



MOBILNE APLIKACIJE

VEŽBE 6

Dobavljači sadržaja

2022/2023

Sadržaj

1. Deljena podešavanja.....	3
2. SQLite	8
3. Prava pristupa	11
3.1 Statička prava pristupa	11
3.2 Dinamička prava pristupa	11
4. Dobavljači sadržaja	12
4.1 Jedinstvena identifikacija resursa	12
4.2 Aplikacioni dobavljači sadržaja	12
5. Punjači.....	16
6. Domaći	17

1. Deljena podešavanja

Deljena podešavanja (*SharedPreferences*) olakšavaju perzistentno skladištenje prostih tipova podataka. Podatke čuvamo u datoteci kao uređene parove (ključ, vrednost).

Kreirali smo klasu *ReviewerPreferenceActivity* i u nju smo smestili fragment *PrefsFragment* (slika 1). Ovaj fragment će predstavljati naša podešavanja. Kreirali smo fragment jer se preporučuje da ekran sa podešavanjima bude baš fragment.

```
public static class PrefsFragment extends PreferenceFragmentCompat {  
  
    private static PrefsFragment newInstance() {  
        Bundle args = new Bundle();  
  
        PrefsFragment fragment = new PrefsFragment();  
        fragment.setArguments(args);  
        return fragment;  
    }  
  
    @Override  
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {  
        addPreferencesFromResource(R.xml.preferences);  
    }  
}
```

Slika 1. *PrefsFragment*

Ovaj fragment neće imati svoj layout, već ćemo koristiti posebnu specifikaciju, koja opisuje koja podešavanja imamo (slika 2). U elementu *<PreferenceScreen>* se definišu samo elementi koji su sastavni deo podešavanja (nema *layout-a*).

PreferenceCategory

Ovaj element služi za grupisanje drugih elemenata. Grupe su u podešavanjima malo odvojene, razdvojene linijom. Atributi koje navodimo su: naslov i ključ grupe.

CheckBoxPreference

Ovaj element može da prikazuje dva stanja (polje je označeno ili nije).

Atributi koje navodimo su:

- *defaultValue* – kada pokrenemo aplikaciju ovo će biti podrazumevana vrednost, u našem primeru vrednost je *false*
- *key* - ključ
- *summary* – kratak opis
- *title* – naziv

ListPreference

Pre elementa *ListPreference* uglavnom sledi *checkBox* element za koji vezujemo listu. Bitni atributi koje navodimo za listu su:

- *dependency* – na ovaj atribut postavljamo ključ *checkBoxPreference* elementa za koji želimo da vežemo ovu listu. Ako korisnik ne čekira prethodno definisan *checkBoxPreference*, onda će lista biti onemogućena za manipulaciju. Primer se vidi na slici 4. Na slici levo *checkBox* nije označen, a desno jeste.
- *summary* - navodimo kog tipa će biti elementi te liste koje prikazujemo (%s-string).
- *entries* - govori šta će korisniku biti prikazano kad se otvori lista.
- *entryValues* - su vrednosti koje će biti upisane u *sharedPreferences* za odgovarajući element.

