je URI upit:

Uri allContacts = Uri.parse("content://contacts/people");

U nastavku, proveravana je verzija Android operativnog sistema, na kojem se aplikacija izvršava. Ukoliko je verzija OS starija od verzije Honeycomb (Android API nivo manji od 11) moguće je koristiti menageQuery() metodu za manipulisanje kursorom koji rukuje svim događajima koji se odnose na pauziranje i restartovanje aplikacija.

Novije verzije Android operativnog sistema napuštaju ovu metodu i koriste CursorLoader klasu (videti priloženi kod).

```
CursorLoader cursorLoader = new CursorLoader(
                    this,allContacts,columns,ContactsContract.Contacts.DISPLAY_NAME
+
                    " LIKE ?",
                    new String[] {"V%"},
                    ContactsContract.Contacts.DISPLAY_NAME + " ASC");
            c = cursorLoader.loadInBackground();
```

Ova klasa izvršava upit uz korišćenje kursora u pozadinskoj niti i na taj način ne blokira korisnički interfejs aplikacije. Objekat klase SimpleCursorAdapter povezuje TextView (ili ImageView) poglede definisane u activity_main.xml datoteci. Takođe, kodom je prikazan prevaziđeni konstruktor ovog objekat i novi (sledeća slika) koji se koristi u svim novim verzijama Androida (API nivo 11 i veći). Novi konstruktor koristi Flegza registrovanje adaptera

i na taj način dobija informacije o promenama na strani provajdera sadržaja (pogledati naredni kod).

Takođe, aplikacija zahteva READ_CONTACTS privilegiju, u AndroidManifest.xml datoteci, da bi mogla da pristupi sadržaju provajdera *Contacts*.

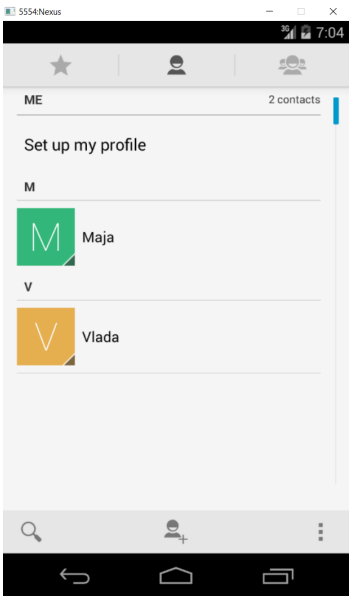
```
/---Honeycomb i novije verzije---
            adapter = new SimpleCursorAdapter(
                    this, R.layout.activity_main, c, columns, views,
                    CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
```

KORIŠĆENJE UGRAĐENIH PROVAJDERA SADRŽAJA – DEMONSTRACIJA

Pokretanjem aplikacije emulatorom prikazuje se lista svih kontakata iz provajdera Contacts.

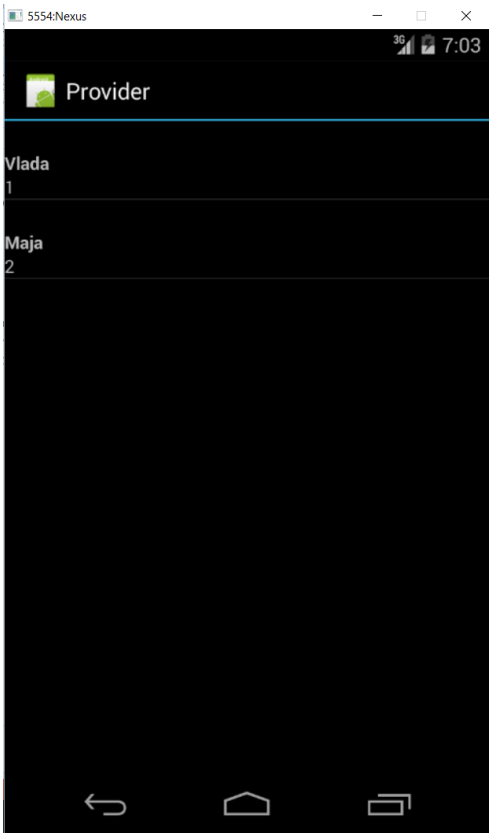
U nastavku je neophodno testirati ugrađenu aplikaciju. U ovu svrhu biće korišćen emulator Genymotion sa AVD instancom koja pokreće verziju Android operativnog sistema 4.4.4 (API verzija 19).

Klikom na Shift + F10 (ili Run app) u Android Studio IDE razvojnom okruženju, aplikacija se prevodi i pokreće emulatorom. Ukoliko je lista kontakata, kreiranog emulatora, prazna, neophodno je uneti nekoliko kontakata primenom aplikacije Contacts (sledeća slika).



Slika 1.2 Dodati kontakti u aplikaciju Contacts AVD instace

Pokretanjem aplikacije emulatorom, ili Android telefonom, prikazuje se lista svih kontakata iz provajdera Contacts (sledeća slika).



Slika 1.3 Prikaz kontakata kreiranom aplikacijom

UGRAĐENE KONSTANTE STRINGA UPITA

Pored URI identifikatora upita, moguće je koristiti i listu ugrađenih konstanti stringa upita za specificiranje URI identifikatora.

Pored URI identifikatora upita, moguće je koristiti i listu ugrađenih konstanti stringa upita za specificiranje URI identifikatora, za različite tipove podataka, u Android aplikacijama. Na primer, sledeće dve naredbe su ekvivalentne:

Uri allContacts = Uri.parse("content://contacts/people");

Uri allContacts = ContactsContract.Contacts.CONTENT_URI;

Slede primeri najčešće korišćenih konstanti za obraćanje provajderima sadržaja:

- ***Browser.BOOKMARKS_URI;***
- ***Browser.SEARCHES_URI;***
- ***CallLOg.CONTENT_URI*** *;*
- *MediaStore.Images.Media.INTERNAL_CONTENT_URI;*
- ***MediaStore.Images.Media.EXTERNAL_CONTENT_URI;***
- *Settings_CONTENT_URI.*

Za očitavanje prvog kontakta, identifikacioni broj se specificira na sledeći način:

Uri allContacts = Uri.parse("content://contacts/people/1");

Kao alternativu, moguće je koristiti redefinisanu konstantu sa metodom *withAppendedID()* klase *ContentUris*:

Uri allContacts = ContentUris. withAppendedID(ContactsContract.Contacts.CONTENT_URI,1)

Podatke je moguće, umesto ListView pogledom, prikazivati i kursorom. Navedeno je prezentovano sledećim listingom.

```
*****
PrintContacts(c);

    }

    private void PrintContacts(Cursor c){
        if (c.moveToFirst()) {
            do{
                String contactID = c.getString(c.getColumnIndex(
                        ContactsContract.Contacts._ID));
                String contactDisplayName =
                        c.getString(c.getColumnIndex(
                                ContactsContract.Contacts.DISPLAY_NAME));
                Log.v("Provajderi sadržaja", contactID + ", " +
                        contactDisplayName);
                //učitavanje broja telefona
                int hasPhone=
                        c.getInt(c.getColumnIndex(
                                ContactsContract.Contacts.HAS_PHONE_NUMBER));
                if (hasPhone==1){
                    Cursor phoneCursor =
                            getContentResolver().query(

ContactsContract.CommonDataKinds.Phone.CONTENT_URI,null,

ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " +
                                                contactID, null, null);
                    while (phoneCursor.moveToNext()){
                        Log.v("Provajderi sadržaja ", phoneCursor.getString(
                                phoneCursor.getColumnIndex(

ContactsContract.CommonDataKinds.Phone.NUMBER)));
                    }
                    phoneCursor.close();
                }
            } while (c.moveToNext());
        }
    }
```

UGRAĐENE KONSTANTE STRINGA UPITA – PRISTUP DODATNIM INFORMACIJAMA

Za preuzimanje dopunskih informacija neophodno je ponovo izvršiti upit nad provajderom sadržaja.

Prethodnim primerom, preuzete su informacije koje se odnose na identifikacioni broj i naziv svakog kontakta iz aplikacije Contacts. Ukoliko se želi preuzimanje još neke informacije, na primer telefonskog broja, neophodno je još jednom izvršiti upit nad provajderom sadržaja (sledeća slika).

