Przechowywanie danych

Rafał Zientara

Cel bloku tematycznego

- Poznanie i zrozumienie różnych sposobów przechowywania danych w Androidzie
- Umiejętność udostępniania danych innym aplikacjom

Sposoby przechowywania danych w Androidzie

1. Shared Preferences
2. Internal Storage
3. External Storage
4. SQLite Databases
5. Web Services (Firebase)

Shared Preferences

- Dane przechowywanie w pliku (prywatny katalog aplikacji) dopóki apka nie zostanie usunięta.
- Możliwość zapisu tylko typów prostych (String, Int, Long, Double, Boolean …)
- Struktura klucz -> wartość
- Zazwyczaj wykorzystywane do przechowywania ustawień aplikacji
- Przed funkcją commit()/apply() możemy wprowadzać wiele danych

```
SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(this);

preferences
        .edit()
        .putString("KLUCZ_ZMIENNEJ", "wpisuje wartosc zmiennej")
        .commit();

String myVariable = preferences.getString("KLUCZ_ZMIENNEJ", "wartosc domyslna");
```

Preferencje dzielone

- Preferencje można rozdzielić na parę różnych plików xml i używać je w ten sam sposób przez obiekt SharedPreferences

```
SharedPreferences preferences1 = getSharedPreferences("preferencje1", MODE_PRIVATE);
SharedPreferences preferences2 = getSharedPreferences("preferencje2", MODE_PRIVATE);
```

Zadanie

Utwórz aplikację która:

- Posiada 3 przyciski z różnymi kolorami
- Po kliknięciu przycisku zmieni się kolor tła
- Po schowaniu aplikacji do tła lub jej zamknięciu ustawiony kolor tła pozostanie taki sam jak po kliknięciu przycisku

Operacje na plikach (External/Internal Storage)

- Do operacjach na plikach często używa się klas dostępnych w pakiecie java.io takie jak File
- listFiles() – zwraca tablicę plików/folderów znajdujących się w folderze
- lastModified() – data ostatniej aktualizacji pliku

- FileInputStream – klasa do wprowadzania danych do pliku
- FileOutputStream – klasa do wczytywania danych z pliku

Ręczne zapisywanie w pliku (Internal Storage)

- Zapisywanie w katalogu pamięci wbudowanej (Internal Storage)

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Zapisywanie danych tymczasowych do Cache

- używamy metody *getCacheDir()* (pliki z tego katalogu mogą być usunięte przez system gdy urządzeniu będzie brakowało pamięci)
- powinniśmy czyścić zbędne dane z katalogu cache nie licząc na interwencję systemu

Ręczne zapisywanie w pliku (External Storage)

- Wymagane uprawnienie

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

- Od Androida 6.0 wymagane zapytanie o uprawnienia do czytania/zapisu danych w niektórych folderach

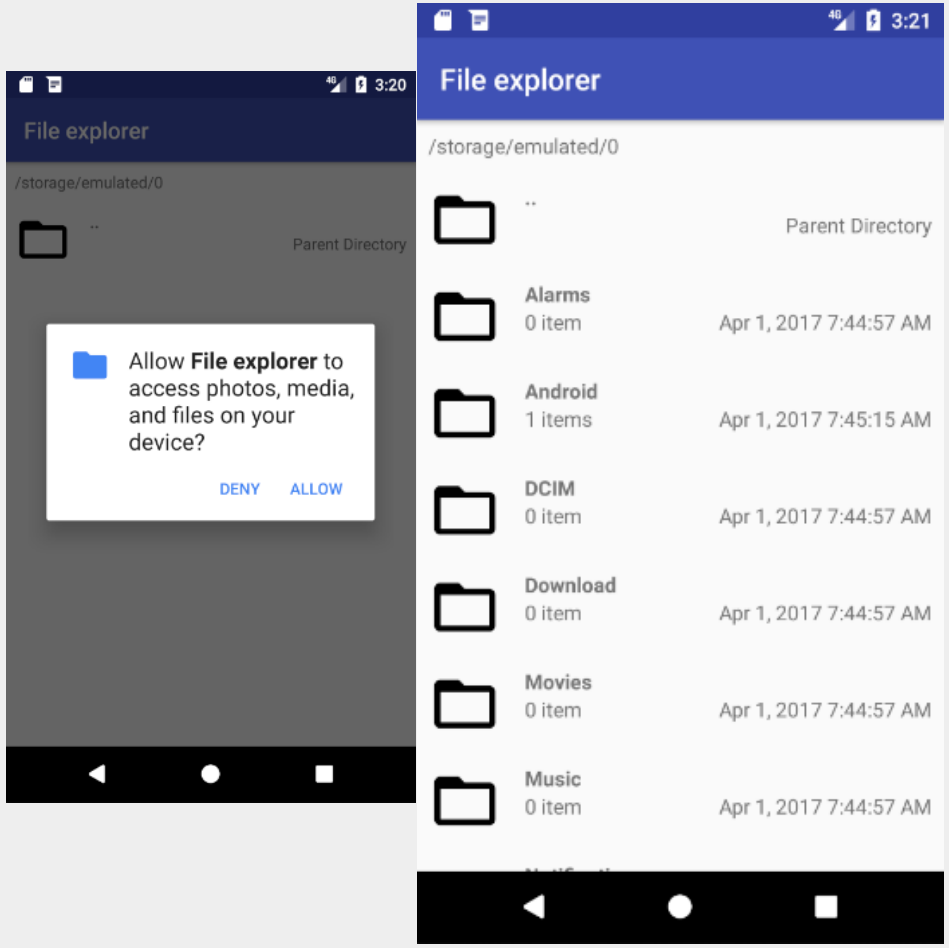
```
ActivityCompat.requestPermissions(this,
        new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, REQUEST_CODE);
```

