

SLOT 12: Content Provider

Tóm tắt

I. Giới thiệu về Content Provider

1. Content Provider là gì?
2. Chức năng và ứng dụng của Content Provider

II. Cấu trúc và cách hoạt động

3. Kiến trúc tổng quan của Content Provider
4. Cách Content Resolver và Content Provider làm việc cùng nhau

III. Các thành phần quan trọng

5. Content Resolver – Thành phần giúp truy vấn dữ liệu
6. Contract – Định nghĩa cấu trúc dữ liệu và URI
7. Content URI và cách sử dụng
8. MIME Type – Xác định kiểu dữ liệu trả về

IV. Cài đặt Content Provider

9. Cách tạo Content Provider trong Android
10. Khai báo Content Provider trong AndroidManifest.xml
11. Sử dụng Content Resolver để truy vấn dữ liệu

V. Các vấn đề quan trọng và lưu ý

12. Quản lý quyền truy cập dữ liệu (Permissions)
13. Tối ưu hóa hiệu suất và tránh rò rỉ bộ nhớ
14. Khi nào nên và không nên sử dụng Content Provider
15. Xử lý xung đột dữ liệu khi có nhiều ứng dụng truy cập



Trọng tâm của bài học:

- **Content Provider** giúp chia sẻ dữ liệu giữa các ứng dụng một cách an toàn.
- **Content Resolver** là cách duy nhất để truy vấn dữ liệu từ Content Provider.
- **Sử dụng Content URI và MIME Type** để truy xuất dữ liệu đúng cách.
- **Phải đặt quyền bảo vệ dữ liệu trong AndroidManifest.xml.**
- **Cần đóng Cursor sau khi sử dụng để tránh Memory Leak.**
- **Không nên dùng Content Provider nếu không thực sự cần thiết.**

1.1 Content Provider là gì?

- Là một thành phần quan trọng trong Android, giúp ứng dụng chia sẻ dữ liệu một cách an toàn giữa các ứng dụng khác nhau.

- Hoạt động như một **lớp trung gian** giữa dữ liệu (SQLite, API, file) và ứng dụng.
- Cho phép thực hiện **CRUD (Create, Read, Update, Delete)** với dữ liệu mà không cần truy cập trực tiếp vào database.

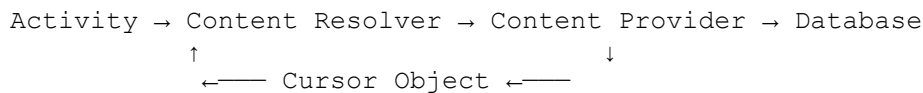
1.2 Cấu trúc và các thành phần chính

◆ Một ứng dụng sử dụng Content Provider gồm có:

1. **Dữ liệu:** Thường là SQLite, nhưng có thể là file, API, v.v.
2. **Content Provider:** Thành phần chịu trách nhiệm truy xuất và cung cấp dữ liệu.
3. **Content Resolver:** Thành phần trung gian giúp ứng dụng truy vấn dữ liệu từ Content Provider.
4. **Contract:** Định nghĩa URI, cấu trúc bảng dữ liệu và MIME Type.
5. **Android Manifest:** Cấu hình quyền truy cập và khai báo Content Provider.

◆ Quy trình hoạt động:

CSS



1.3 Các phương thức quan trọng của Content Provider

1. `query()` → Lấy dữ liệu
2. `insert()` → Thêm dữ liệu
3. `update()` → Cập nhật dữ liệu
4. `delete()` → Xóa dữ liệu
5. `getType()` → Trả về MIME Type của dữ liệu

Ví dụ:

java

```

@Override
public Uri insert(Uri uri, ContentValues values) {
    long id = database.insert("wordlist", null, values);
    return Uri.parse(CONTENT_URI + "/" + id);
}
  
```

1.4 Content URI và URI Matcher

- **Content URI** xác định dữ liệu cần truy vấn:

less

`content://com.example.provider/words`

- **Cấu trúc Content URI:**

```
less
```

```
content://authority/path/id
```

- o scheme: Luôn là content://
- o authority: Định danh của Content Provider
- o path: Định danh bảng dữ liệu
- o id: ID của bản ghi

- **URI Matcher** giúp xác định kiểu truy vấn, ví dụ:

```
java
```

```
private static final int WORDS = 1;
private static final int WORD_ID = 2;

static {
    uriMatcher.addURI(AUTHORITY, "words", WORDS);
    uriMatcher.addURI(AUTHORITY, "words/#", WORD_ID);
}
```

1.5 MIME Type trong Content Provider

MIME Type giúp xác định kiểu dữ liệu trả về:

```
arduino
```

```
vnd.android.cursor.dir/vnd.com.example.provider.words // Nhiều dữ liệu
vnd.android.cursor.item/vnd.com.example.provider.words // Một bản ghi
```

Ví dụ:

```
java
```

```
@Override
public String getType(Uri uri) {
    switch (sUriMatcher.match(uri)) {
        case WORDS:
            return "vnd.android.cursor.dir/vnd.com.example.provider.words";
        case WORD_ID:
            return "vnd.android.cursor.item/vnd.com.example.provider.words";
        default:
            return null;
    }
}
```

1.6 Cấu hình quyền truy cập trong AndroidManifest.xml

- Mặc định, mọi ứng dụng đều có thể truy cập Content Provider nếu không đặt quyền bảo vệ.
- Cần thiết lập quyền truy cập:

```
xml
```

```
<provider
    android:name=".WordListContentProvider"
    android:authorities="com.example.provider"
    android:exported="true"
```

```
android:readPermission="com.example.PERMISSION.READ"  
android:writePermission="com.example.PERMISSION.WRITE"/>
```

- Ứng dụng khác cần quyền mới có thể truy xuất dữ liệu:

xml

```
<uses-permission android:name="com.example.PERMISSION.READ"/>
```

1.7 Content Resolver – Cách ứng dụng truy xuất dữ liệu

Ứng dụng không thể truy cập trực tiếp Content Provider mà phải sử dụng **Content Resolver**.

Ví dụ lấy dữ liệu:

java


```
Cursor cursor = getContentResolver().query(  
    Uri.parse("content://com.example.provider/words"),  
    new String[] { "word", "definition" },  
    null, null, "word ASC"  
);
```

2. Những vấn đề cần lưu ý khi làm việc với Content Provider

2.1 Quyền truy cập dữ liệu (Permissions)

Phải kiểm soát quyền truy cập dữ liệu:

- Nếu không đặt quyền (`android:exported="false"`), ứng dụng khác **không thể** truy vấn dữ liệu.
- Nếu không cẩn thận, **dữ liệu nhạy cảm có thể bị lộ**.