Slika 1.4 Preuzimanje broja telefona iz Contacts aplikacije

Prethodni kod, sadržan u proširenoj metodi PrintContacts(), prvo proverava da li kontakt sadrži telefonski broj primenom polja ContactsContract.Contacts.HAS_PHONE_NUMBER. Ukoliko kontakt sadrži bar jedan telefonski broj, upit nad provajderom sadržaja *Contacts* biće ponovo izvršen i preuzeti brojevi mogu da se pročitaju u *LogCat* prozoru kojem je moguće pristupiti iz Android Studio IDE razvojnog okrzženja.

Slika 1.5 LogCat prozor za kreiranu aplikaciju

PROJEKCIJE

Projekcija je parametar kojim se određuje koliko kolona se vraća prilikom izvršavanja upita.

U oba načina, prevaziđena primena metode *manageQuery()* i aktuelna primena klase *CursorLoader*, koriste parametre kojim se određuje koliko kolona se vraća prilikom izvršavanja upita. Ovaj parametar se naziva projekcija. U Aktuelnom primeru, njegova vrednost iznosi *null* i to je prikazano sledećom slikom.

Slika 1.6 Projekcija

Ovaj način, manipulacije podacima, omogućava da se tačno specificira broj kolona koje se vraćaju kada se kreiraju polja koja sadrže nazive kolona. Navedeno je prikazano sledećim kodom.

Slika 1.7 Projekcija vraća tri kolone

FILTRIRANJE I SORTIRANJE

Filtriranje i sortiranje su omogućeni kroz izvršavanje SQL klauzula WHERE i ORDER BY.

Filtriranje je omogućeno kroz izvršavanje SQL klauzule WHERE , a to je određeno trećim i četvrtim parametrom prevaziđene metode *manageQuery* i četvrtim i petim parametrom aktuelnog pristupa koji podrazumeva korišćenje klase *CursorLoader*. Na primer, sledeća naredba omogućava učitavanje samo onih kontakata koji počinu slovom *v*.

Slika 1.8 Filtriranje

Poslednji parametar, oba pristupa, omogućava specificiranje SQL klauzule ORDER BY kojom se realizuje sortiranje rezultata izvršavanja upita. Primena sortiranja je prikazana kodom sa sledeće slike.

Slika 1.9 Sortiranje sadržaja

ZADATAK 1 - PRISTUPITE UGRAĐENOM PROVAJDERU SADRŽAJA

Pokušajte sami

Pokušajte samostalno da kreirate projekat po uzoru na https://android.googlesource.com/platform/development/+/master/samples/RandomMusicPlayer/src/com/example/android/musicplayer/MusicRetriever.java.

Pokrenite i dokumentujte izvršavanje.

Poglavlje 2

Android 6.0 model dozvola

PROBLEM KORIŠĆENJA DOZVOLA

Uvođenjem novih API verija Android operativnog sistema, promenjen je model upravljanja dozvolama.

U prethodnom izlaganju prikazan je tradicionalni Android način pristupa i korišćenja dozvola prilikom upotrebe provajdera sadržaja u kreiranim aplikacijama. Međutim, ukoliko bi prethodna aplikacija bila pokrenuta nekim Android uređajem na kojem je instaliran operativni sistem Android verzije6.0, došlo bi do izvesnih problema prilikom izvršavanja aplikacije. Korisnici bi mogli da uoče otkazivanje aplikacije (slika 2), a ako se pogleda u monitor Android Studio IDE razvojnog okruženja mogli mi da se i pronađu razlozi ovakvog ponašanja kreirane aplikacije (slika 2). Razlog je jednostavan, sa Android 6.0 verzijom operativnog sistema došlo je do promene mehanizama i načina korišćenja dozvola za pristup provajderima sadržaja u Android aplikacijama,

Za demonstraciju novog pristupa korišćenju dozvola, biće kreiran odgovarajući primer koji će, korak po korak, predstavljati probleme i uvoditi novine koje donosi Android 6.0.

Slika 2.1 Info o razlogu otkazivanja aplikacije

Slika 2.2 Otkazivanje apliakcije sa provajderom sadržaja

NOVI MODEL DOZVOLA

Do verzije Android 6.0, sve neophodne dozvole su bile deklarisane u datoteci AndroidManifest.xml.

Novi model dozvola, predstavljen Android 6.0 operativnim sistemom, je veoma važan problem kojeg Android programeri moraju ozbiljno da uzmu u razmatranje. Sve do verzije Android 6.0, sve neophodne dozvole su bile deklarisane u datoteci AndroidManifest.xml i fokus programera je bio na implementaciji poslovne logike. U međuvremenu, Android je postajao sve kompleksniji i donosio je više bezbednosti i kontrole nad aplikacijama prema krajnjem korisniku.

Kada se prvi put pokrene bilo koja Android aplikacija, Android 6 verzija operativnog sistema reaguje dijalog okvirima kojima se obraća korisniku aplikacije i sistema sa ciljem potvrđivanja izvesne dozvole korišćenja aplikacije. Navedeno je pokazano sledećom slikom.

Slika 2.3 Dozvola čitanja konatakata (izvor: http://www.javahelps.com)

Na sličan način, kao u prethodnom primeru, biće pristupljeno problemu čitanja kontakata iz provajdera sadržaja Contacts i biće povlačene paralele između novog i starog pristupa.

Za početak, biće kreiran nov projekat pod nazivom ProvajderAnd6Demo i u prvom koraku će u njegovoj AndroidManifest.xml datoteci biti omogućeno posebnom dozvolom čitanje iz kontakata. Datoteka je data sledećim listingom.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.metropolitan.provajderand6demo">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

PRIMER 2 - DATOTEKE PROJEKTA ZA UPRAVLJANJE DOZVOLAMA

U narednom koraku je neophodno implementirati odgovarajuće datoteke kreiranog projekta.

Nijedna Android aplikacije ne može da funkcioniše bez korisničkog interfejsa pa će u narednom koraku biti, upravo, definisana i kreirana GUI XML datoteka. Neka ova datoteka omogući da glavna klasa aktivnosti učita pogled sa listom. Navedeno je dato sledećim listingom.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/lstNames"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

U nastavku akcenat se prenosi na glavnu klasu aktivnosti. U prvom koraku, u implementaciji podrazumevane metode onCreate() za pronalaženje GUI ***ListView*** komponente, biće neophodno kreirati instancu ***ListView*** pogleda, kao i poziv metode za čitanje i prikazivanje kontakata. Navedeno je prikazano sledećim listingom. Budući da je već pričano o čitanju kontakata, kod će biti priložen i neće biti posebno obrazlagan.

```
import android.content.ContentResolver;
import android.database.Cursor;
import android.provider.ContactsContract;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
    // ListView
    private ListView lstNames;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Find the list view
        this.lstNames = (ListView) findViewById(R.id.lstNames);
        // Read and show the contacts
        showContacts();
    }
    /**
     * Prikazuje kontakte u ListView.
     */
    private void showContacts() {
        List<String> contacts = getContactNames();
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, contacts);
        lstNames.setAdapter(adapter);
    }
    /**
```

```
     * Čita nazive svih konatakata
     *
     * Vraća listu sa imenima.
     */
    private List<String> getContactNames() {
        List<String> contacts = new ArrayList<>();
        // Uzima ContentResolver
        ContentResolver cr = getContentResolver();
        // Get the Cursor of all the contacts
        Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null,
null, null);
        // Pomera kursor na prvog i proverava da li je prazan ili ne
        if (cursor.moveToFirst()) {
            // Iterira kroz kursor
            do {
                // Uzima naziv kontakta
                String name =
cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
                contacts.add(name);
            } while (cursor.moveToNext());
        }
        // Zatvara kursor
        cursor.close();
        return contacts;
    }
}
```

KATEGORIJE DOZVOLA

Android 6.0 uvodi dve kategorije dozvola

Pokretanjem ovako kreirane aplikacije na emulatoru koji je definisan sa Android verzijom generacije 5 ili niže, program će normalno da se ponaša i izvršiće se veoma slično prethodno pokazanom primeru. Ako se aplikacija izvrši na Android 6.0 operativnom sistemu, javiće se pomenute greške u izvršavanju programa. U monitoru Android Studio IDE okruženja, moguće je primetiti da je dozvola pristupa čitanju kontakata odbijena iako je deklarisana u AndroidManifest.xml datoteci. Razlog je što Android 6.0 insistira na modelu upravljanja dozvolama tokom vremena izvršavanja aplikacije (runtime permission model.).