Sledi da liste *entries* i *entryValues* moraju biti iste dužine (za svaki *entry* koji se prikazuje, postoji odgovarajući *entryValue*, koji se postavlja kao vrednost). Za prvu listu smo postavili listu *pref_syncConnectionTypes_entries*, a za drugu *pref_syncConnectionTypes_values*. Te dve liste smo definisali u datoteci *arrays.xml* unutar direktorijuma *res/values/* (slika 3). Na slici 5 se vidi kako to izgleda na primeru, kada korisnik klikne na *Sync interval*.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
3
4     <PreferenceCategory
5         android:title="Auto data sync"
6         android:key="sync_settings">
7
8         <CheckBoxPreference
9             android:defaultValue="false"
10            android:key="pref_sync"
11            android:summary="Allow app to sync data automatically, when wifi is aval..."
12            android:title="Allow Sync" />
13
14        <ListPreference
15            android:dependency="pref_sync"
16            android:dialogTitle="Sync interval"
17            android:entries="@array/pref_syncConnectionTypes_entries"
18            android:entryValues="@array/pref_syncConnectionTypes_values"
19            android:key="pref_sync_list"
20            android:title="Sync interval"
21            android:summary="%s"
22            android:defaultValue="1"/>
23
24    </PreferenceCategory>
25
26 </PreferenceScreen>
```

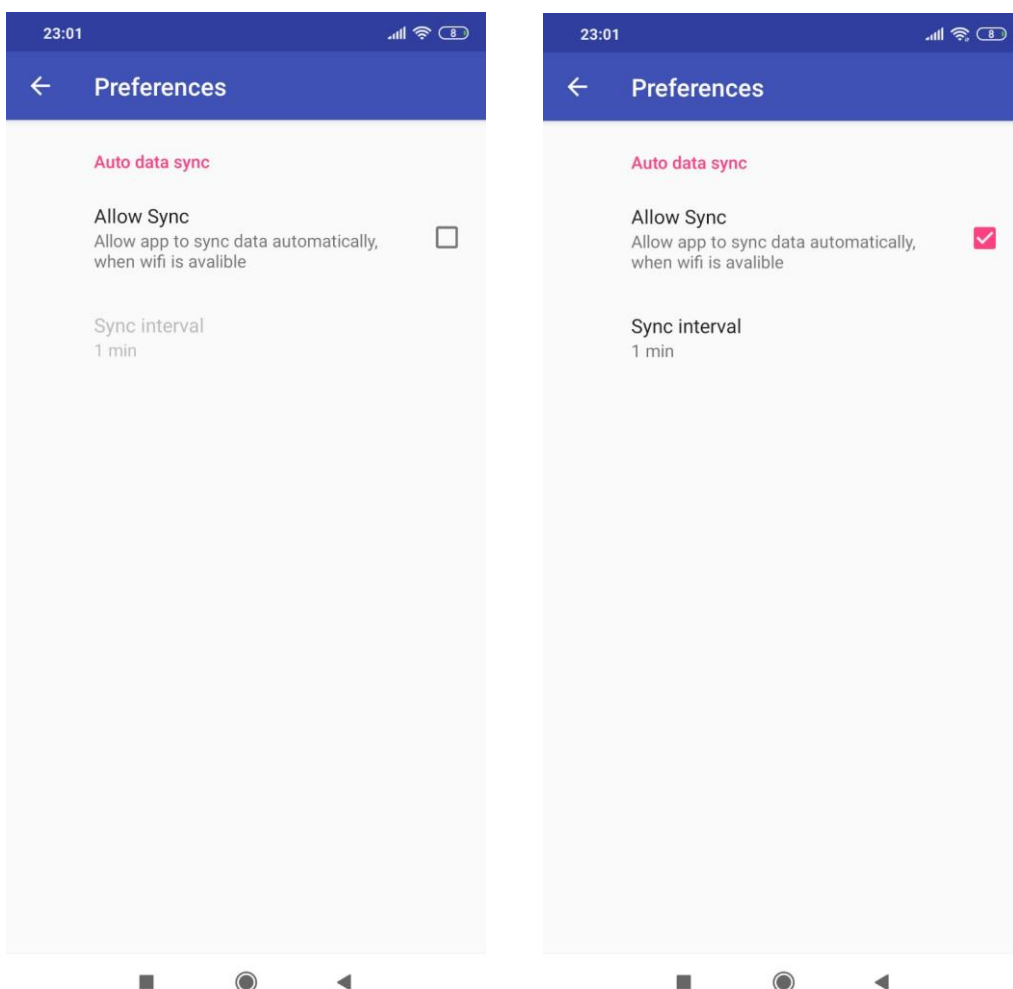
Slika 2. *preferences.xml*