 **Lưu ý:** Không nên sử dụng `android:exported="true"` mà không đặt quyền bảo vệ dữ liệu.

2.2 Hiệu suất và tối ưu hóa truy vấn

Cải thiện hiệu suất Content Provider bằng cách:

- **Chỉ lấy dữ liệu cần thiết** (`projection` chỉ định cột cụ thể).
- **Sử dụng chỉ mục trên SQLite** để tăng tốc truy vấn.
- **Sử dụng Loader (CursorLoader)** thay vì truy vấn trực tiếp để tránh giết lag UI.


2.3 Tránh Memory Leak khi sử dụng Cursor

 **Cursor cần được đóng** sau khi sử dụng để tránh rò rỉ bộ nhớ:

```
java
```


```
Cursor cursor = getContentResolver().query(CONTENT_URI, null, null, null, null);
try {
    while (cursor.moveToNext()) {
        String word = cursor.getString(cursor.getColumnIndex("word"));
        Log.d("ContentProvider", "Word: " + word);
    }
} finally {
    cursor.close();
}
```


2.4 Không nên sử dụng Content Provider nếu không cần thiết

 **Chỉ sử dụng Content Provider nếu cần chia sẻ dữ liệu giữa các ứng dụng.**

- Nếu dữ liệu **chỉ sử dụng nội bộ**, nên dùng SQLite, SharedPreferences hoặc Room Database thay vì Content Provider.

2.5 Tránh xung đột dữ liệu khi có nhiều ứng dụng truy cập

 Nếu nhiều ứng dụng truy cập Content Provider đồng thời, có thể xảy ra lỗi **xung đột dữ liệu**.

 **Giải pháp:**

- Sử dụng **transactions** để đảm bảo tính toàn vẹn dữ liệu.
- Định nghĩa **onConflictStrategy** khi sử dụng SQLite.

Ví dụ:

```
java
```

```
database.insertWithOnConflict("wordlist", null, values,
    SQLiteDatabase.CONFLICT_REPLACE);
```

Ứng Dụng Trong Thực Tế

Content Provider là một thành phần quan trọng trong Android giúp chia sẻ dữ liệu giữa các ứng dụng hoặc giữa các tiến trình trong cùng một ứng dụng. Trong thực tế, Content Provider được ứng dụng vào nhiều trường hợp cụ thể, từ quản lý danh bạ, thư viện ảnh, nhật ký cuộc gọi cho đến đồng bộ dữ liệu và chia sẻ dữ liệu giữa các ứng dụng.

- ✔ Truy xuất danh bạ, tin nhắn, nhật ký cuộc gọi, hình ảnh/video.
- ✔ Chia sẻ dữ liệu giữa các ứng dụng bằng Content Provider tùy chỉnh.
- ✔ Đồng bộ hóa dữ liệu với server hoặc giữa các thiết bị.
- ✔ Chia sẻ file an toàn bằng FileProvider.

1. Truy xuất dữ liệu hệ thống (System Content Providers)

Android cung cấp sẵn một số Content Provider để truy cập vào dữ liệu hệ thống như **danh bạ, tin nhắn, hình ảnh, video, nhật ký cuộc gọi...** Các ứng dụng có thể sử dụng ContentResolver để lấy dữ liệu này.

1.1. Truy xuất danh bạ điện thoại (Contacts Provider)

◆ Ứng dụng thực tế:

- Hiển thị danh sách liên hệ trong ứng dụng gọi điện, nhắn tin.
- Cho phép tìm kiếm và chọn số liên lạc từ danh bạ.

◆ Ví dụ truy xuất danh bạ:

```
java

Cursor cursor = getContentResolver().query(
    ContactsContract.Contacts.CONTENT_URI,
    null, null, null, null);

if (cursor != null && cursor.moveToFirst()) {
    do {
        String name =
cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME
));
        Log.d("CONTACT_NAME", name);
    } while (cursor.moveToNext());
    cursor.close();
}
```

1.2. Truy xuất tin nhắn SMS

◆ Ứng dụng thực tế:

- Ứng dụng quản lý tin nhắn (SMS Backup, chặn tin nhắn spam).

- Hiển thị danh sách tin nhắn của người dùng trong một ứng dụng nhắn tin.

◆ Ví dụ lấy danh sách tin nhắn SMS:

```
java

Uri uri = Uri.parse("content://sms/inbox");
Cursor cursor = getContentResolver().query(uri, null, null, null, null);

if (cursor != null && cursor.moveToFirst()) {
    do {
        String body = cursor.getString(cursor.getColumnIndex("body"));
        Log.d("SMS_BODY", body);
    } while (cursor.moveToNext());
    cursor.close();
}
```

1.3. Truy xuất lịch sử cuộc gọi (Call Log)

◆ Ứng dụng thực tế:

- Ứng dụng ghi âm cuộc gọi.
- Ứng dụng quản lý cuộc gọi, thống kê thời gian gọi.

◆ Ví dụ truy xuất lịch sử cuộc gọi:

```
java

Cursor cursor = getContentResolver().query(
    CallLog.Calls.CONTENT_URI,
    null, null, null, CallLog.Calls.DATE + " DESC");

if (cursor != null && cursor.moveToFirst()) {
    do {
        String number =
        cursor.getString(cursor.getColumnIndex(CallLog.Calls.NUMBER));
        int duration =
        cursor.getInt(cursor.getColumnIndex(CallLog.Calls.DURATION));
        Log.d("CALL_LOG", "Số: " + number + ", Thời gian: " + duration + "
        giây");
    } while (cursor.moveToNext());
    cursor.close();
}
```

1.4. Truy xuất thư viện hình ảnh, video (Media Store)

◆ Ứng dụng thực tế:

- Ứng dụng thư viện ảnh, trình phát nhạc, quản lý video.
- Truy xuất danh sách hình ảnh/video để hiển thị trong app.