Android 6.0 nudi dve kategorije dozvola:

- Normalne dozvole;
- Opasne dozvole.

Normalne dozvole pokrivaju delove aplikacije u kojima je mali, ili ne postoji, rizik narušavanja korisnikove privatnosti ili operacija drugih aplikacija. Na primer, dozvola korišćenja blica kamere je normalna dozvola. Upravljanje ovom kategorijom dozvola je nepromenjeno u odnosu na ranije verzije Android operativnog sistema. Ukoliko je sistem pronađe, u AndroidManifest.xml datoteci, automatski je odobrava.

Opasne dozvole se odnose na delove aplikacije koje rukuju privatnim podacima ili nose potencijalne efekte na sačuvane podatke i operacije drugih aplikacija..

Na primer, čitanje kontakata iz provajdera sadržaja Contacts smatra se opasnom dozvolom po Android 6.0 operativnom sistemu. Ukoliko sistem naiđe na takvu dozvolu, od korisnika će biti zatraženo da je potvrdi. Upravo ovu novinu će biti neophodno implementirati u kreiranoj aplikaciji. To znači, budući da je ***READ_CONTACTS*** opasna dozvola, neophodno je u glavnu klasu aktivnosti dodati kod kojim će korisniku biti omogućeno da potvrdi dozvolu tokom vremena izvršavanja aplikacije.

U tu svrhu će biti neophodno dodati, u postojeći kod, konstantu ***PERMISSIONS_REQUEST_READ_CONTACTS*** sa definisanim prioritetom (broj je veći od nule, što je manji prioritet je viši). Takođe, neophodno je izvršiti i redefinisanje metode showContacts().

```
// Traži kod za čitanje kontakata
    private static final int PERMISSIONS_REQUEST_READ_CONTACTS = 100;

    /**
     * SPrikazuje kontakte u pogledu ListView.
     */
    private void showContacts() {
        // Proverava SDK verziju
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;
checkSelfPermission(Manifest.permission.READ_CONTACTS) !=
PackageManager.PERMISSION_GRANTED) {
            requestPermissions(new String[]{Manifest.permission.READ_CONTACTS},
PERMISSIONS_REQUEST_READ_CONTACTS);

        } else {
            // Android verzija je niža od Android 6 i dozvola se automatski
dodeljuje
            List<String> contacts = getContactNames();
            ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, contacts);
            lstNames.setAdapter(adapter);
        }
    }
```

KATEGORIJE DOZVOLA - DODATNA RAZMATRANJA

Još jedna novina je korišćenje metode checkSelfPermission() čiji je zadatak da proveri u sistemu da li je korisnik već odobrio izvesnu dozvolu ili ne.

Iz priloženog koda, redefinisane metode showContacts(), moguće je primetiti da se prvo vrši provera SDK verzije Android operativnog sistema. Ukoliko je SDK verzija veća ili jednaka Android 6.0 verziji, dozvola nije odobrena i prosleđuje se korisniku na potvrđivanje. Još jedna

novina je korišćenje metode checkSelfPermission() čiji je zadatak da proveri u sistemu da li je korisnik već odobrio izvesnu dozvolu ili ne. Ukoliko je SDK verzija niža od Android 6.0 verzije i dozvola postoji u AndroidManifest.xml datoteci, smatraće se odobrenom.

Ukoliko korisnik nije već odobrio dozvolu, Android će mu proslediti upit za odobravanje. Navedeno je realizovano implementacijom nove metode, glavne klase aktivnosti, pod nazivom onRequestPermissionsResult(). Implementacioni kod ove metode je priložen sledećim listingom.

```
@Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                           int[] grantResults) {
        if (requestCode == PERMISSIONS_REQUEST_READ_CONTACTS) {
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // Dozvola je potvrđena
                showContacts();
            } else {
                Toast.makeText(this, "Dok ne potvrdite dozvolu, ne mogu da prikažem
kontakte", Toast.LENGTH_SHORT).show();
            }
        }
    }
```

Na ovaj način, zaokružena je definicija glavne klase aktivnosti koja upravlja dozvolama na način koji zahtevaju najnoviji Android standardi predstavljeni verzijom operativnog sistema Android 6.0. Kompletirana klasa aktivnosti projekta, priložena je sledećim listingom:

```
package com.metropolitan.provajderand6demo;

import android.Manifest;
import android.content.ContentResolver;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.os.Build;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.support.v7.app.AppCompatActivity;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
    // ListView
    private ListView lstNames;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Find the list view
```

```
        this.lstNames = (ListView) findViewById(R.id.lstNames);
        // Read and show the contacts
        showContacts();
    }

    // Traži kod za čitanje kontakata
    private static final int PERMISSIONS_REQUEST_READ_CONTACTS = 100;

    /**
     * SPrikazuje kontakte u pogledu ListView.
     */
    private void showContacts() {
        // Proverava SDK verziju
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;am
p;amp;amp;amp; checkSelfPermission(Manifest.permission.READ_CONTACTS) !=
PackageManager.PERMISSION_GRANTED) {
            requestPermissions(new String[]{Manifest.permission.READ_CONTACTS},
PERMISSIONS_REQUEST_READ_CONTACTS);

        } else {
            // Android verzija je niža od Android 6 i dozvola se automatski
dodeljuje
            List<String> contacts = getContactNames();
            ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, contacts);
            lstNames.setAdapter(adapter);
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                           int[] grantResults) {
        if (requestCode == PERMISSIONS_REQUEST_READ_CONTACTS) {
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // Dozvola je potvrđena
                showContacts();
            } else {
                Toast.makeText(this, "Dok ne potvrdite dozvolu, ne mogu da prikažem
kontakte", Toast.LENGTH_SHORT).show();
            }
        }
    }

    /**
     * Čita nazive svih konatakata
     *
     * Vraća listu sa imenima.
     */
    private List<String> getContactNames() {
        List<String> contacts = new ArrayList<>();
        // Uzima ContentResolver
        ContentResolver cr = getContentResolver();
```

```
        // Get the Cursor of all the contacts
        Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null,
null, null);
        // Pomera kursor na prvog i proverava da li je prazan ili ne
        if (cursor.moveToFirst()) {
            // Iterira kroz kursor
            do {
                // Uzima naziv kontakta
                String name =
cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
                contacts.add(name);
            } while (cursor.moveToNext());
        }
        // Zatvara kursor
        cursor.close();
        return contacts;
    }
}
```

DEMONSTRACIJA UPRAVLJANJA DOZVOLAMA U ANDROID 6.0.

Kreirana aplikacija se testira u Android 6.0 emulatoru ili mobilnom uređaju.

Sa ciljem demonstriranja upravljanja dozvolama u Android 6.0 okruženju, neophodno je izvršiti debagovanje i pokretanje kreirane Android aplikacije. U Android Studio IDE razvojnom okruženju, izborom opcije Run App (ili Shift + F10) prevodi se i pokreće kreirana aplikacija. Aplikacija prvi put zahteva preuzimanje i čitanje kontakata iz provajdera sadržaja Contacts. Upravo zbog toga korisnik dobija upit da potvrdi dozvolu. Navedeno je prikazano sledećom slikom.

Slika 2.4 Potvrđivanje dozvole pristupa u Android 6.0 operativnom sistemu

Korisnik može da odobri (ALLOW) ili da odbije (DENY) dozvolu. U slučaju prihvatanja, dozvola prelazi iz kategorije opasne u normalnu i kreirana aplikacija, u konkretnom slučaju, može da radi sa kontaktima. Navedeno je prikazano sledećom slikom.

Slika 2.5 Dozvola za čitanje kontakata je prihvaćena.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER 3 - UPRAVLJANJE DOZVOLAMA I PRISTUP PROVAJDERU

Vežbanje upravljanja pristupom provajderu sadržaja.