```

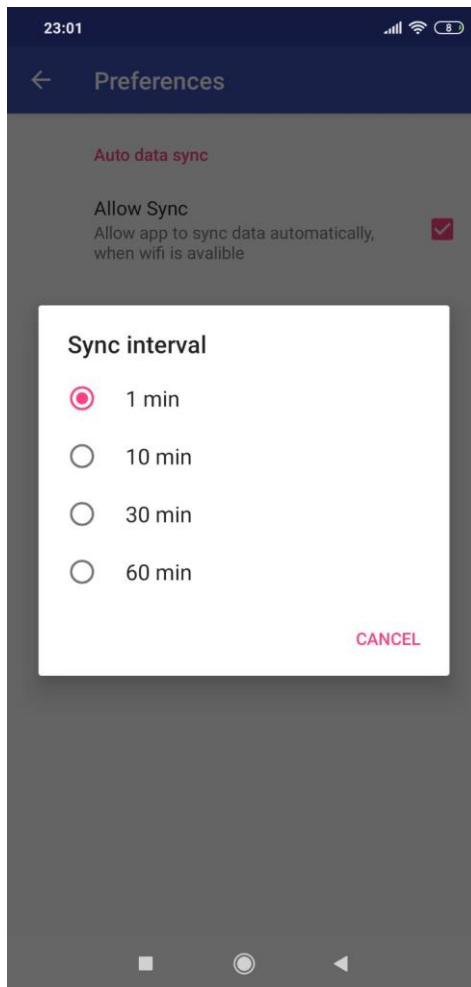
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3
4   <string-array name="pref_syncConnectionTypes_entries">
5     <item>1 min</item>
6     <item>10 min</item>
7     <item>30 min</item>
8     <item>60 min</item>
9   </string-array>
10
11   <string-array name="pref_syncConnectionTypes_values">
12     <item>1</item>
13     <item>10</item>
14     <item>30</item>
15     <item>60</item>
16   </string-array>

```

Slika 3. Primeri nizova koji se prikazuju



Slika 4. Prikaz podešavanja



Slika 5. Prikaz liste sa opcijama koja se prikazuje kada se klikne na *Sync interval*

U *MainActivity* klasi se nalazi metoda *consultPreferences()*. U ovoj metodi pristupamo deljenim podešavanjima tako što pozivamo metodu *getSharedPreferences* ugrađene klase *Context* (slika 6).

Metoda *contains* proverava da li postoji podesavanje pod prosleđenim naslovom u okviru Shared Preferences. Metoda *getString* vraća string iz podešavanja, a metoda *getBoolean* vraća boolean vrednost. Gore smo napomenuli da u deljena podešavanja smeštamo parove (ključ, vrednost), te ćemo ključeve koristiti da dobavimo određene vrednosti. Kada pozivamo metode za dobavljanje, prosleđujemo 2 parametra:

- ključ - prvi parametar je ključ, čiju vrednost želimo da dobavimo.
- podrazumevana vrednost - drugi parametar je vrednost, koja treba da nam se vrati nazad, u slučaju da ništa nije zapisano pod prosleđenim ključem.

```

134
135 private void consultPreferences(){
136
137     /*...*/
147     /*...*/
152     sharedPreferences = getSharedPreferences( name: "com.example.vozbe6_preferences", Context.MODE_PRIVATE);
153     /*...*/
154
155     if(sharedPreferences.contains("pref_sync_list")){
156         Log.i( tag: "REZ_SP", msg: "Postoji pref_sync_list kljuc");
157         /*...*/
166         synctime = sharedPreferences.getString( s: "pref_sync_list", s1: "1"); // pola minuta
167         Log.i( tag: "REZ_SP", msg: "SYNC TIME = " + synctime.toString());
168
169     }
170
171     if(sharedPreferences.contains("pref_sync")){
172         Log.i( tag: "REZ_SP", msg: "Postoji pref_sync kljuc");
173         allowSync = sharedPreferences.getBoolean( s: "pref_sync", b: false);
174         Log.i( tag: "REZ_SP", msg: "ALLOW SYNC = " + allowSync);
175     }
176
177     if(sharedPreferences.contains("pref_name")){
178         Log.i( tag: "REZ_SP", msg: "Postoji pref_name kljuc");
179         String name = sharedPreferences.getString( s: "pref_name", s1: "default");
180         Log.i( tag: "REZ_SP", msg: "PREF NAME = " + name);
181     }else{
182         Log.i( tag: "REZ_SP", msg: "Ne postoji pref_name kljuc");
183     }
184
185 }

```

Slika 6. Čitanje zapisanih vrednosti

Na sličan način možemo da pristupimo deljenim podešavanjima i da smestimo neko podešavanje unutar njega. Pomoću metode *putString*, *putBoolean* itd. možemo da dodamo podešavanje sa nekim ključem i vrednošću (slika 7).