◆ Ví dụ lấy danh sách hình ảnh từ bộ nhớ:

```
java
```

```

Cursor cursor = getContentResolver().query(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    null, null, null, null);

if (cursor != null && cursor.moveToFirst()) {
    do {
        String path =
cursor.getString(cursor.getColumnIndex(MediaStore.Images.Media.DATA));
        Log.d("IMAGE_PATH", path);
    } while (cursor.moveToNext());
    cursor.close();
}

```

2. Tạo Content Provider để chia sẻ dữ liệu giữa các ứng dụng

Trong nhiều ứng dụng thực tế, chúng ta cần chia sẻ dữ liệu giữa các ứng dụng khác nhau. Ví dụ:

- Một ứng dụng **ghi chú** có thể chia sẻ nội dung với một ứng dụng khác.
- Một ứng dụng **quản lý tài chính** có thể chia sẻ dữ liệu giao dịch với ứng dụng khác.

Ví dụ tạo Content Provider tùy chỉnh

◆ Ứng dụng thực tế:

- Ứng dụng quản lý công việc có thể chia sẻ danh sách nhiệm vụ với ứng dụng khác.
- Ứng dụng lịch có thể cho phép các ứng dụng khác đọc/sửa sự kiện.

Tạo Content Provider trong ứng dụng

1 Khai báo Content Provider trong AndroidManifest.xml

```

xml

<provider
    android:name=".data.MyContentProvider"
    android:authorities="com.example.myapp.provider"
    android:exported="true"
    android:grantUriPermissions="true"/>

```

2 Tạo lớp MyContentProvider.java

```

java

public class MyContentProvider extends ContentProvider {
    private static final String AUTHORITY = "com.example.myapp.provider";
    private static final UriMatcher uriMatcher = new
UriMatcher(UriMatcher.NO_MATCH);

    static {
        uriMatcher.addURI(AUTHORITY, "tasks", 1);
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();

```



```

        return db.query("tasks", projection, selection, selectionArgs, null,
null, sortOrder);
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        long id = db.insert("tasks", null, values);
        return Uri.parse("content://" + AUTHORITY + "/tasks/" + id);
    }
}

```

👉 Ứng dụng khác có thể truy vấn dữ liệu bằng ContentResolver!

3. Đồng bộ hóa dữ liệu giữa các thiết bị hoặc với server

◆ Ứng dụng thực tế:

- Ứng dụng lịch hoặc ghi chú có thể đồng bộ dữ liệu giữa các thiết bị.
- Ứng dụng sức khỏe có thể đồng bộ bước chân với Google Fit.

◆ Cách thực hiện:

- Kết hợp Content Provider với **SyncAdapter** hoặc **WorkManager** để đồng bộ dữ liệu tự động.

Ví dụ sử dụng SyncAdapter để đồng bộ dữ liệu giữa local database và server:

```

java

public void onPerformSync(Account account, Bundle extras, String authority,
ContentProviderClient provider, SyncResult syncResult) {
    // Gửi dữ liệu từ Content Provider lên server
}

```

4. Chia sẻ dữ liệu bảo mật bằng FileProvider

◆ Ứng dụng thực tế:

- Chia sẻ ảnh/video giữa các ứng dụng một cách an toàn.
- Gửi file đính kèm trong email hoặc tin nhắn.

◆ Cách sử dụng FileProvider để chia sẻ file:

1 Thêm FileProvider vào AndroidManifest.xml

```

xml

<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="com.example.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
</provider>

```

2 Tạo file file_paths.xml để định nghĩa thư mục được chia sẻ

xml

```
<paths>
    <external-files-path name="shared_images" path="Pictures/" />
</paths>
```

3 Chia sẻ file ảnh

java

```
File file = new File(getExternalFilesDir(Environment.DIRECTORY_PICTURES),
    "image.jpg");
Uri uri = FileProvider.getUriForFile(context, "com.example.fileprovider",
    file);
intent.setDataAndType(uri, "image/*");
intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
```

Một số nội dung chi tiết

1. Content Provider là gì?

Content Provider là một thành phần trong Android cho phép các ứng dụng chia sẻ dữ liệu một cách có tổ chức và bảo mật. Nó giúp trừu tượng hóa quá trình truy cập dữ liệu, nghĩa là ứng dụng không cần biết dữ liệu được lưu trữ ở đâu hoặc dưới định dạng nào.

Chức năng chính của Content Provider:

- Cung cấp một giao diện thống nhất để truy cập dữ liệu.
- Hỗ trợ thao tác dữ liệu với các phương thức `query()`, `insert()`, `update()`, `delete()`.
- Quản lý quyền truy cập bằng cách cấp phép trong `AndroidManifest.xml`.
- Có thể lưu dữ liệu ở nhiều nguồn khác nhau (SQLite, tệp tin, API, v.v.).

2. Content Resolver là gì?

Content Resolver là lớp trung gian giúp ứng dụng giao tiếp với Content Provider. Thay vì truy cập trực tiếp vào cơ sở dữ liệu, ứng dụng sử dụng Content Resolver để gửi yêu cầu đến Content Provider.

Các phương thức chính của Content Resolver:

- `query(Uri, projection, selection, selectionArgs, sortOrder)`: Truy vấn dữ liệu.
- `insert(Uri, ContentValues)`: Thêm dữ liệu mới.
- `update(Uri, ContentValues, selection, selectionArgs)`: Cập nhật dữ liệu.
- `delete(Uri, selection, selectionArgs)`: Xóa dữ liệu.

Ví dụ truy vấn với Content Resolver:

```
java
Cursor cursor = getContentResolver().query(
    Uri.parse("content://com.example.provider/words"),
    new String[] { "word", "definition" },
    "word LIKE ?",
    new String[] { "%Android%" },
    "word ASC"
);
```

3. Cách hoạt động của Content Provider và Content Resolver

Content Resolver và Content Provider hoạt động theo cơ chế client-server:

1. **Activity/Adapter** gọi Content Resolver để gửi yêu cầu dữ liệu.
2. **Content Resolver** gửi yêu cầu đến Content Provider.
3. **Content Provider** truy xuất dữ liệu từ nguồn (SQLite, backend, file, v.v.).
4. **Content Resolver** nhận kết quả và chuyển về Activity dưới dạng `Cursor`.

CSS

Ví dụ lấy dữ liệu từ Content Provider:

java

4. Lợi ích của Content Provider

- ## 5. Các thành phần chính của một ứng dụng sử dụng Content Provider

- ## 6. Contract là gì?