Zadatak:

1. Koristiti Android Studio IDE i emulator pod Android 6.0 operativnim sistemom;
2. Aplikacija se obraća provajderu sadržaja, može kontaktima - koji ima dozvolu pristupa iz kategorije opasna;
3. Realizovati po Android 6.0 standardima upravljanje dozvolama;
4. Na ekranu prikazati analogni časovnik i kao pozadinu postaviti logo Metropolitan Univerziteta (primer na slici).

Slika 2.6 Izgled ekrana

Datoteka korisničkog interfejsa je sledeća:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:background="@drawable/metpozadina">

    <ListView
        android:id="@+id/lstNames"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true" />

</RelativeLayout>
```

U AndroidManifest.xml je neophodno dodati sledeću dozvolu:

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

Klasa aktivnosti ima sledeći oblik:

```
package com.metropolitan.provajderand6demo;

import android.Manifest;
import android.content.ContentResolver;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.os.Build;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.support.v7.app.AppCompatActivity;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
    // ListView
    private ListView lstNames;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Find the list view
        this.lstNames = (ListView) findViewById(R.id.lstNames);
        // Read and show the contacts
        showContacts();
    }

    // Traži kod za čitanje kontakata
    private static final int PERMISSIONS_REQUEST_READ_CONTACTS = 100;

    /**
     * SPrikazuje kontakte u pogledu ListView.
     */
    private void showContacts() {
        // Proverava SDK verziju
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M
&amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;
checkSelfPermission(Manifest.permission.READ_CONTACTS) !=
PackageManager.PERMISSION_GRANTED) {
            requestPermissions(new String[]{Manifest.permission.READ_CONTACTS},
PERMISSIONS_REQUEST_READ_CONTACTS);

        } else {
            // Android verzija je niža od Android 6 i dozvola se automatski
dodeljuje
            List<String> contacts = getContactNames();
            ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, contacts);
            lstNames.setAdapter(adapter);
        }
```

```
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                           int[] grantResults) {
        if (requestCode == PERMISSIONS_REQUEST_READ_CONTACTS) {
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // Dozvola je potvrđena
                showContacts();
            } else {
                Toast.makeText(this, "Dok ne potvrdite dozvolu, ne mogu da prikažem
kontakte", Toast.LENGTH_SHORT).show();
            }
        }
    }

    /**
     * Čita nazive svih konatakata
     *
     * Vraća listu sa imenima.
     */
    private List<String> getContactNames() {
        List<String> contacts = new ArrayList<>();
        // Uzima ContentResolver
        ContentResolver cr = getContentResolver();
        // Get the Cursor of all the contacts
        Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null,
null, null);
        // Pomera kursor na prvog i proverava da li je prazan ili ne
        if (cursor.moveToFirst()) {
            // Iterira kroz kursor
            do {
                // Uzima naziv kontakta
                String name =
cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
                contacts.add(name);
            } while (cursor.moveToNext());
        }
        // Zatvara kursor
        cursor.close();
        return contacts;
    }
}
```

ZADATAK 2 - UPRAVLJAJTE DOZVOLAMA U ANDROID APLIKACIJI

Pokušajte sami

Kreirajte Android projekat i omogućite:

1. Upravljanje baren jednom standardnom dozvolom;

2 . Upravljanje barem jednom dozvolom iz kategorije "opasnih".

Poglavlje 3

Kreiranje provajdera sadržaja

KLASA VLASTITOG PROVAJDERA SADRŽAJA

Klasa provajdera sadržaja je naslednica klase ContentProvider.

Kreiranje vlastitog provajdera sadržaja je, u osnovi, veoma jednostavno. Neophodno je implementirati klasu, koja nasleđuje iz apstraktne klase *ContentProvider* i potom definisati različite metode te klase. Kao primer, biće kreiran provajder sadržaja koji skladišti knjige u tabeli baze podataka. Tabela sadrži tri polja:

- *_id;*
- *naslov;*
- *isbn.*

Za kreiranje baze podataka biće iskorišćen alat za upravljanje bazama podataka SQLite Browser (slika 1).

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Sledećom slikom prikazana je tabela baze podataka u kojoj će kreirani provajder sadržaja skladištiti knjige. Slika pokazuje izgled programa SQLite Browser u kojem je moguće primetiti kreiranu bazu podataka sa pripadajućim tabelama, kao i alat za kreiranje i testiranje SQL upita nad kreiranom SQLite bazom podataka.

SQLite Browser je moguće preuzeti sa linka http://sqlitebrowser.org/.

Slika 3.1 Tabela baze podataka

KREIRANJE KLASE PROVAJDERA SADRŽAJA

Prvi zadatak je kreiranje prazne klase provajdera sadržaja.

U prvom koraku biće kreirana klasa provajdera sadržaja bez koje realizacija ovog projekta ne bi bila moguća. Sastavni deo klase provajdera predstavljaće klasa DataBaseHelper, koja nasleđuje osnovnu klasu SQLiteOpenHelper i poseduje mehanizme za kreiranje baze podataka. Inicijalni kod klase prikazan je sledećim listingom i u narednom izlaganju biće proširivan kroz uvođenje novih funkcionalnosti u aplikaciju.

```
public class BooksProvider extends ContentProvider
{
```

```
    static final String PROVIDER_NAME =
            "com.metropolitan.vlastitiprovajderdemo.Books";

    static final Uri CONTENT_URI =
            Uri.parse("content://"+ PROVIDER_NAME + "/books");

    static final String _ID = "_id";
    static final String TITLE = "naslovi";
    static final String ISBN = "isbn";

    static final int BOOKS = 1;
    static final int BOOK_ID = 2;

    private static final UriMatcher uriMatcher;
    static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "books", BOOKS);
        uriMatcher.addURI(PROVIDER_NAME, "books/#", BOOK_ID);
    }
//---kreiranje baze podataka---
    SQLiteDatabase booksDB;
    static final String DATABASE_NAME = "Books";
    static final String DATABASE_TABLE = "knjige";
    static final int DATABASE_VERSION = 1;
    static final String DATABASE_CREATE =
            "create table " + DATABASE_TABLE +
                    " (_id integer primary key autoincrement, "
                    + "naslovi text not null, isbn text not null);";

private static class DatabaseHelper extends SQLiteOpenHelper
    {
        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db)
        {
            db.execSQL(DATABASE_CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion,
                              int newVersion) {
            Log.w("Baza podataka provajera sadržaja",
                    "Ažuriranje verzije sa " +
                            oldVersion + " na verziju " + newVersion +
                            ", stari podaci biće uništeni");
            db.execSQL("DROP TABLE IF EXISTS titles");
            onCreate(db);
        }
    }
  }
```

KLASA PROVAJDERA SADRŽAJA – METODE GETTYPE(), ONCREATE(), QUERY()

Klasu provajdera sadržaja je neophodno proširiti metodama za CRUD podršku.

Pored podrške za kreiranje baze podataka, klasu provajdera sadržaja je neophodno snabdeti metodama koje obezbeđuju izvođenje operacija nad bazom podataka. Prvo će biti implementirane metode getType(), onCreate() i query().

Metoda getType() dopunjuje kod klase provajdera sadržaja donoseći sledeće linije programskog koda.

```
@Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)){
            //---preuzimanje svih knjiga---
            case BOOKS:
                return "vnd.android.cursor.dir/vnd.metropolitan.books ";

            //---preuzimanje jedne knjige---
            case BOOK_ID:
                return "vnd.android.cursor.item/vnd.metropolitan.books ";

            default:
                throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
    }
```

Potom je moguće pristupiti proširivanju klase provajdera sadržaja kodom koji odgovara implementaciji metode onCreate(). Taj kod je priložen narednim listingom.

```
@Override
    public boolean onCreate() {
        Context context = getContext();
        DatabaseHelper dbHelper = new DatabaseHelper(context);
        booksDB = dbHelper.getWritableDatabase();
        return (booksDB == null)? false:true;
    }
```

Potom, klasa provajdera sadržaje se proširuje i kodom koji odgovara implementaciji metode query(). Kod je priložen sledećim listingom.