```

186
187 private void setPreferences(){
188     sharedPreferences = getSharedPreferences( name: "com.example.vozbe6_preferences", Context.MODE_PRIVATE);
189     SharedPreferences.Editor sp_editor = sharedPreferences.edit();
190     Log.i( tag: "REZ_SP", msg: "Set pref_name kljuc");
191     if(!sharedPreferences.contains("pref_name")){
192         sp_editor.putString( s: "pref_name", s1: "iReviewer");
193         sp_editor.commit();
194     }
195
196 }
197

```

Slika 7. Zapisivanje vrednosti u deljenim podešavanjima

2. SQLite

Android aplikacije mogu da koriste ugradjen sistem za upravljanje bazama podataka (*SQLite*). *SQLite* se izvršava u istom procesu kao i sama aplikacija.

Baza podataka predstavljena je klasom *SQLiteDatabase*.

CRUD operacije nad bazom podataka izvršavaju se pozivom *insert*, *query*, *update* i *delete* metoda. *query* metoda je specifična jer kao rezultat vraća *Cursor*. *Cursor* vraća adresu gde se podaci nalaze tj. relacija koja je rezultat SQL upita predstavljena je kursorom. Kursore koristimo da bismo vršili navigaciju kroz rezultat upita.

U bazi svaka tabela ima redove i kolone. Potrebno je pozicionirati se na odgovarajući red i odgovarajuću kolonu. Prvo se pozicioniramo na red sa nekom od metoda:

- `boolean move(int offset)`
Pomeri se za određeni broj redova od trenutne pozicije.
- `boolean moveToFirst()`
Pređi na prvi red.
- `boolean moveToLast()`
Pređi na poslednji red
- `boolean moveToNext()`
Pređi na sledeći red.
- `boolean moveToPrevious()`
Pređi na prethodni red.

Kada se pozicioniramo na red, sledeći korak je da odaberemo kolonu, tj. da pročitamo rezultat upita. Za to možemo da iskoristimo neku od metoda:

- `int getCount()`
- `int getColumnIndex(String column_name)`
- `String getColumnName(int column_index)`
- `String getString(int column_index)`
- `int getInt(int column_index)`
- `long getLong(int column_index)`
- `float getFloat(int column_index)`

U primeru za ove vežbe kreirali smo klasu *ReviewerSQLiteHelper* koja nasleđuje klasu *SQLiteOpenHelper*. Klasa *SQLiteOpenHelper* nam omogućava da napravimo, izmenimo ili otvorimo bazu podataka.

Implementiramo metode:

- `void onCreate(SQLiteDatabase database)`

- `void onOpen(SQLiteDatabase database)`
- `void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion)`
- `void onDowngrade(SQLiteDatabase database, int oldVersion, int newVersion)`

onCreate

U ovoj metodi kreiramo našu bazu tako što pozivamo `db.execSQL` za kreiranu tabelu (slika 8).

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

Slika 8. Kreiranje tabele

onUpgrade

Metodu `onUpgrade` pozivamo kad baza treba da se promeni (slika 9). Prvi korak je da dropujemo sve tabele koje imamo.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_CINEMA);
    onCreate(db);
}
```

Slika 9. Izmena tabele

U klasi *Util* kreirali smo metodu *initDB* koja služi za inicijalizaciju naše baze (slika 10). U toj metodi smo kreirali dva entiteta i uneli ih u tabelu pozivanjem metode *insert*.

```

public class Util {
    public static void initDB(Activity activity) {
        ReviewerSQLiteHelper dbHelper = new ReviewerSQLiteHelper(activity);
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        {
            ContentValues entry = new ContentValues();
            entry.put(ReviewerSQLiteHelper.COLUMN_NAME, "Arena");
            entry.put(ReviewerSQLiteHelper.COLUMN_DESCRIPTION, "Cineplexx 3D");
            entry.put(ReviewerSQLiteHelper.COLUMN_AVATAR, -1);

            activity.getContentResolver().insert(DBContentProvider.CONTENT_URI_CINEMA, entry);

            entry = new ContentValues();
            entry.put(ReviewerSQLiteHelper.COLUMN_NAME, "Cinestar");
            entry.put(ReviewerSQLiteHelper.COLUMN_DESCRIPTION, "Najnoviji 5D");
            entry.put(ReviewerSQLiteHelper.COLUMN_AVATAR, -1);

            activity.getContentResolver().insert(DBContentProvider.CONTENT_URI_CINEMA, entry);
        }
        db.close();
    }
}

```

Slika 10. Metoda za inicijalizacije *SQLite* baze

Na slici 11 se vidi da se metoda *initDB*, klase *Util*, poziva u trenutku kada korisnik klikne na stavku menija *new* (drugi case u switch-u). Kada pokrenete aplikaciju na uređaju videćete da se, nakon što kliknete na dugme *new*, pojave dva filma, koja smo kreirali u metodi *initDB*.