Chức năng của Contract:

- 12

Ví dụ về lớp Contract:

```
java

public final class WordContract {
    public static final String AUTHORITY = "com.example.provider";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY
+ "/words");
    public static final String TABLE_NAME = "wordlist";
}
```

7. Content URI là gì?

Content URI là đường dẫn duy nhất đại diện cho một tập dữ liệu trong Content Provider.

Cấu trúc của Content URI:

less

content://authority/path/id

- scheme: Luôn là content://
- authority: Định danh của Content Provider.
- path: Định danh bảng dữ liệu.
- id: ID của bản ghi dữ liệu (tùy chọn).

Ví dụ về Content URI:

```
java

public static final Uri CONTENT_URI =
    Uri.parse("content://com.example.provider/words");
```

8. MIME Type trong Content Provider

MIME Type giúp ứng dụng xác định kiểu dữ liệu trả về từ Content Provider.

Cấu trúc MIME Type:

bash

vnd.android.cursor.dir/vnd.<authority>.<type>

- **vnd.android.cursor.item/** → Dữ liệu đơn.
- **vnd.android.cursor.dir/** → Danh sách dữ liệu.

Ví dụ:

```
java

@Override
public String getType(Uri uri) {
    switch (sUriMatcher.match(uri)) {
        case URI_ALL_ITEMS_CODE:
            return "vnd.android.cursor.dir/vnd.com.example.provider.words";
    }
}
```

```

        case URI_ONE_ITEM_CODE:
            return "vnd.android.cursor.item/vnd.com.example.provider.words";
        default:
            return null;
    }
}

```

9. Cài đặt Content Provider

Để triển khai Content Provider, cần thực hiện:

1. **Kế thừa `ContentProvider`**
2. **Override các phương thức:**
 - o `query()`
 - o `insert()`
 - o `update()`
 - o `delete()`
 - o `getType()`
3. **Định nghĩa URI Matcher để xử lý yêu cầu.**
4. **Khai báo Content Provider trong `AndroidManifest.xml`.**

Ví dụ cài đặt `insert()`:

```

java
@Override
public Uri insert(Uri uri, ContentValues values) {
    long id = database.insert("wordlist", null, values);
    return Uri.parse(CONTENT_URI + "/" + id);
}

```

10. Khai báo Content Provider trong `AndroidManifest.xml`

Content Provider phải được khai báo trong `AndroidManifest.xml` với quyền truy cập:

```

xml
<provider
    android:name=".WordListContentProvider"
    android:authorities="com.example.provider"
    android:exported="true"
    android:readPermission="com.example.PERMISSION.READ"
    android:writePermission="com.example.PERMISSION.WRITE"/>

```

Ứng dụng khác cần khai báo quyền:

```

xml
<uses-permission android:name="com.example.PERMISSION.READ"/>

```

11. Content Resolver và cách gọi

Ứng dụng truy xuất dữ liệu thông qua Content Resolver:

```

java

```

```
Cursor cursor = getContentResolver().query(CONTENT_URI, null, null, null, null);
```

Các phương thức:

- `query()`
- `insert()`
- `update()`
- `delete()`

12. Tổng kết các bước triển khai Content Provider

1. **Tạo cơ sở dữ liệu (SQLite hoặc backend khác).**
2. **Viết contract để định nghĩa Content URI, cấu trúc bảng.**
3. **Tạo Content Provider kế thừa `ContentProvider`.**
4. **Override các phương thức: `query()`, `insert()`, `update()`, `delete()`.**
5. **Khai báo Content Provider trong `AndroidManifest.xml`.**
6. **Sử dụng Content Resolver để truy xuất dữ liệu.**

1. Content Provider là gì?

Content Provider là một thành phần trong Android cho phép các ứng dụng chia sẻ dữ liệu một cách có tổ chức và bảo mật. Nó giúp trừu tượng hóa quá trình truy cập dữ liệu, nghĩa là ứng dụng không cần biết dữ liệu được lưu trữ ở đâu hoặc dưới định dạng nào.

Chức năng chính của Content Provider:

- Cung cấp một giao diện thống nhất để truy cập dữ liệu.
- Hỗ trợ thao tác dữ liệu với các phương thức `query()`, `insert()`, `update()`, `delete()`.
- Quản lý quyền truy cập bằng cách cấp phép trong `AndroidManifest.xml`.
- Có thể lưu dữ liệu ở nhiều nguồn khác nhau (SQLite, tệp tin, API, v.v.).

2. Content Resolver là gì?

Content Resolver là lớp trung gian giúp ứng dụng giao tiếp với Content Provider. Thay vì truy cập trực tiếp vào cơ sở dữ liệu, ứng dụng sử dụng Content Resolver để gửi yêu cầu đến Content Provider.

Các phương thức chính của Content Resolver:

- `query(Uri, projection, selection, selectionArgs, sortOrder)`: Truy vấn dữ liệu.
- `insert(Uri, ContentValues)`: Thêm dữ liệu mới.
- `update(Uri, ContentValues, selection, selectionArgs)`: Cập nhật dữ liệu.
- `delete(Uri, selection, selectionArgs)`: Xóa dữ liệu.

Ví dụ truy vấn với Content Resolver:

```
java
```

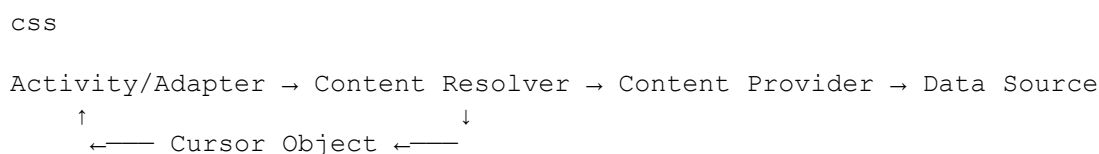
```
Cursor cursor = getContentResolver().query(
    Uri.parse("content://com.example.provider/words"),
    new String[] { "word", "definition" },
    "word LIKE ?",
    new String[] { "%Android%" },
    "word ASC"
);
```

3. Cách hoạt động của Content Provider và Content Resolver

Content Resolver và Content Provider hoạt động theo cơ chế client-server:

5. **Activity/Adapter** gọi Content Resolver để gửi yêu cầu dữ liệu.
6. **Content Resolver** gửi yêu cầu đến Content Provider.
7. **Content Provider** truy xuất dữ liệu từ nguồn (SQLite, backend, file, v.v.).
8. **Content Resolver** nhận kết quả và chuyển về Activity dưới dạng `Cursor`.

