



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 domyslne");
```

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

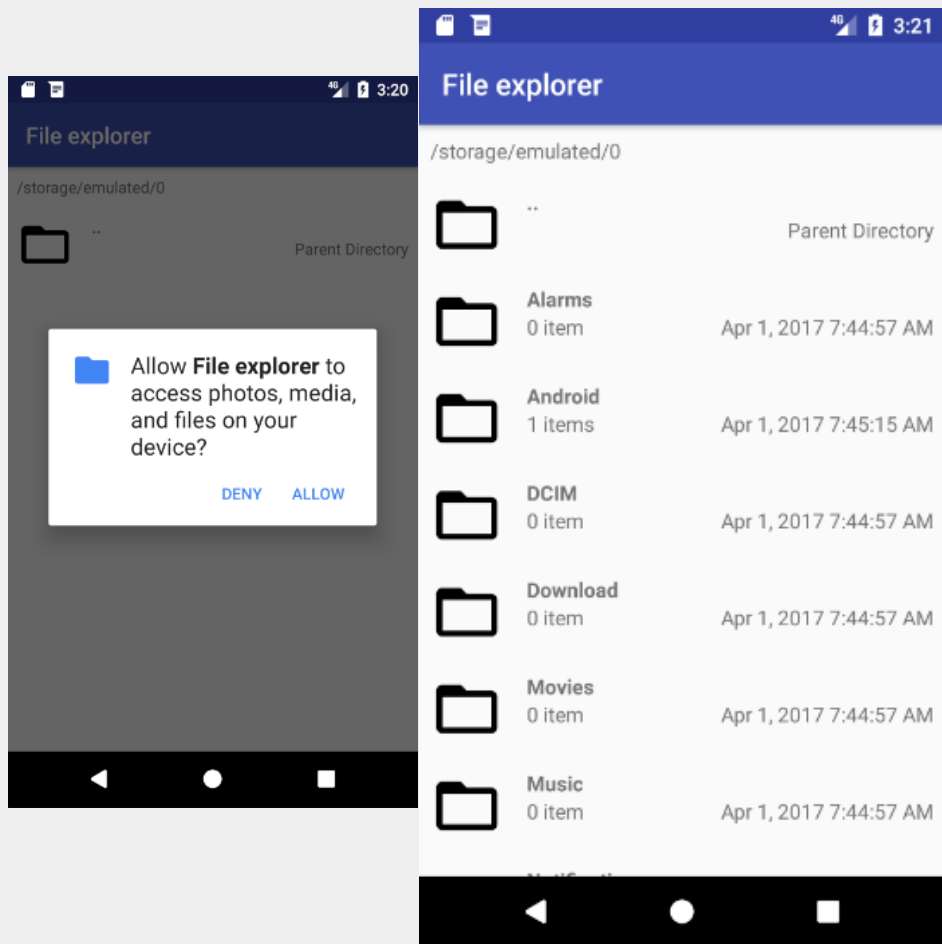
```
ActivityCompat.requestPermissions(this,
    new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, REQUEST_CODE);
```

- 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()*, *getWritableDatabase()* 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)

The screenshot shows the SQLite browser interface with the following components:

- Left Panel:** A tree view showing the database structure. The 'apod.db' database is selected, showing tables: 'android_metadata', 'rss_items', and 'sqlite_sequence'. Other databases like 'IndexedDB', 'Local Storage', 'Session Storage', 'Cookies', and 'Application Cache' are also listed.
- SQL Editor:** The main area for writing SQL queries. The current query is `SELECT * FROM rss_items;`. Below the query, the results are displayed in a table with columns: '_id', 'title', 'description_image_u...', and 'descri...'. The results show 49 rows of data, including titles like 'Our Galaxys Magneti...', 'The Milky Way over ...', 'A Twisted Solar Eru...', 'Light from Cygnus A', 'Interior View', 'Launch to Lovejoy', and 'The Complex Ion Tai...'. A red error message is visible: 'near "CONTAINS": syntax error (code 1): , while compi...'. Below the error, another query is shown: `SELECT _id, title FROM rss_items WHERE description_text CONTAINS '%comet%';`, with results showing one row: '_id' 49, 'title' 'The Complex Ion Tail of Comet Lovejoy'.
- Bottom Panel:** The 'PRAGMA user_version;' query is executed, showing the result 'user_version' 1.

The screenshot shows the SQLite Database Browser application window with the following components:

- Top Bar:** Contains buttons for 'New Database', 'Open Database', 'Write Changes', and 'Revert Changes'.
- Navigation Tabs:** 'Database Structure', 'Browse Data', 'Edit Pragma', and 'Execute SQL'. The 'Browse Data' tab is selected.
- Table Selection:** A dropdown menu shows 'total_members' as the selected table. Buttons for 'New Record' and 'Delete Record' are present.