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            Intent i = new Intent( packageContext: this, ReviewerPreferenceActivity.class);
            startActivity(i);
            return true;
        case R.id.action_new:
            Util.initDB( activity: MainActivity.this);
            finish();
            startActivity(getIntent());
    }

    return super.onOptionsItemSelected(item);
}

```

Slika 11. Pozivanje metode *initDB*

3. Prava pristupa

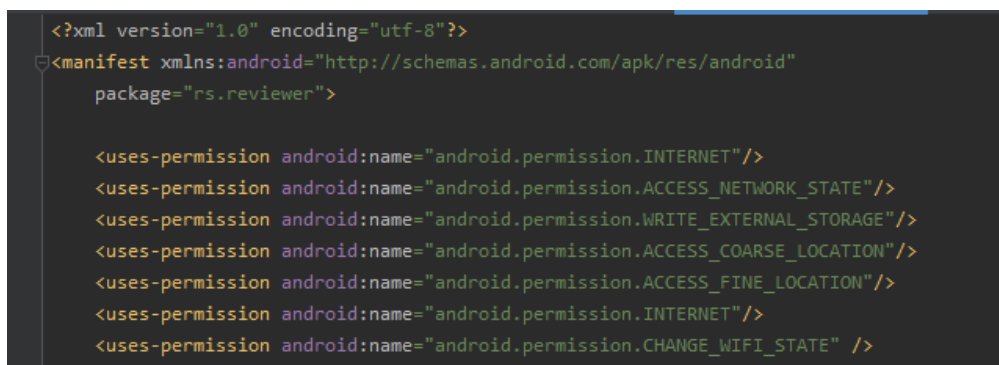
Aplikacija ne može da izvršava operacije koje mogu negativno da utiču na druge aplikacije, OS ili korisnike, ukoliko joj to nije dozvoljeno. Postoje 2 vrste prava pristupa:

1. Statička prava pristupa (do Androida 5.1)
2. Dinamička prava pristupa (od Androida 6)

3.1 Statička prava pristupa

Do Androida 5.1 sva prava pristupa koja su bila potrebna za uspešno izvršavanje aplikacije, statički se navode u *AndroidManifest* datoteci (slika 12). Prilikom instalacije aplikacije korisnik odobrava prava pristupa, koja aplikacija traži od njega ili odustaje od instalacije.

U *AndroidManifest* datoteci navodi se element `<user-permission>` za svako pravo pristupa koje nam je potrebno.

A screenshot of an XML file, specifically an AndroidManifest.xml, with a dark background and light-colored text. The code defines a package named 'rs.viewer' and lists several permissions using the <uses-permission> tag. The permissions listed are: android.permission.INTERNET, android.permission.ACCESS_NETWORK_STATE, android.permission.WRITE_EXTERNAL_STORAGE, android.permission.ACCESS_COARSE_LOCATION, android.permission.ACCESS_FINE_LOCATION, android.permission.INTERNET (repeated), and android.permission.CHANGE_WIFI_STATE.

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

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
```

Slika 12. Definisana prava pristupa u u *AndroidManifest* datoteci

3.2 Dinamička prava pristupa

Od Android verzije 6.0 uvode se dinamička prava pristupa. Aplikacija svaki put pre nego što izvrši neku operaciju, koja zahteva pravo pristupa, proverava da li ima odobreno to pravo. Android može automatski da odobri pravo pristupa aplikaciji ili može da zatraži od korisnika da joj odobri pravo pristupa.

Kada koristite npr. *Viber* i želite da uslikate kamerom i pošaljete sliku na čit, ako pre toga ovu akciju niste izvršavali, *Viber* će vas pitati da odobrite pravo pristupa vašoj kameri. Izaći će vam poruka „Allow Viber to take pictures and record video“. Ako prihvatite bićete u mogućnosti da uslikate kamerom i pošaljete sliku, u suprotnom nećete moći da pristupite kameri.

Korisnik može u svakom trenutku aplikaciji da oduzme određeno pravo pristupa.

Primer i objašnjenje koda za dinamička prava pristupa možete pronaći u PDF-u 08 Lociranje i mape.

Prilikom kreiranja vaših aplikacija treba da pokrijete i statička i dinamička prava pristupa, jer kreirate aplikacije i za starije verzije Android-a.