- Powinniśmy użyć metody *getExternalStorageState()* żeby sprawdzić czy pamięć zewnętrzna (karta pamięci) jest dostępna i czy możemy dokonywać na niej zapisu

- Ścieżkę do pamięci zewnętrznej można pobrać przez:

```
Environment.getExternalStorageDirectory().getPath()
```

Zadanie



File explorer

- Pod belką aplikacji wyświetlona aktualna ścieżka
- Widoczna lista plików (ListView/RecyclerView)
- Każdy z elementów będzie miał widoczną nazwę folderu/pliku, datę modyfikacji, ikonę folderu/pliku
- Aplikacja będzie potrzebowała zapytania o uprawnienia do niektórych folderów

Korzystanie z bazy danych Sqlite

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

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

Korzystanie z bazy danych Sqlite

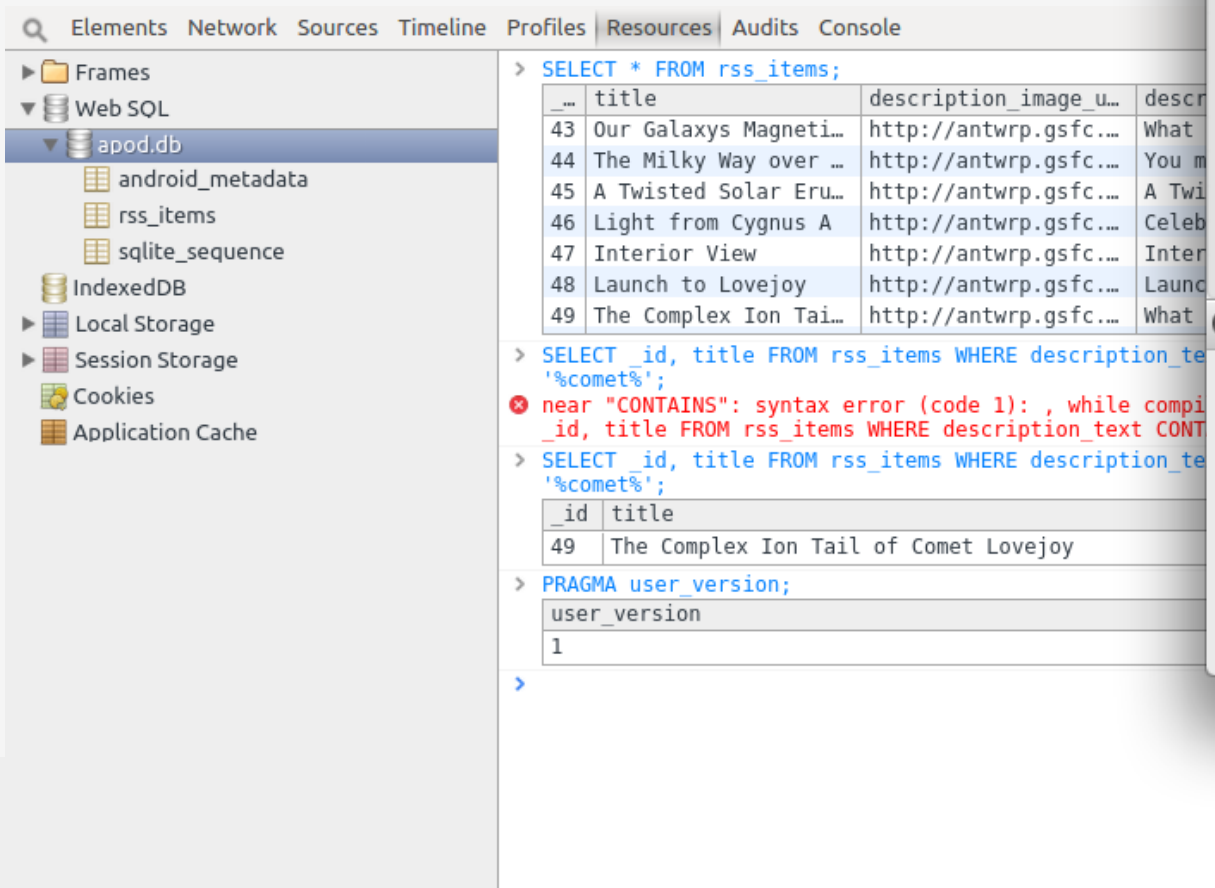
- Tworzymy instancję **SqliteOpenHelper** i korzystając z metod *getReadableDatabase(), getWriteableDatabase()* uzyskujemy obiekt, który umożliwia nam wykonywanie zapytań Sql