```
@Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
        sqlBuilder.setTables(DATABASE_TABLE);

        if (uriMatcher.match(uri) == BOOK_ID)
```

```
            //---ako se pruzima konkretna knjiga---
            sqlBuilder.appendWhere(
                    _ID + " = " + uri.getPathSegments().get(1));

        if (sortOrder==null || sortOrder=="")
            sortOrder = TITLE;

        Cursor c = sqlBuilder.query(
                booksDB,
                projection,
                selection,
                selectionArgs,
                null,
                null,
                sortOrder);

        //---registrovanje praćenja promena URI identifikatora---
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }
```

KLASA PROVAJDERA SADRŽAJA – METODE INSERT(), DELETE() I UPDATE().

Definicija klase proširuje se sa još tri metode.

Pored priloženih metoda getType(), onCreate() i query(), neophodno je dodati i metode insert(), delete() i update(). Upravo ovim metodama su realizovane operacije nad bazom podataka.

Dodavanje novih knjiga u bazu podataka realizovano je metodom insert(). Kod ove metode, koji proširuje inicijalni kod klase provajdera priložen je sledećim listingom.

```
@Override
    public Uri insert(Uri uri, ContentValues values) {
        //---dodaje novu knjigu---
        long rowID = booksDB.insert(
                DATABASE_TABLE,
                "",
                values);

        //---ako je dodavanje uspešno---
        if (rowID>0)
        {
            Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
            getContext().getContentResolver().notifyChange(_uri, null);
            return _uri;
        }
        throw new SQLException("Dodavanje u vrstu nije uspelo " + uri);
    }
```

Brisanje knjiga iz baze podataka je povereno metodi delete(). Kod ove metode, koji proširuje inicijalni kod klase provajdera priložen je sledećim listingom.

```
@Override
    public int delete(Uri arg0, String arg1, String[] arg2) {
        // arg0 = uri
        // arg1 = selection
        // arg2 = selectionArgs
        int count=0;
        switch (uriMatcher.match(arg0)){
            case BOOKS:
                count = booksDB.delete(
                        DATABASE_TABLE,
                        arg1,
                        arg2);
                break;
            case BOOK_ID:
                String id = arg0.getPathSegments().get(1);
                count = booksDB.delete(
                        DATABASE_TABLE,
                        _ID + " = " + id +
                                (!TextUtils.isEmpty(arg1) ? " AND (" +
                                        arg1 + ')' : ""),
                        arg2);
                break;
            default: throw new IllegalArgumentException("Unknown URI " + arg0);
        }
        getContext().getContentResolver().notifyChange(arg0, null);
        return count;
    }
```

Na kraju, definicija klase provajdera je zaokružena metodom update(), za ažuriranje sadržaja baze podataka, čiji je kod priložen sledećim listingom.

```
@Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)){
            case BOOKS:
                count = booksDB.update(
                        DATABASE_TABLE,
                        values,
                        selection,
                        selectionArgs);
                break;
            case BOOK_ID:
                count = booksDB.update(
                        DATABASE_TABLE,
                        values,
                        _ID + " = " + uri.getPathSegments().get(1) +
                                (!TextUtils.isEmpty(selection) ? " AND (" +
                                        selection + ')' : ""),
```

```
                        selectionArgs);
                break;
            default: throw new IllegalArgumentException("Nepoznat URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
```

PREGLED OBJEDINJENE KLASE PROVAJDERA

Klasa provajdera sadržaja je snabdevena metodama za izvođenje operacija nad SQLite bazom podataka.

U prethodnom izlaganju je pokazano kako se u inicijalno definisani kod klase provajdera dodaju metode koje imaju konkretne zadatke i kojima se funkcionalnost ove klase proširuje. Kompletirana klasa provajdera sadržaja sa svim navedenim metodama i ugrađenom pomoćnom klasom, za pomoć pri obraćanju i rukovanju SQLite bazom podataka, priložena je sledećim listingom. Posebno, pokazane su i ***import*** instrukcije neophodne za implementaciju ove klase.

```
package com.metropolitan.vlastitiprovajderdemo;


import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;
import android.util.Log;

/**
 * Created by Vladimir Milicevic on 9.11.2016..
 */

public class BooksProvider extends ContentProvider
{
    static final String PROVIDER_NAME =
            "com.metropolitan.vlastitiprovajderdemo.Books";

    static final Uri CONTENT_URI =
            Uri.parse("content://"+ PROVIDER_NAME + "/books");
```

```
    static final String _ID = "_id";
    static final String TITLE = "naslovi";
    static final String ISBN = "isbn";

    static final int BOOKS = 1;
    static final int BOOK_ID = 2;

    private static final UriMatcher uriMatcher;
    static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "books", BOOKS);
        uriMatcher.addURI(PROVIDER_NAME, "books/#", BOOK_ID);
    }

    //---kreiranje baze podataka---
    SQLiteDatabase booksDB;
    static final String DATABASE_NAME = "Books";
    static final String DATABASE_TABLE = "knjige";
    static final int DATABASE_VERSION = 1;
    static final String DATABASE_CREATE =
            "create table " + DATABASE_TABLE +
                    " (_id integer primary key autoincrement, "
                    + "naslovi text not null, isbn text not null);";

    private static class DatabaseHelper extends SQLiteOpenHelper
    {
        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db)
        {
            db.execSQL(DATABASE_CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion,
                              int newVersion) {
            Log.w("Baza podataka provajera sadržaja",
                    "Ažuriranje verzije sa " +
                            oldVersion + " na verziju " + newVersion +
                            ", stari podaci biće uništeni");
            db.execSQL("DROP TABLE IF EXISTS titles");
            onCreate(db);
        }
    }

    @Override
    public int delete(Uri arg0, String arg1, String[] arg2) {
        // arg0 = uri
        // arg1 = selection
        // arg2 = selectionArgs
```

```
        int count=0;
        switch (uriMatcher.match(arg0)){
            case BOOKS:
                count = booksDB.delete(
                        DATABASE_TABLE,
                        arg1,
                        arg2);
                break;
            case BOOK_ID:
                String id = arg0.getPathSegments().get(1);
                count = booksDB.delete(
                        DATABASE_TABLE,
                        _ID + " = " + id +
                                (!TextUtils.isEmpty(arg1) ? " AND (" +
                                        arg1 + ')' : ""),
                        arg2);
                break;
            default: throw new IllegalArgumentException("Unknown URI " + arg0);
        }
        getContext().getContentResolver().notifyChange(arg0, null);
        return count;
    }

    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)){
            //---preuzimanje svih knjiga---
            case BOOKS:
                return "vnd.android.cursor.dir/vnd.metropolitan.books ";

            //---preuzimanje jedne knjige---
            case BOOK_ID:
                return "vnd.android.cursor.item/vnd.metropolitan.books ";

            default:
                throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        //---dodaje novu knjigu---
        long rowID = booksDB.insert(
                DATABASE_TABLE,
                "",
                values);

        //---ako je dodavanje uspešno---
        if (rowID>0)
        {
            Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
            getContext().getContentResolver().notifyChange(_uri, null);
            return _uri;
```

```
        }
        throw new SQLException("Dodavanje u vrstu nije uspelo " + uri);
    }

    @Override
    public boolean onCreate() {
        Context context = getContext();
        DatabaseHelper dbHelper = new DatabaseHelper(context);
        booksDB = dbHelper.getWritableDatabase();
        return (booksDB == null)? false:true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
        sqlBuilder.setTables(DATABASE_TABLE);

        if (uriMatcher.match(uri) == BOOK_ID)
            //---ako se pruzima konkretna knjiga---
            sqlBuilder.appendWhere(
                    _ID + " = " + uri.getPathSegments().get(1));

        if (sortOrder==null || sortOrder=="")
            sortOrder = TITLE;

        Cursor c = sqlBuilder.query(
                booksDB,
                projection,
                selection,
                selectionArgs,
                null,
                null,
                sortOrder);

        //---registrovanje praćenja promena URI identifikatora---
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)){
            case BOOKS:
                count = booksDB.update(
                        DATABASE_TABLE,
                        values,
                        selection,
                        selectionArgs);
                break;
            case BOOK_ID:
```

```
                count = booksDB.update(
                        DATABASE_TABLE,
                        values,
                        _ID + " = " + uri.getPathSegments().get(1) +
                                (!TextUtils.isEmpty(selection) ? " AND (" +
                                        selection + ')' : ""),
                        selectionArgs);
                break;
            default: throw new IllegalArgumentException("Nepoznat URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
}
```

PROVAJDER SADRŽAJA – ANDROIDMANIFEST.XML DATOTEKA

Provajdera sadržaja je neophodno uključiti AndroidManifest.xml datotekom.

Da bi rad sa provajderom sadržaja bio moguć, pored kreiranja klase provajdera sadržaja, neophodno je napraviti i izvesne modifikacije u datoteci AndroidManifest.xml. Konkretne izmene se odnose na dodavanje vlastitog provajdera sadržaja u projekat. Koristeći XML tag*<provider> ... </provider>,* ovom XML datotekom se uključuje provajder sadržaja. To je pokazano sedećim segmentom XML koda.