Sơ đồ hoạt động:



Ví dụ lấy dữ liệu từ Content Provider:

```
java
Cursor cursor = getContentResolver().query(CONTENT_URI, null, null, null,
null);
if (cursor != null) {
    while (cursor.moveToNext()) {
        String word = cursor.getString(cursor.getColumnIndex("word"));
        Log.d("ContentProvider", "Word: " + word);
    }
    cursor.close();
}
```

4. Lợi ích của Content Provider

- **Chia sẻ dữ liệu an toàn giữa các ứng dụng.**
→ Ví dụ: Ứng dụng A lưu danh bạ, ứng dụng B có thể đọc danh bạ qua Content Provider.
- **Quản lý quyền truy cập chi tiết** (chỉ đọc, chỉ ghi, hoặc cả hai).
- **Tạo lớp trừu tượng giúp quản lý dữ liệu dễ dàng hơn.**
- **Cung cấp giao diện chuẩn hóa** để truy cập dữ liệu từ nhiều nguồn khác nhau (SQLite, API, tệp tin).

5. Các thành phần chính của một ứng dụng sử dụng Content Provider

7. **Data:** Dữ liệu thường được lưu trữ trong SQLite, nhưng có thể là backend khác.
8. **OpenHelper:** Quản lý SQLite Database.

9. **Contract:** Chứa thông tin về Content Provider.
10. **Content Provider:** Giao diện xử lý dữ liệu.
11. **Content Resolver:** Thành phần gửi yêu cầu và xử lý kết quả.
12. **Manifest Permissions:** Quy định quyền truy cập.

6. Contract là gì?

Contract là một lớp công khai giúp Content Provider dễ dàng được sử dụng bởi các ứng dụng khác.

Chức năng của Contract:

- **Định nghĩa URI** để truy vấn dữ liệu.
- **Cấu trúc bảng** và tên cột trong cơ sở dữ liệu.
- **MIME Type** giúp định dạng dữ liệu trả về.
- **Các hằng số** giúp dễ dàng thay đổi khi cần.

Ví dụ về lớp Contract:

```
java

public final class WordContract {
    public static final String AUTHORITY = "com.example.provider";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY
+ "/words");
    public static final String TABLE_NAME = "wordlist";
}
```

7. Content URI là gì?

Content URI là đường dẫn duy nhất đại diện cho một tập dữ liệu trong Content Provider.

Cấu trúc của Content URI:

```
less
```

```
content://authority/path/id
```

- **scheme:** Luôn là content://
- **authority:** Định danh của Content Provider.
- **path:** Định danh bảng dữ liệu.
- **id:** ID của bản ghi dữ liệu (tùy chọn).

Ví dụ về Content URI:

```
java
```

```
public static final Uri CONTENT_URI =
    Uri.parse("content://com.example.provider/words");
```

8. MIME Type trong Content Provider

MIME Type giúp ứng dụng xác định kiểu dữ liệu trả về từ Content Provider.

Cấu trúc MIME Type:

bash

```
vnd.android.cursor.dir/vnd.<authority>.<type>
```

- **vnd.android.cursor.item/** → Dữ liệu đơn.
- **vnd.android.cursor.dir/** → Danh sách dữ liệu.

Ví dụ:

java

```
@Override
public String getType(Uri uri) {
    switch (sUriMatcher.match(uri)) {
        case URI_ALL_ITEMS_CODE:
            return "vnd.android.cursor.dir/vnd.com.example.provider.words";
        case URI_ONE_ITEM_CODE:
            return "vnd.android.cursor.item/vnd.com.example.provider.words";
        default:
            return null;
    }
}
```

9. Cài đặt Content Provider

Để triển khai Content Provider, cần thực hiện:

5. **Kế thừa `ContentProvider`**
6. **Override các phương thức:**
 - o `query()`
 - o `insert()`
 - o `update()`
 - o `delete()`
 - o `getType()`
7. **Định nghĩa URI Matcher để xử lý yêu cầu.**
8. **Khai báo Content Provider trong `AndroidManifest.xml`.**

Ví dụ cài đặt `insert()`:

java

```
@Override
public Uri insert(Uri uri, ContentValues values) {
    long id = database.insert("wordlist", null, values);
    return Uri.parse(CONTENT_URI + "/" + id);
}
```

10. Khai báo Content Provider trong AndroidManifest.xml

Content Provider phải được khai báo trong `AndroidManifest.xml` với quyền truy cập:

xml

```
<provider
    android:name=".WordListContentProvider"
    android:authorities="com.example.provider"
    android:exported="true"
    android:readPermission="com.example.PERMISSION.READ"
    android:writePermission="com.example.PERMISSION.WRITE"/>
```

Ứng dụng khác cần khai báo quyền:

xml

```
<uses-permission android:name="com.example.PERMISSION.READ"/>
```

11. Content Resolver và cách gọi

Ứng dụng truy xuất dữ liệu thông qua Content Resolver:

java

```
Cursor cursor = getContentResolver().query(CONTENT_URI, null, null, null, null);
```

Các phương thức:

- `query()`
- `insert()`
- `update()`
- `delete()`

12. Tổng kết các bước triển khai Content Provider

1. **Tạo cơ sở dữ liệu (SQLite hoặc backend khác).**
2. **Viết contract để định nghĩa Content URI, cấu trúc bảng.**
3. **Tạo Content Provider kế thừa `ContentProvider`.**
4. **Override các phương thức: `query()`, `insert()`, `update()`, `delete()`.**
5. **Khai báo Content Provider trong `AndroidManifest.xml`.**
6. **Sử dụng Content Resolver để truy xuất dữ liệu.**

KINH NGHIỆM THỰC TIỄN

Content Provider là một thành phần quan trọng trong Android, giúp ứng dụng chia sẻ dữ liệu một cách an toàn giữa các ứng dụng khác nhau. Tuy nhiên, khi sử dụng Content Provider trong thực tế, có một số vấn đề và kinh nghiệm thực tiễn cần lưu ý.