- Data Table:** A table with columns 'list', 'month', and 'members'. The data is filtered to show rows 1 and 2. Row 1: 'gluster-board', '2013-09-05', '99999'. Row 2: 'gluster-users', '2013-09-05', '99999'. Below the table, a pagination control shows '1 - 2 of 12' and a 'Go to:' field with the value '1'.
- SQL Log:** A section for viewing SQL queries. It includes a dropdown for 'Show SQL submitted by' (set to 'Application') and a 'Clear' button. The log contains the following SQL queries:

```
PRAGMA foreign_keys = "1";
PRAGMA encoding;
SELECT type, name, sql, tbl_name FROM sqlite_master;
SELECT COUNT(*) FROM (SELECT rowid,* FROM `total_members` ORDER BY `rowid` ASC);
SELECT rowid,* FROM `total_members` ORDER BY `rowid` ASC LIMIT 0, 50000;
```
- Bottom Right:** The encoding is set to 'UTF-8'.

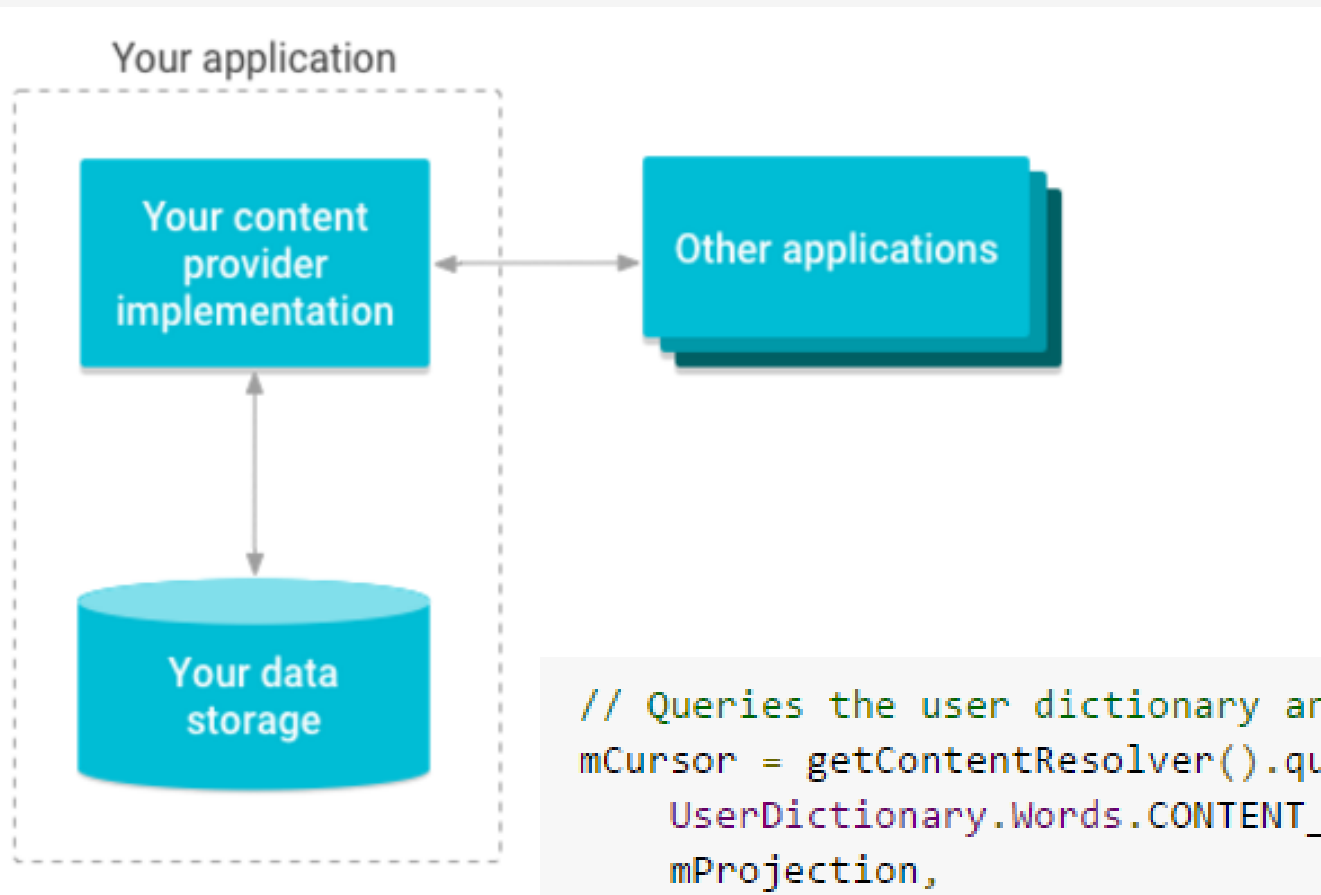
Inne biblioteki warte uwagi



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



Udostępnianie danych na zewnątrz (Content Providers)



- Abstrakcja dostępu do danych
- Zarządzanie 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
```