```
<provider android:name="BooksProvider"

android:authorities="com.metropolitan.vlastitiprovajderdemo.provider.Books">
        </provider>
```

Kada se priloženi kod ugradi u polazni kod koji odgovara AndroidManifest.xml datoteci, dobija se sledeći listing.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.metropolitan.vlastitiprovajderdemo">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

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

android:authorities="com.metropolitan.vlastitiprovajderdemo.provider.Books">
        </provider>
    </application>

</manifest>
```

NAČIN FUNKCIONISANJA KLASE PROVAJDERA SADRŽAJA

Funkcionisanje klase provajdera sadržaja bazira se na funkcionalnostima predefinisanih metoda osnovne klase ContentProvider.

Nakon prvog koraka, kreiranja klase naslednice osnovne klase ContentProvider, redefinisano je nekoliko metoda bazne klase:

- getType() – vraća MIME tip sa odgovarajućim URI;
- onCreate()– izvršava se sa pokretanjem provajdera;
- query() – učitava zahtev klijenta i vraća *Cursor* objekat;
- insert() – unosi novi zapis u provajder sadržaja;
- delete() – briše zapis pomoću provajdera sadržaja;
- update() – koristi provajder sadržaja za ažuriranje zapisa.

U inicijalnom kodu klase moguće je primetiti da je korišćen objekat UriMatcher (videti kod) za analiziranje sadržaja URI identifikatora koji je prosleđen provajderu objektom ContentResolver. Na primer, sledećim URI identifikatorima su zatraženi zahtevi za učitavanjem svih knjiga i knjige čiji identifikator ima vrednost 1, respektivno.

Uri.parse("content://"+ PROVIDER_NAME + "/books"); Uri.parse("content://"+ PROVIDER_NAME + "/books/1");

Provajder koristi SQLite bazu podataka pa je iskorišćena SQLiteHelper pomoćna klasa za lakše upravljanje bazom podataka (videti kod).

Redefinisanjem metode getType() tako što mu se predaju *URI* objekat, dobijen je unikatan način za opisivanje tipa podataka za provajder sadržaja. Primenom UriMatcherobjekta više knjiga se učitava pomoću: *vnd.android.cursor.dir/vnd.learn2develop.books*, a pojedinačne knjige pomoću: ***vnd.android.cursor.item/vnd.metropolitan.books*** *(videti kod metode getType).*

U sledećem koraku redefinisana je onCreate()metoda sa ciljem omogućavanja konekcije sa bazom podataka nakon pokretanja provajdera sadržaja (videti priloženi kod metode).

Takođe, redefinisana je i query() metoda kojom je omogućeno klijentima da postavljaju upite za knjige. Metoda je podešena tako da se rezultat upita vraća kao tip Cursor, sortiran po polju TITLE.

NAČIN FUNKCIONISANJA KLASE PROVAJDERA SADRŽAJA - NASTAVAK

Unošenje nove knjige, brisanje iz baze i ažuriranje omogućeno je predefinisanjem metoda insert(), delete() i update().

Da bi nov podatak bio unešen u bazu podataka, primenom provajdera sadržaja, neophodno je koristiti redefinisanu metoduinsert() koja preuzima kao argumente dva objekta Uri i ContatntValues. Kao rezultat, metoda vraća tip podataka Uri. Nakon obavljenog dodavanja, novog zapisa u bazu, izvršava se metoda notifyChange() objekta klase ContentResolver(videti priloženi kod).

Za uklanjanje zapisa, u konkretnom slučaju knjige, iz baze podataka, a primenom provajdera sadržaja, redefinisana je i upotrebljena metoda delete() iz osnovne klase koju nasleđuje klasa provajdera sadržaja. Takođe, metoda delete(), omogućava da se izvrši metoda notifyChange() objekta klase ContentResolver(videti priloženi kod) nakon izvršenog uklanjanja podataka. Na ovaj način se obaveštavaju registrovani posmatrači da je obrisana odgovarajuća vrsta.

U nastavku je redefinisana i iskorišćena metoda update() koja, slično kao i prethodne dve metode, izvršava metodu notifyChange() objekta klase ContentResolver(videti priloženi kod). Kroz ovu akciju obaveštavaju se registrovani posmatrači da je ažurirana odgovarajuća vrsta.

Na samom kraju, da bi provajder sadržaja bio registrovan i angažovan u Android sistemu, modifikovana je datoteka AndroidManifest.xml dodavanjem XML elementa <provider> .

KLASA AKTIVNOSTI I UPOTREBA PROVAJDERA SADRŽAJA.

Nakon definisanja provajdera sadržaja neophodno je omogućiti mehanizme implementacije u Android aplikaciji.

Za primenu kreiranog provajdera sadržaja, u Android aplikaciji, neophodno je kreirati klasu aktivnosti i korisnički interfejs aplikacije preko kojeg će korisnik i aplikacija komunicirati. Klasom aktivnosti je učitan glavni korisnički interfejs i omogućeno pokretanje same aplikacije za demonstraciju rada sa vlastitim provajderima sadržaja. Sledećim listingom dat je kod odgovarajuće klase aktivnosti koja omogućava angažovanje provajdera sadržaja.

```
package com.metropolitan.vlastitiprovajderdemo;

/**
 * Created by Vladimir Milicevic on 9.11.2016..
 */
```

```
import android.support.v7.app.AppCompatActivity;
import android.content.ContentValues;
import android.content.CursorLoader;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
//import android.provider.ContactsContract;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    /** Poziva se kada se aktivnost kreira. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClickAddTitle(View view) {
        /*
        //---dodaj knjigu---
        ContentValues values = new ContentValues();
        values.put(BooksProvider.TITLE, ((EditText)
                findViewById(R.id.txtTitle)).getText().toString());
        values.put(BooksProvider.ISBN, ((EditText)
                findViewById(R.id.txtISBN)).getText().toString());
        Uri uri = getContentResolver().insert(
                BooksProvider.CONTENT_URI, values);
         */
        ContentValues values = new ContentValues();
        values.put("naslovi", ((EditText)
                findViewById(R.id.txtTitle)).getText().toString());
        values.put("isbn", ((EditText)
                findViewById(R.id.txtISBN)).getText().toString());
        Uri uri = getContentResolver().insert(

Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books"),
                values);

        Toast.makeText(getBaseContext(),uri.toString(),
                Toast.LENGTH_LONG).show();
    }

    public void onClickRetrieveTitles(View view) {
        //---vraća naslov---
        Uri allTitles =
Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books");
        Cursor c;
        CursorLoader cursorLoader = new CursorLoader(this,allTitles, null, null,
                null,"naslovi desc");
```

```
        c = cursorLoader.loadInBackground();
        if (c.moveToFirst()) {
            do{
                Toast.makeText(this,
                        c.getString(c.getColumnIndex(
                                BooksProvider._ID)) + ", " +
                                c.getString(c.getColumnIndex(
                                        BooksProvider.TITLE)) + ", " +
                                c.getString(c.getColumnIndex(
                                        BooksProvider.ISBN)),
                        Toast.LENGTH_SHORT).show();
            } while (c.moveToNext());
        }
    }

    public void updateTitle() {
        ContentValues editedValues = new ContentValues();
        editedValues.put(BooksProvider.TITLE, "Android Tipovi i trikovi.");
        getContentResolver().update(

Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books/
2"),
                editedValues,null,null);
    }

    public void deleteTitle() {

        //---brisanje naslova---
        getContentResolver().delete(

Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books/
2"),
                null, null);
        //---brisanje svih naslova---
        getContentResolver().delete(

Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books"),
                null, null);
    }


}
```

XML DATOTEKA KORISNIČKOG INTERFEJSA ZA IMPLEMENTACIJU PROVAJDERA SADRŽAJA.