- **Content Provider rất hữu ích** khi chia sẻ dữ liệu giữa ứng dụng, nhưng không nên lạm dụng.
- **Bảo mật dữ liệu** là yếu tố quan trọng nhất khi triển khai Content Provider.
- **Hiệu suất truy vấn** cần được tối ưu hóa để tránh ảnh hưởng đến UI.
- **Cập nhật theo phiên bản Android** để đảm bảo tương thích với các thiết bị mới

1. Khi nào nên sử dụng Content Provider?

Content Provider chủ yếu được sử dụng trong các trường hợp sau:

- Khi bạn muốn chia sẻ dữ liệu giữa **các ứng dụng khác nhau**.
- Khi dữ liệu cần được truy cập từ **nhiều quy trình (processes)**.
- Khi bạn muốn sử dụng **CursorLoader** để tải dữ liệu bất đồng bộ.
- Khi bạn cần truy cập dữ liệu từ **các Content Provider hệ thống** (Contacts, Media Store, Call Logs...).

❌ Khi nào KHÔNG cần dùng Content Provider?

- Nếu dữ liệu chỉ được sử dụng **nội bộ trong ứng dụng**, hãy dùng **Room Database** hoặc **SharedPreferences**.
- Nếu không cần truy cập từ nhiều process, có thể dùng **LiveData + ViewModel** thay vì Content Provider.

2. Kinh nghiệm thực tiễn khi sử dụng Content Provider

2.1. Bảo mật dữ liệu

◆ **Vấn đề:** Nếu Content Provider không được bảo mật đúng cách, các ứng dụng bên ngoài có thể truy cập dữ liệu mà không có sự cho phép.

◆ **Giải pháp:**

- **Hạn chế quyền truy cập** bằng cách đặt `android:exported="false"` trong `AndroidManifest.xml` nếu không muốn ứng dụng khác truy cập.
- Nếu cần chia sẻ dữ liệu, sử dụng **permission kiểm soát truy cập**:

xml

```
<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.myapp.provider"
    android:exported="true"
    android:grantUriPermissions="true"
    android:permission="com.example.myapp.READ_DATA">
</provider>
```

- Nếu chỉ muốn cấp quyền tạm thời, sử dụng **Uri permission** thay vì cấp quyền toàn cục:

```
java

grantUriPermission("com.example.otherapp", uri,
Intent.FLAG_GRANT_READ_URI_PERMISSION);
```

2.2. Tránh rò rỉ bộ nhớ (Memory Leak) khi sử dụng Cursor

◆ **Vấn đề:** Cursor không được đóng sau khi sử dụng có thể gây rò rỉ bộ nhớ.

◆ **Giải pháp:**

- Luôn đóng **Cursor** sau khi sử dụng bằng cách gọi `.close()`.

```
java

Cursor cursor = getContentResolver().query(uri, null, null, null,
null);
try {
    if (cursor != null && cursor.moveToFirst()) {
        // Xử lý dữ liệu
    }
} finally {
    if (cursor != null) {
        cursor.close();
    }
}
```

- Nếu sử dụng **LoaderManager** hoặc **LiveData**, hệ thống sẽ tự động quản lý việc đóng **Cursor**.

2.3. Hiệu suất truy vấn dữ liệu

◆ **Vấn đề:**

- Truy vấn với **ContentResolver** có thể chậm nếu truy xuất dữ liệu lớn.
- Lạm dụng **Content Provider** có thể gây **hiệu suất kém**.

◆ **Giải pháp:**

- **Sử dụng Projection** để chỉ lấy các cột cần thiết, tránh **SELECT ***:

```
java

String[] projection = {ContactsContract.Contacts.DISPLAY_NAME,
ContactsContract.Contacts._ID};
Cursor cursor =
getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,
projection, null, null, null);
```

- **Sử dụng Selection và SelectionArgs** để lọc dữ liệu thay vì lấy tất cả.

```

java

String selection = ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?";
String[] selectionArgs = {"%John%"};
Cursor cursor =
getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,
projection, selection, selectionArgs, null);

```

- Dùng **Loader (CursorLoader)** hoặc **LiveData** để xử lý bất đồng bộ, tránh làm chậm UI Thread.

2.4. Quản lý xung đột dữ liệu khi có nhiều ứng dụng truy cập

◆ **Vấn đề:** Nếu nhiều ứng dụng ghi vào cùng một Content Provider, có thể xảy ra **xung đột dữ liệu**.

◆ **Giải pháp:**

- Dùng **transaction** để đảm bảo tính toàn vẹn dữ liệu.

```

java

SQLiteDatabase db = dbHelper.getWritableDatabase();
db.beginTransaction();
try {
    // Chèn dữ liệu
    db.setTransactionSuccessful();
} finally {
    db.endTransaction();
}

```

- Sử dụng **ContentObserver** để lắng nghe thay đổi dữ liệu và cập nhật UI.

2.5. Đảm bảo tính tương thích giữa các phiên bản Android

◆ **Vấn đề:** Một số Content Provider hệ thống thay đổi trong các phiên bản Android mới.

◆ **Giải pháp:**

- Luôn kiểm tra phiên bản Android trước khi truy vấn dữ liệu.

```

java

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
    // Cách truy vấn mới
} else {
    // Cách truy vấn cũ
}

```

- Đọc tài liệu về các thay đổi API của Google trước khi nâng cấp SDK.

2.6. Xử lý quyền truy cập dữ liệu trên Android 10+

◆ Vấn đề:

- Từ Android 10 (API 29), các ứng dụng không thể truy cập trực tiếp vào bộ nhớ ngoài bằng đường dẫn tĩnh.
- **Scoped Storage** yêu cầu sử dụng `ContentResolver` để truy cập file media.

◆ Giải pháp:

- **Sử dụng MediaStore API** thay vì truy cập đường dẫn tĩnh.

```
java
```

```
ContentValues values = new ContentValues();
values.put(MediaStore.Images.Media.DISPLAY_NAME, "my_image.jpg");
values.put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg");
Uri imageUri =
getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);
```

- **Yêu cầu quyền truy cập file cụ thể thay vì toàn bộ bộ nhớ.**

3. Tổng kết

| Vấn đề | Giải pháp |
|---------------------------------|---|
| Bảo mật dữ liệu | Hạn chế quyền truy cập, dùng Uri Permission thay vì cấp quyền toàn cục. |
| Memory Leak với Cursor | Luôn đóng Cursor sau khi sử dụng hoặc dùng Loader/LiveData. |
| Hiệu suất chậm khi truy vấn | Chỉ lấy cột cần thiết, dùng selection, CursorLoader để load bất đồng bộ. |
| Xung đột dữ liệu | Dùng transaction khi ghi dữ liệu, sử dụng ContentObserver để cập nhật UI. |
| Tương thích phiên bản Android | Kiểm tra API trước khi truy vấn, cập nhật code theo tài liệu Android. |
| Scoped Storage trên Android 10+ | Sử dụng MediaStore API thay vì truy cập file trực tiếp. |