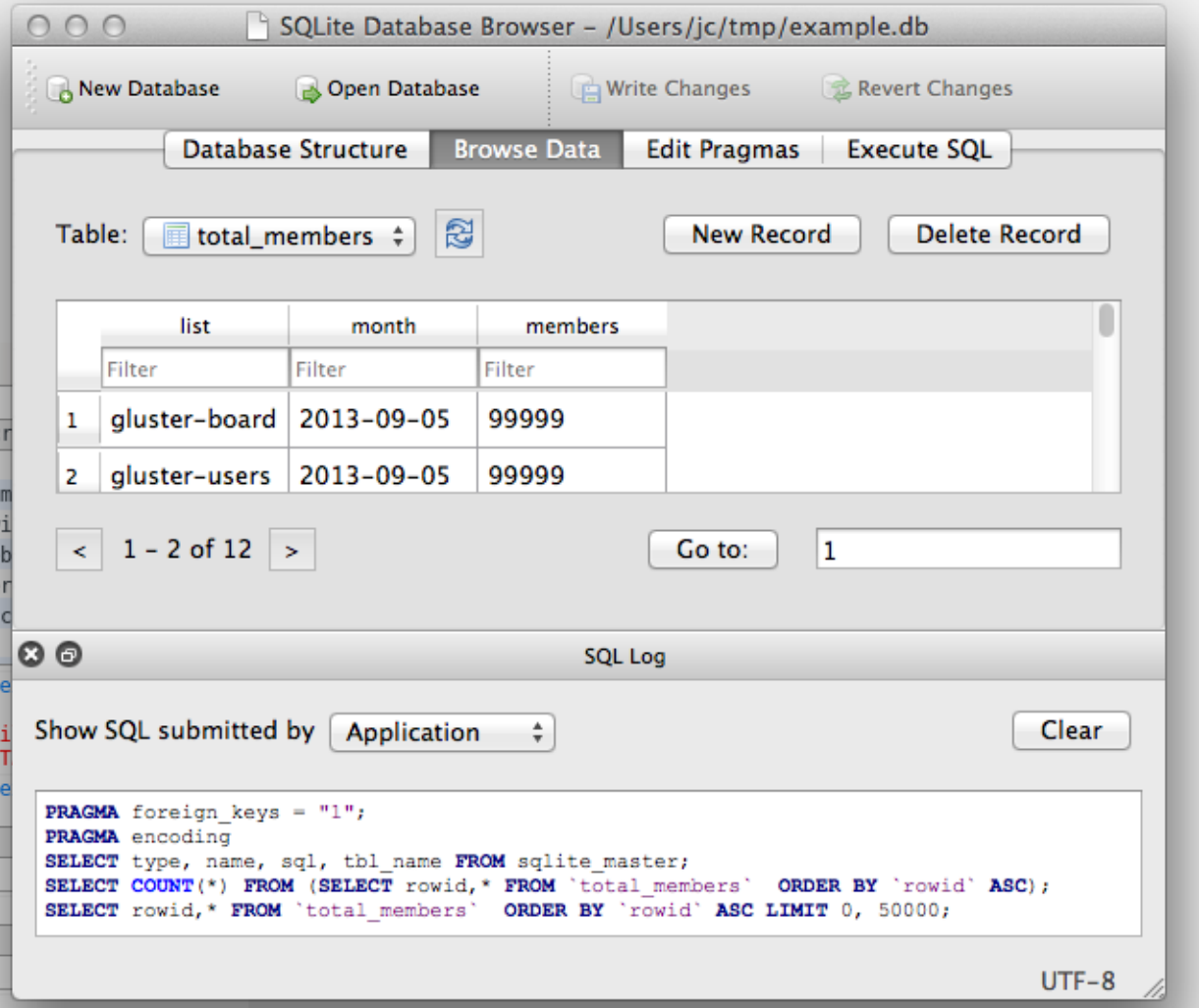
- Możemy skorzystać z klasy **SQLiteQueryBuilder** gdy chcemy zbudować bardziej skomplikowane zapytania