Aplikacija za primenu provajdera sadržaja kompletirana je definicijom korisničkog interfejsa.

U folderu projekta, podfolder *res/layout*, bira se XML datoteka activity_main.xml i menjaju se njena osnovna podešavanja dodavanjem sledećeg koda koji odgovara elementima korisničkog interfejsa i njihovom rasporedu na ekranu. Novom definicijom ove datoteke, kompletirana je aplikacija za demonstraciju primene vlastitog provajdera sadržaja. Aplikacija sada, primenom razvojnog okruženja Android Studio IDE može da bude dabagovana i pokrenuta na izabranom emulatoru za testiranje.

Sledećim kodom data je activity_main.xml datoteka.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="ISBN" />
    <EditText
        android:id="@+id/txtISBN"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Naslov" />
    <EditText
        android:id="@+id/txtTitle"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />
    <Button
        android:text="Dodaj naslov"
        android:id="@+id/btnAdd"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onClickAddTitle" />
    <Button
        android:text="Preuzmi naslove"
        android:id="@+id/btnRetrieve"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onClickRetrieveTitles"  />
</LinearLayout>
```

FUNKCIONISANJE APLIKACIJE ZA ANGAŽOVANJE PROVAJDERA SADRŽAJA

Funkcionisanje programa počinje izvršavanjem metode onCreate() i učitavanjem UI za upravljanje provajderom sadržaja.

Klikom na opciju Run app ili Shift + F10, u Android Studio IDE razvojnom okruženju, program je preveden i pokrenut emulatorom. Pokreće se aktivnost koja je modifikovana tako da korisniku omogućava da unese *ISBN* broj i naslov knjige pomoću prethodno kreiranog provajdera sadržaja.

Slika 3.2 Početna strana aplikacije

Da bi provajder sadržaja dodao novu knjigu, neophodno je kreirati, u metodi onClickAddTitle(), objekat tipa ContentValues u koji se pakuju informacije koje se odnose na konkretnu knjigu (videti priloženi programski kod).

Takođe, kada se pogleda priloženi kod klase aktivnosti, za navedenu metodu moguće je uočiti da se jedan deo koda pojavljuje obeležen oznakom za komentare, a ispod njega je aktivan kod. Oba koda imaju zadatak da ukažu na polja *ISBN* i *naslovi* sa razlikom da prvi blok koda koristi konstane *BooksProvider.ISBN* i *BooksProvider.TITLE,* respektivno, za pristupanje poljima, ukoliko je provajder sadržaja kreiran u istom paketu kao aplikacija, dok drugi blok koda omogućava pristupanje provajderu sadržaja, koji ne mora da se nalazi u paketu aplikacije, direktnim navođenjem naziva polja i kompletnog URI identifikatora sadržaja. Razlike su prikazane sledećom slikom.

Slika 3.3 Provajderi i paketi aplikacija

FUNKCIONISANJE APLIKACIJE ZA ANGAŽOVANJE PROVAJDERA SADRŽAJA - NASTAVAK

Metode za ažuriranje i brisanje oslanjaju se na navođenje URI identifikatora sadržaja kojim se ukazuje na identifikator konkretne knjige.

Angažovanjem metode onClickRetrieveTitles(), omogućeno je učitavanje knjiga koja su prethodno definisane provajderom sadržaja. Takođe, u metodu je ugrađen upit koji kao rezultat vraća niz knjiga sortiran po polju title u opadajućem poretku. Kod ove metode je priložen sledećim listingom izdvojenim iz glavne klase aktivnosti.

```
public void onClickRetrieveTitles(View view) {
        //---vraća naslov---
        Uri allTitles =
Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books");
        Cursor c;
        CursorLoader cursorLoader = new CursorLoader(this,allTitles, null, null,
                null,"naslovi desc");
        c = cursorLoader.loadInBackground();
        if (c.moveToFirst()) {
            do{
                Toast.makeText(this,
                        c.getString(c.getColumnIndex(
                                BooksProvider._ID)) + ", " +
                                c.getString(c.getColumnIndex(
```

```
                                        BooksProvider.TITLE)) + ", " +
                                c.getString(c.getColumnIndex(
                                        BooksProvider.ISBN)),
                        Toast.LENGTH_SHORT).show();
            } while (c.moveToNext());
        }
    }
```

Preuzete kniige prikazuju se na ekranu uređaja (sledeća slika).

Takođe, aplikacija omogućava da se ažururiraju detalji, koji se odnose na konkretnu knjigu, primenom metode updateTitle()klase aktivnosti aplikacije. Navođenjem URI identifikatora sadržaja ukazuje se na identifikator konkretne knjige (videti sledeću sliku – obeleženo crvenom bojom).

Takođe, primenom URI identifikatora sadržaja, metodom deleteTitle(), omogućeno je brisanje pojedinačnih ili svih knjiga (videti sledeću sliku – obeleženo crnom bojom).

Slika 3.4 Brisanje, ažuriranje i prikaz knjiga

PRIMER 4 - KREIRANJE KORISNIČKOG PROVAJDERA SADRŽAJA

Vežbanje kreiranja i angažovanja vlastitog provajdera sadržaja u Android aplikaciji.

Zadatak:

- Kreirati klasu provajdera sadržaja za pristup bazi podataka;
- Kreirati osnovnu klasu aktivnosti za rukovanje provajderom sadržaja;
- Registrovati provajder sadržaja u AndroidManifest.xml datoteci;
- Omogućiti da provajder sadržaja izvede pad CRUD operacija nad bazom podataka.

GUI datoteka je data sledećim XML kodom:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="ISBN" />
    <EditText
        android:id="@+id/txtISBN"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />
    <TextView
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Naslov" />
    <EditText
        android:id="@+id/txtTitle"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />
    <Button
        android:text="Dodaj naslov"
        android:id="@+id/btnAdd"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onClickAddTitle" />
    <Button
        android:text="Preuzmi naslove"
        android:id="@+id/btnRetrieve"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onClickRetrieveTitles"  />
</LinearLayout>
```