4. Kết luận

- **Content Provider** rất hữu ích khi chia sẻ dữ liệu giữa ứng dụng, nhưng không nên lạm dụng.
- **Bảo mật dữ liệu** là yếu tố quan trọng nhất khi triển khai Content Provider.
- **Hiệu suất truy vấn** cần được tối ưu hóa để tránh ảnh hưởng đến UI.
- **Cập nhật theo phiên bản Android** để đảm bảo tương thích với các thiết bị mới

EXERCISE

Quản lý Danh sách Từ vựng (Word List)

1. Mô tả dự án

Mục tiêu

Xây dựng một ứng dụng Android đơn giản sử dụng **Content Provider** để quản lý danh sách từ vựng (**Word List**). Ứng dụng sẽ cho phép người dùng **thêm, sửa, xóa và hiển thị từ vựng**, đồng thời cung cấp khả năng **truy xuất dữ liệu từ ứng dụng khác** bằng Content Resolver.

2. Công nghệ sử dụng

- **Ngôn ngữ:** Java
- **Cơ sở dữ liệu:** SQLite
- **Thành phần chính:**
 - **Content Provider** để chia sẻ dữ liệu
 - **Content Resolver** để truy vấn dữ liệu
 - **RecyclerView** để hiển thị danh sách từ vựng
 - **Room Database** (hoặc SQLite) để lưu trữ dữ liệu

3. Các bước thực hiện

1 Khởi tạo dự án Android

Bước 1: Tạo dự án mới trong Android Studio

1. **Mở Android Studio** → Chọn **New Project**.
2. Chọn **Empty Activity**, đặt tên ứng dụng là **WordListApp**.
3. Chọn **Java** làm ngôn ngữ lập trình.
4. Nhấn **Finish** để tạo dự án.

2 Cấu hình Room Database

Bước 2: Thêm dependencies vào build.gradle (Module: app)

Mở file `build.gradle` (Module: app) và thêm các dependencies sau:

```
gradle

dependencies {
    implementation 'androidx.recyclerview:recyclerview:1.2.1'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.1'
    implementation 'androidx.room:room-runtime:2.5.2'
    annotationProcessor 'androidx.room:room-compiler:2.5.2'
}
```


Nhấn **Sync Now** để cập nhật dependencies.

3 Tạo Room Database

Bước 3: Tạo Entity cho từ vựng

Tạo package **model**, sau đó tạo file `Word.java`:

```
java

package com.example.wordlistapp.model;

import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "word_table")
public class Word {
    @PrimaryKey(autoGenerate = true)
    private int id;
    private String word;
    private String meaning;

    public Word(String word, String meaning) {
        this.word = word;
        this.meaning = meaning;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getWord() { return word; }
    public String getMeaning() { return meaning; }
}
```

Bước 4: Tạo DAO để truy vấn dữ liệu

Tạo file `WordDao.java` trong package **dao**:

```
java

package com.example.wordlistapp.dao;

import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Delete;
import java.util.List;
import com.example.wordlistapp.model.Word;

@Dao
public interface WordDao {
    @Insert
    void insert(Word word);

    @Query("SELECT * FROM word_table ORDER BY word ASC")
    LiveData<List<Word>> getAllWords();
}
```

```

        @Delete
        void delete(Word word);
    }

```

Bước 5: Tạo Database

Tạo file `WordDatabase.java` trong package **database**:

```

java

package com.example.wordlistapp.database;

import android.content.Context;
import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import com.example.wordlistapp.dao.WordDao;
import com.example.wordlistapp.model.Word;

@Database(entities = {Word.class}, version = 1)
public abstract class WordDatabase extends RoomDatabase {
    public abstract WordDao wordDao();
    private static volatile WordDatabase INSTANCE;

    public static WordDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (WordDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE =
Room.databaseBuilder(context.getApplicationContext(),
                        WordDatabase.class, "word_database")
                            .fallbackToDestructiveMigration()
                            .build();
                }
            }
        }
        return INSTANCE;
    }
}

```

4 Tạo Content Provider

Bước 6: Tạo Content Provider

Tạo file `WordContentProvider.java` trong package **provider**:

```

java

package com.example.wordlistapp.provider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.net.Uri;
import androidx.annotation.NonNull;

```

```

import androidx.annotation.Nullable;
import com.example.wordlistapp.dao.WordDao;
import com.example.wordlistapp.database.WordDatabase;
import com.example.wordlistapp.model.Word;

public class WordContentProvider extends ContentProvider {
    private static final String AUTHORITY =
"com.example.wordlistapp.provider";
    private static final String TABLE_NAME = "word_table";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY
+ "/" + TABLE_NAME);

    private static final int WORDS = 1;
    private static final int WORD_ID = 2;
    private static final UriMatcher uriMatcher = new
UriMatcher(UriMatcher.NO_MATCH);

    static {
        uriMatcher.addURI(AUTHORITY, TABLE_NAME, WORDS);
        uriMatcher.addURI(AUTHORITY, TABLE_NAME + "/" + "#", WORD_ID);
    }

    private WordDao wordDao;

    @Override
    public boolean onCreate() {
        wordDao = WordDatabase.getDatabase(getContext()).wordDao();
        return true;
    }

    @Nullable
    @Override
    public Cursor query(@NonNull Uri uri, @Nullable String[] projection,
@Nullable String selection,
                        @Nullable String[] selectionArgs, @Nullable String
sortOrder) {
        return wordDao.getAllWords().getValue(); // Trả về danh sách từ vựng
    }

    @Nullable
    @Override
    public Uri insert(@NonNull Uri uri, @Nullable ContentValues values) {
        String word = values.getAsString("word");
        String meaning = values.getAsString("meaning");
        Word newWord = new Word(word, meaning);
        wordDao.insert(newWord);
        return ContentUris.withAppendedId(CONTENT_URI, newWord.getId());
    }

    @Override
    public int delete(@NonNull Uri uri, @Nullable String selection, @Nullable
String[] selectionArgs) {
        return 0; // Chưa hỗ trợ xóa
    }
}

```

Bước 7: Khai báo Content Provider trong `AndroidManifest.xml`

Thêm đoạn sau vào `<application>`:

```
xml
```

```
<provider
    android:name=".provider.WordContentProvider"
    android:authorities="com.example.wordlistapp.provider"
    android:exported="true"
    android:grantUriPermissions="true"/>
```