- W wyniku zapytań otrzymujemy obiekt **Cursor** przy pomocy którego możemy odczytywać dane z poszczególnych wierszy i kolumn tabeli

Przeglądanie/debugowanie zapisanych danych

- SqLite browser (sqlitebrowser.org)

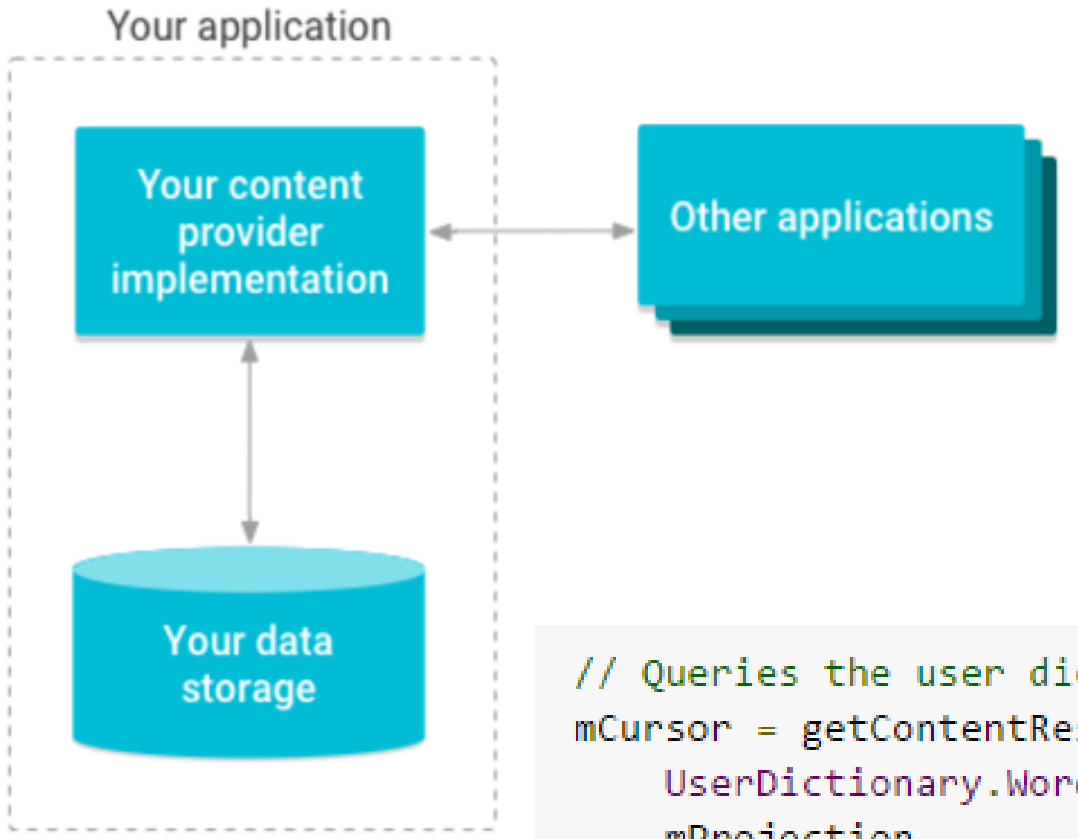




Inne biblioteki warte uwagi

- ActiveAndroid
- Sugar ORM
- DBFlow
- Cupboard
- JDXA ORM

Udostępnianie danych na zewnątrz (Content Providers)



- Abstrakcja dostępu do danych
- Zarządzenie dostępem do danych
- Powiadomienia o zmianach

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    mProjection,                        // The columns to return for each row
    mSelectionClause                    // Selection criteria
    mSelectionArgs,                     // Selection criteria
    mSortOrder);                        // The sort order for the returned rows
```