4. Dobavljači sadržaja

Dobavljači sadržaja (*ContentProvider*) upravljaju podacima aplikacije i omogućavaju da im se pristupi na standardizovan način. Podaci se mogu nalaziti u bazi podataka, na internetu ili nekom drugom mestu. Npr. postoji dobavljač sadržaja koji dozvoljava korisniku da uređuje listu kontakata na uređaju. Ovo znači da svaka aplikacija, ako ima odobrenje, može da pristupa određenim dobavljačima i da čita ili upisuje neke podatke.

Postoje dve vrste dobavljača sadržaja:

1. Sistemski
2. Aplikacioni

Sistemski dobavljači su uključeni u Android (*Browser, Calendar, CallLog, Contacts..*).

Aplikacione dobavljače sadržaja prave programeri, koji su zaduženi za kreiranje same aplikacije. Ako želimo da resursima naše aplikacije mogu da pristupe neke druge aplikacije, onda moramo to i da im omogućimo kreiranjem dobavljača sadržaja. Kako se prave aplikacioni dobavljači sadržaja predstavljeno je u potpoglavlju 4.2.

4.1 Jedinstvena identifikacija resursa

Resurse opisuje URI i MIME tip.

Primer URI-ja:

`content://user_dictionary/words`

URI se sastoji iz 3 dela:

- šeme,
- imena dobavljača i
- imena tabele.

MIME tip specifikira tip sadržaja (text/plain, text/pdf, image/jpeg, itd.). Kada se navede kao tip npr. *jpeg* to znači da ćemo dobiti samo slike sa tom ekstenzijom.

Primer: U galeriji telefona se čuvaju slike koje kamera napravi. Galeriji pristupamo preko dobavljača sadržaja i tražimo određenu sliku. Kao povratna vrednost stiže URI, a ne slika, jer takva situacija ne bi bila optimalna. Sa tim URI-jem možemo da tražimo direktan pristup resursu.

4.2 Aplikacioni dobavljači sadržaja

Kreiramo klasu *DBContentProvider* koja nasleđuje *ContentProvider*.

Deklarišemo klasu *DBContentProvider* u *AndroidManifest.xml* datoteci (slika 13) tako što navedemo element `<provider>`.

```
<provider
    android:name="rs.reviewer.database.DBContentProvider"
    android:authorities="rs.reviewer"
    android:exported="false" />
```

Slika 13. Deklarisanje dobavljača sadržaja u *AndroidManifest* datoteci

Na slici 14 se nalazi primer kreiranja URI-ja. *Authority* polje je ime dobavljača, što je najčešće paket. *Cinema_path* je naziv tabele.

Ako korisnik želi informacije o pojedinačnom resursu onda definišemo *UriMatcher*. Svaki put proveravamo da li je zahtev stigao za celu tabelu ili za pojedinačan red i u zavisnosti od toga se pretražuje baza i vraća se rezultat nazad.

MIME tip specificira tip sadržaja i kada želimo da dobijemo sve slike sa ekstenzijom jpeg, onda navodimo image/jpeg, a ako želimo sve slike nezavisno od ekstenzije, šaljemo image/*.

```
private static final int CINEMA = 10;
private static final int CINEMA_ID = 20;

private static final String AUTHORITY = "rs.reviewer";

private static final String CINEMA_PATH = "cinema";

public static final Uri CONTENT_URI_CINEMA = Uri.parse("content://" + AUTHORITY + "/" + CINEMA_PATH);

private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO_MATCH);

static {
    sURIMatcher.addURI(AUTHORITY, CINEMA_PATH, CINEMA);
    sURIMatcher.addURI(AUTHORITY, path: CINEMA_PATH + "/" + "#", CINEMA_ID);
}
```

Slika 14. Kreiranje URI-ja

Prilikom nasleđivanja *ContentProvider* klase, treba implementirati metode:

- *insert()*,
- *query()*,
- *update()*,
- *delete()*,
- *getType()* i
- *onCreate()*

Na slici 15 se nalazi primer metode *insert*.

```

@Nullable
@Override
public Uri insert(Uri uri, ContentValues values) {
    Uri retVal = null;
    int uriType = sURIMatcher.match(uri);
    SQLiteDatabase sqlDB = database.getWritableDatabase();
    long id = 0;
    switch (uriType) {
        case CINEMA:
            id = sqlDB.insert(ReviewerSQLiteHelper.TABLE_CINEMA, nullColumnHack: null, values);
            retVal = Uri.parse(CINEMA_PATH + "/" + id);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI: " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, observer: null);
    return retVal;
}