Klasa provajdera sadržaja je data sledećim kodom:

```
package com.metropolitan.vlastitiprovajderdemo;


import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;
import android.util.Log;

/**
 * Created by Vladimir Milicevic on 9.11.2016..
 */

public class BooksProvider extends ContentProvider
{
    static final String PROVIDER_NAME =
            "com.metropolitan.vlastitiprovajderdemo.Books";

    static final Uri CONTENT_URI =
            Uri.parse("content://"+ PROVIDER_NAME + "/books");

    static final String _ID = "_id";
    static final String TITLE = "naslovi";
```

```
    static final String ISBN = "isbn";

    static final int BOOKS = 1;
    static final int BOOK_ID = 2;

    private static final UriMatcher uriMatcher;
    static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "books", BOOKS);
        uriMatcher.addURI(PROVIDER_NAME, "books/#", BOOK_ID);
    }

    //---kreiranje baze podataka---
    SQLiteDatabase booksDB;
    static final String DATABASE_NAME = "Books";
    static final String DATABASE_TABLE = "knjige";
    static final int DATABASE_VERSION = 1;
    static final String DATABASE_CREATE =
            "create table " + DATABASE_TABLE +
                    " (_id integer primary key autoincrement, "
                    + "naslovi text not null, isbn text not null);";

    private static class DatabaseHelper extends SQLiteOpenHelper
    {
        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db)
        {
            db.execSQL(DATABASE_CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion,
                              int newVersion) {
            Log.w("Baza podataka provajera sadržaja",
                    "Ažuriranje verzije sa " +
                            oldVersion + " na verziju " + newVersion +
                            ", stari podaci biće uništeni");
            db.execSQL("DROP TABLE IF EXISTS titles");
            onCreate(db);
        }
    }

    @Override
    public int delete(Uri arg0, String arg1, String[] arg2) {
        // arg0 = uri
        // arg1 = selection
        // arg2 = selectionArgs
        int count=0;
        switch (uriMatcher.match(arg0)){
```

```
            case BOOKS:
                count = booksDB.delete(
                        DATABASE_TABLE,
                        arg1,
                        arg2);
                break;
            case BOOK_ID:
                String id = arg0.getPathSegments().get(1);
                count = booksDB.delete(
                        DATABASE_TABLE,
                        _ID + " = " + id +
                                (!TextUtils.isEmpty(arg1) ? " AND (" +
                                        arg1 + ')' : ""),
                        arg2);
                break;
            default: throw new IllegalArgumentException("Unknown URI " + arg0);
        }
        getContext().getContentResolver().notifyChange(arg0, null);
        return count;
    }

    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)){
            //---preuzimanje svih knjiga---
            case BOOKS:
                return "vnd.android.cursor.dir/vnd.metropolitan.books ";

            //---preuzimanje jedne knjige---
            case BOOK_ID:
                return "vnd.android.cursor.item/vnd.metropolitan.books ";

            default:
                throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        //---dodaje novu knjigu---
        long rowID = booksDB.insert(
                DATABASE_TABLE,
                "",
                values);

        //---ako je dodavanje uspešno---
        if (rowID>0)
        {
            Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
            getContext().getContentResolver().notifyChange(_uri, null);
            return _uri;
        }
        throw new SQLException("Dodavanje u vrstu nije uspelo " + uri);
```

```
    }

    @Override
    public boolean onCreate() {
        Context context = getContext();
        DatabaseHelper dbHelper = new DatabaseHelper(context);
        booksDB = dbHelper.getWritableDatabase();
        return (booksDB == null)? false:true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
        sqlBuilder.setTables(DATABASE_TABLE);

        if (uriMatcher.match(uri) == BOOK_ID)
            //---ako se pruzima konkretna knjiga---
            sqlBuilder.appendWhere(
                    _ID + " = " + uri.getPathSegments().get(1));

        if (sortOrder==null || sortOrder=="")
            sortOrder = TITLE;

        Cursor c = sqlBuilder.query(
                booksDB,
                projection,
                selection,
                selectionArgs,
                null,
                null,
                sortOrder);

        //---registrovanje praćenja promena URI identifikatora---
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)){
            case BOOKS:
                count = booksDB.update(
                        DATABASE_TABLE,
                        values,
                        selection,
                        selectionArgs);
                break;
            case BOOK_ID:
                count = booksDB.update(
                        DATABASE_TABLE,
```

```
                        values,
                        _ID + " = " + uri.getPathSegments().get(1) +
                                (!TextUtils.isEmpty(selection) ? " AND (" +
                                        selection + ')' : ""),
                        selectionArgs);
                break;
            default: throw new IllegalArgumentException("Nepoznat URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
}
```

Glavna klasa aktivnosti je data priloženim kodom:

```
package com.metropolitan.vlastitiprovajderdemo;

import android.support.v7.app.AppCompatActivity;
import android.content.ContentValues;
import android.content.CursorLoader;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
//import android.provider.ContactsContract;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    /** Poziva se kada se aktivnost kreira. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClickAddTitle(View view) {
        /*
        //---dodaj knjigu---
        ContentValues values = new ContentValues();
        values.put(BooksProvider.TITLE, ((EditText)
                findViewById(R.id.txtTitle)).getText().toString());
        values.put(BooksProvider.ISBN, ((EditText)
                findViewById(R.id.txtISBN)).getText().toString());
        Uri uri = getContentResolver().insert(
                BooksProvider.CONTENT_URI, values);
         */
        ContentValues values = new ContentValues();
        values.put("naslovi", ((EditText)
                findViewById(R.id.txtTitle)).getText().toString());
        values.put("isbn", ((EditText)
                findViewById(R.id.txtISBN)).getText().toString());
```

```
        Uri uri = getContentResolver().insert(

Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books"),
                values);

        Toast.makeText(getBaseContext(),uri.toString(),
                Toast.LENGTH_LONG).show();
    }

    public void onClickRetrieveTitles(View view) {
        //---vraća naslov---
        Uri allTitles =
Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books");
        Cursor c;
        CursorLoader cursorLoader = new CursorLoader(this,allTitles, null, null,
                null,"naslovi desc");
        c = cursorLoader.loadInBackground();
        if (c.moveToFirst()) {
            do{
                Toast.makeText(this,
                        c.getString(c.getColumnIndex(
                                BooksProvider._ID)) + ", " +
                                c.getString(c.getColumnIndex(
                                        BooksProvider.TITLE)) + ", " +
                                c.getString(c.getColumnIndex(
                                        BooksProvider.ISBN)),
                        Toast.LENGTH_SHORT).show();
            } while (c.moveToNext());
        }
    }

    public void updateTitle() {
        ContentValues editedValues = new ContentValues();
        editedValues.put(BooksProvider.TITLE, "Android Tipovi i trikovi.");
        getContentResolver().update(

Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books/
2"),
                editedValues,null,null);
    }

    public void deleteTitle() {

        //---brisanje naslova---
        getContentResolver().delete(

Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books/
2"),
                null, null);
        //---brisanje svih naslova---
        getContentResolver().delete(

Uri.parse("content://com.metropolitan.vlastitiprovajderdemo.provider.Books/books"),
```

```
                null, null);
    }

}
```

ZADATAK 3 - PAKOVANJE OMILJENIH KONTAKATA U LISTU

Pokušajte sami!!!

Pokušajte da kreirate aplikaciju koja u listu pakuje vaše omiljene kontakte odakle ih po potrebi poziva.

Poglavlje 4

Domaći zadatak 6

ZADATAK 1

Vežbanje kreiranja i korišćenja provajdera sadržaja.

Zadatak 1:

Kreirati android aplikaciju po sledećim zahtevima:

1. Koristiti Android Studio IDE okruženje i API verziju za Android 6.0;
2. Kreirati provajder sadržaja za komunikaciju sa bazom podataka *Konsultacije*;
3. Baza podataka *Konsultacije* sadrži jednu tabelu *Predmeti (*int sifra_predmeta primery key autoincrement, Text naziv_predmeta, Text rad_sa_studentima);
4. Kreirati metode koje omogućavaju CRUD operacije nad bazom primenom provajdera sadržaja;
5. Korisnički interfejs mora da obezbedi korisnicima kontrole za unos, ažuriranje, prikazivanje i brisanje predmeta iz tabele;
6. Realizovati angažovanje provajdera kreiranjem glavne klase aktivnosti aplikacije.

Zadatak 2:

U zadatak priložen u temi Kreiranje provajdera sadržaja ugraditi GUI kontrole za brisanje i ažuriranje sadržaja baze podataka.

Pregled Lekcije07

PREGLED LEKCIJE 06

Lekcijom07 je detaljno opisan koncept provajdera sadržaja u Android aplikacijama.

U lekciji je detaljno opisan koncept provajdera sadržaja u Android aplikacijama sa posebnim akcentom na načine korišćenja. Lekcijom su provajderi sadržaja podeljeni u dve grupe, koje su posebno i razmatrane. Prvu grupu čine provajderi sadržaja koji su ugrađeni u Android operativni sistem. Upravo jedan od njih, *Contacts* provajder sadržaja, bio je predmet bavljenja prvog dela lekcije. U nastavku, lekcija se fokusira na drugu grupu provajdera sadržaja, koji su kreirani za konkretne potrebe korisnika. Kreiranje i primena ovakvog provajdera sadržaja posebno je prikazana ovom lekcijom.

LITERATURA

Za pripremanje lekcije korišćena je sledeća literatura

1. Lee W. M. 2012. *Android 4 – razvoj aplikacija*, Wiley Publishing, INC
2. http://developer.android.com/training/index.html
3. http://www.tutorialspoint.com/android/
4. http://www.vogella.com/tutorials/android.html
5. http://www.javahelps.com/2015/10/android-60-runtime-permission-model.html