5 Hiển thị danh sách từ vựng với RecyclerView

Bước 8: Tạo RecyclerView Adapter

Tạo file WordAdapter.java trong package adapter:

```
java

package com.example.wordlistapp.adapter;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import com.example.wordlistapp.R;
import com.example.wordlistapp.model.Word;
import java.util.List;

public class WordAdapter extends
    RecyclerView.Adapter<WordAdapter.WordViewHolder> {
    private List<Word> words;

    public void setWords(List<Word> words) {
        this.words = words;
        notifyDataSetChanged();
    }

    @NonNull
    @Override
    public WordViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
        viewType) {
        View view =
        LayoutInflater.from(parent.getContext()).inflate(R.layout.word_item, parent,
        false);
        return new WordViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull WordViewHolder holder, int
        position) {
        holder.wordText.setText(words.get(position).getWord());
        holder.meaningText.setText(words.get(position).getMeaning());
    }

    @Override
    public int getItemCount() { return words == null ? 0 : words.size(); }

    static class WordViewHolder extends RecyclerView.ViewHolder {
        TextView wordText, meaningText;
        WordViewHolder(View itemView) {
            super(itemView);
        }
    }
}
```

```

        wordText = itemView.findViewById(R.id.wordText);
        meaningText = itemView.findViewById(R.id.meaningText);
    }
}
}

```

6 Thiết kế Giao diện

Bước 9: Cập nhật layout chính (activity_main.xml)

Mở res/layout/activity_main.xml và chỉnh sửa:

```

xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <Button
        android:id="@+id/buttonAddWord"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Thêm từ mới" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingTop="8dp" />
</LinearLayout>

```

Bước 10: Tạo layout cho từng item trong RecyclerView

Tạo file res/layout/word_item.xml:

```

xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="8dp"
    android:background="@android:color/darker_gray"
    android:marginBottom="8dp">

    <TextView
        android:id="@+id/wordText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:textStyle="bold"
        android:textColor="@android:color/white" />

    <TextView

```

```

        android:id="@+id/meaningText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="14sp"
        android:textColor="@android:color/white" />
</LinearLayout>

```

7 Viết Code cho MainActivity

Bước 11: Cập nhật MainActivity.java

Tạo RecyclerView, xử lý thêm từ vựng mới.

Mở file MainActivity.java:

```

java

package com.example.wordlistapp;

import android.content.ContentValues;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.example.wordlistapp.adapter.WordAdapter;
import com.example.wordlistapp.model.Word;
import com.example.wordlistapp.provider.WordContentProvider;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
    private static final int ADD_WORD_REQUEST = 1;
    private RecyclerView recyclerView;
    private WordAdapter adapter;
    private List<Word> wordList = new ArrayList<>();
    private Button buttonAddWord;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        adapter = new WordAdapter();
        recyclerView.setAdapter(adapter);

        buttonAddWord = findViewById(R.id.buttonAddWord);
        buttonAddWord.setOnClickListener(v -> {
            Intent intent = new Intent(MainActivity.this,
AddWordActivity.class);
            startActivityForResult(intent, ADD_WORD_REQUEST);
        });
    }
}

```

```

        loadWordsFromContentProvider();
    }

    private void loadWordsFromContentProvider() {
        wordList.clear();
        Uri uri = WordContentProvider.CONTENT_URI;
        Cursor cursor = getContentResolver().query(uri, null, null, null,
null);

        if (cursor != null) {
            while (cursor.moveToNext()) {
                int id = cursor.getInt(cursor.getColumnIndexOrThrow("id"));
                String word =
cursor.getString(cursor.getColumnIndexOrThrow("word"));
                String meaning =
cursor.getString(cursor.getColumnIndexOrThrow("meaning"));
                wordList.add(new Word(word, meaning));
            }
            cursor.close();
        }
        adapter.setWords(wordList);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == ADD_WORD_REQUEST && resultCode == RESULT_OK) {
            loadWordsFromContentProvider();
        }
    }
}

```

8 Tạo Activity để Thêm Từ Mới

Bước 12: Tạo giao diện `activity_add_word.xml`

Tạo file `res/layout/activity_add_word.xml`:

```

xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextWord"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nhập từ mới" />

    <EditText
        android:id="@+id/editTextMeaning"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nhập nghĩa của từ" />

```

```

        <Button
            android:id="@+id/buttonSave"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Lưu từ mới" />
    </LinearLayout>

```

Bước 13: Viết Code cho AddWordActivity.java

Tạo file AddWordActivity.java:

```

java

package com.example.wordlistapp;

import android.content.ContentValues;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.example.wordlistapp.provider.WordContentProvider;

public class AddWordActivity extends AppCompatActivity {
    private EditText editTextWord, editTextMeaning;
    private Button buttonSave;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add_word);

        editTextWord = findViewById(R.id.editTextWord);
        editTextMeaning = findViewById(R.id.editTextMeaning);
        buttonSave = findViewById(R.id.buttonSave);

        buttonSave.setOnClickListener(v -> saveWord());
    }

    private void saveWord() {
        String word = editTextWord.getText().toString().trim();
        String meaning = editTextMeaning.getText().toString().trim();

        if (word.isEmpty() || meaning.isEmpty()) {
            Toast.makeText(this, "Vui lòng nhập đầy đủ thông tin",
                Toast.LENGTH_SHORT).show();
            return;
        }

        ContentValues values = new ContentValues();
        values.put("word", word);
        values.put("meaning", meaning);

        Uri uri =
        getContentResolver().insert(WordContentProvider.CONTENT_URI, values);


        if (uri != null) {
            Toast.makeText(this, "Đã thêm từ thành công!",
                Toast.LENGTH_SHORT).show();

```



```
        setResult (RESULT_OK);  
        finish();  
    } else {  
        Toast.makeText (this, "Thêm từ thất bại!",  
        Toast.LENGTH_SHORT).show();  
    }  
}  
}
```

Kết quả

 Ứng dụng hoàn chỉnh, hỗ trợ:

- **Thêm từ mới** vào danh sách.
- **Hiển thị danh sách từ** bằng RecyclerView.
- **Lưu trữ từ** vào SQLite thông qua Room Database.
- **Truy vấn từ dữ liệu** bằng Content Provider.