```

Slika 15. Metoda *insert*

Na slici 16 se nalazi primer *query* metode, koja kao rezultat vraća *Cursor*.

```

@Nullable
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    // Using SQLiteQueryBuilder instead of query() method
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();

    // check if the caller has requested a column which does not exist //checkColumns(projection);
    int uriType = sURIMatcher.match(uri);
    switch (uriType) {
        case CINEMA_ID:
            // Adding the ID to the original query
            queryBuilder.appendWhere( inWhere: ReviewerSQLiteHelper.COLUMN_ID + "="
                + uri.getLastPathSegment());
            //FALL-THROUGH$
        case CINEMA:
            // Set the table
            queryBuilder.setTables(ReviewerSQLiteHelper.TABLE_CINEMA);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI: " + uri);
    }

    SQLiteDatabase db = database.getWritableDatabase();
    Cursor cursor = queryBuilder.query(db, projection, selection,
        selectionArgs, groupBy: null, having: null, sortOrder);
    // make sure that potential listeners are getting notified
    cursor.setNotificationUri(getContext().getContentResolver(), uri);
    return cursor;
}

```

Slika 16. Metoda *query*

U primeru za ove vežbe možete pogledati i sve ostale metode.

Šta ako se pojave dve aplikacije koje u isto vreme pošalju zahtev za neki resurs naše aplikacije? Treba da vodimo računa o sinhronizaciji. Sve metode osim metode *onCreate()* treba da budu *thread-safe*. Treba izbegavati izvršavanje dugačkih operacija u metodi *onCreate()*.

Primećujemo da metoda *onDestroy* ne postoji, jer dobavljači sadržaja postoje od početka do kraja procesa.

Klasa *ContentResolver* omogućava izvršavanje CRUD (create, read, update, delete) operacija nad skladištem podataka. Podacima pristupamo tako što zatražimo odgovarajuća prava pristupa, a potom izvršavamo upit nad dobavljenim sadržajem.

Upit se postavlja na sličan način na koji se postavlja SQL upit. Sadrži URI, spisak kolona koje treba vratiti (projekciju), uslov koji vraćene vrste treba da zadovolje (selekciju) i način sortiranja rezultata.

Na sličan način na koji je moguće pristupiti podacima, moguće ih je i promeniti.

Primer upotrebe *ContentResolver*-a možete videti na slici 10 kada se vrši inicijalizacija SQLite baze. Metodi *initDB* prosleđujemo aktivnost, u našem primeru se prosleđuje *MainActivity*, i pozivamo metodu *getContentResolver*, a potom i metodu *insert* da ubacimo kreirani film.

5. Punjači

Punjači (*Loader*) omogućavaju asinhrono učitavanje podataka u aktivnosti i fragmente. Oni nadgledaju izvore podataka i isporučuju sadržaj kada se podaci promene. Takođe, oni vode računa o promeni stanja aktivnosti ili fragmenta.

SimpleCursorAdapter povezuje kursor sa *ListView* ili *GridView* pogledom. Na slici 17 se nalazi primer upotrebe *SimpleCursorAdapter*-a.

```
4      @Override
5      public void onCreate(Bundle savedInstanceState) {
6          super.onCreate(savedInstanceState);
7          Toast.makeText(getActivity(), "My Fragment - onCreate()", Toast.LENGTH_SHORT).show();
8          /*...*/
9          LoaderManager.getInstance(this).initLoader(0, null, callback: this);
10
11          String[] from = new String[] { ReviewerSQLiteHelper.COLUMN_NAME, ReviewerSQLiteHelper.COLUMN_DESCRIPTION };
12          int[] to = new int[] { R.id.name, R.id.description };
13          adapter = new SimpleCursorAdapter(getActivity(), R.layout.cinema_list, c: null, from, to, flags: 0);
14          setListAdapter(adapter);
15      }
```

Slika 17. Primer upotrebe *SimpleCursorAdapter*-a

6. Domaći

Domaći se nalazi na *Canvas-u* (*canvas.ftn.uns.ac.rs*) na putanji *Вежбе/06 Вежбе/06 Задатак.pdf*.

Primer *Vezbe6* možete preuzeti na sledećem linku: <https://gitlab.com/antesevicceca/mobilne-aplikacije>

Za dodatna pitanja možete se obratiti asistentima:

- Svetlana Antešević (svetlanaantesevic@uns.ac.rs)
- Jelena Matković (matkovic.jelena@uns.ac.rs)