Android là gì

Android là một hệ điều hành mã nguồn mở dựa trên nhân Linux, được phát triển bởi Google chủ yếu dành cho các thiết bị di động như điện thoại thông minh, máy tính bảng, đồng hồ thông minh, TV và thậm chí cả ô tô.

**🔹 Đặc điểm chính của Android**

✔ **Mã nguồn mở**: Được phát triển trên nền tảng Linux, cho phép các nhà sản xuất và lập trình viên tùy chỉnh.  
✔ **Được hỗ trợ bởi Google**: Liên tục được cập nhật và cải tiến qua các phiên bản mới.  
✔ **Hệ sinh thái rộng lớn**: Hỗ trợ hàng triệu ứng dụng trên Google Play Store.  
✔ **Đa thiết bị**: Có thể chạy trên nhiều loại phần cứng khác nhau từ điện thoại đến TV, ô tô (Android Auto) và thiết bị đeo tay (Wear OS).  
✔ **Ngôn ngữ lập trình phổ biến**: Hỗ trợ Java, Kotlin (chính thức), C++, Python và nhiều ngôn ngữ khác.

**🔹 Các thành phần chính của Android**

📌 **Kernel (Nhân Linux)**: Cung cấp các dịch vụ hệ thống cơ bản như quản lý bộ nhớ, tiến trình và thiết bị.  
📌 **Android Runtime (ART)**: Thay thế Dalvik VM trước đây, giúp tăng hiệu suất ứng dụng.  
📌 **Framework**: Cung cấp API giúp lập trình viên dễ dàng phát triển ứng dụng.  
📌 **Ứng dụng**: Hệ điều hành hỗ trợ các ứng dụng của bên thứ ba chạy trên Android.

**🔹 Một số phiên bản Android quan trọng**

📌 Android 4.4 (KitKat) - Cải tiến hiệu suất trên thiết bị cấu hình thấp.  
📌 Android 5.0 (Lollipop) - Giới thiệu Material Design.  
📌 Android 6.0 (Marshmallow) - Hỗ trợ quyền truy cập ứng dụng.  
📌 Android 9.0 (Pie) - AI tối ưu hóa pin.  
📌 Android 10 - Chế độ tối Dark Mode.  
📌 Android 12 - Thiết kế Material You.  
📌 Android 14 (mới nhất) - Cải tiến bảo mật và hiệu năng.

Bạn đang quan tâm đến khía cạnh nào của Android? Lập trình ứng dụng, hệ điều hành hay phần cứng?

Cấu trúc các thư mục trong project Android Studio

📂 MyAndroidProject/

├── 📂 .gradle/ # Thư mục hệ thống của Gradle

├── 📂 .idea/ # Cấu hình Android Studio

├── 📂 app/ # Thư mục chính của ứng dụng

│ ├── 📂 manifests/ # Chứa AndroidManifest.xml

│ ├── 📂 java/ # Chứa mã nguồn Java/Kotlin

│ ├── 📂 res/ # Chứa tài nguyên (layout, hình ảnh, v.v.)

│ └── 📜 build.gradle # Cấu hình build cho module app

├── 📂 gradle/ # Thư mục Gradle Wrapper

├── 📜 build.gradle # Cấu hình Gradle toàn bộ project

├── 📜 settings.gradle # Cấu hình module trong project

└── 📜 gradle.properties # Cấu hình Gradle bổ sung

AndroidManifest là gì

**AndroidManifest.xml** là một tệp quan trọng trong mọi ứng dụng Android, giúp hệ điều hành hiểu được:  
✅ Tên ứng dụng, biểu tượng, chủ đề  
✅ Các **Activity, Service, BroadcastReceiver**  
✅ Các quyền yêu cầu (ví dụ: Internet, Camera)  
✅ Cấu hình màn hình chính (MainActivity)

## 🎯 **Tóm tắt**

📌 **AndroidManifest.xml giúp Android hiểu cấu trúc ứng dụng**  
📌 **Chứa thông tin về Activity, Service, quyền truy cập, v.v.**  
📌 **Mọi ứng dụng Android đều phải có AndroidManifest.xml**

Gradle là gì

**Gradle** là một công cụ **tự động hóa build** (build automation) được sử dụng trong **Android Studio** để:  
✅ **Quản lý dependencies** (các thư viện)  
✅ **Biên dịch mã nguồn** Kotlin/Java/XML  
✅ **Tạo tệp APK/AAB** để cài đặt lên thiết bị Android  
✅ **Tự động hóa quá trình build** (kiểm tra, đóng gói, ký ứng dụng)

## 🎯 **Tóm lại**

✅ **Gradle giúp quản lý build và dependencies trong Android Studio**  
✅ **Có 2 tệp build.gradle.kts: một cho toàn project, một cho từng module**  
✅ **Gradle tự động tải thư viện, biên dịch & đóng gói ứng dụng Android**

Đơn vị đo trong Android: dp và sp

## 📏 **Đơn vị đo trong Android:** dp **và** sp

Trong Android, để thiết kế giao diện **đồng nhất trên mọi thiết bị**, chúng ta không dùng **px** (pixel) mà sử dụng **dp (density-independent pixels) và sp (scale-independent pixels)**.

## 1️⃣ dp **(Density-independent Pixels) - Đơn vị độc lập với mật độ màn hình**

📌 **Dùng để đo kích thước layout (width, height, margin, padding,...)**  
📌 **Giúp giao diện hiển thị đồng nhất trên các màn hình có mật độ DPI khác nhau**

### 🔹 ****Công thức chuyển đổi**** dp ****sang**** px

px = dp \* (dpi / 160)

## 2️⃣ sp **(Scale-independent Pixels) - Đơn vị độc lập với tỷ lệ phóng to văn bản**

📌 **Dùng để đo kích thước chữ (textSize)**  
📌 **Tự động thay đổi kích thước theo cài đặt phóng to chữ của người dùng**

### 🔹 ****Công thức chuyển đổi**** sp ****sang**** px

px = sp \* (dpi / 160) \* scaleFactor  
(**scaleFactor** thay đổi theo cài đặt kích thước font của người dùng)

Khai báo các giá trị Resource: dimen, color, font-family, image, drawable,..

## 1️⃣ dimen.xml **- Khai báo kích thước (**dp**,** sp**)**

📌 **Dùng để khai báo giá trị dp cho layout và sp cho chữ**  
📍 **Vị trí:** res/values/dimens.xml

## 2️⃣ colors.xml **- Khai báo màu sắc**

📌 **Dùng để định nghĩa màu sắc theo mã HEX hoặc dạng rgb, rgba**  
📍 **Vị trí:** res/values/colors.xml

## 3️⃣ font **- Khai báo font chữ (**font-family**)**

📌 **Dùng để khai báo font chữ trong thư mục res/font/**  
📍 **Vị trí:** res/font/roboto.xml

## 4️⃣ image **- Khai báo hình ảnh**

📌 **Tất cả ảnh đặt trong thư mục res/drawable/ hoặc res/mipmap/**  
📍 **Vị trí:** res/drawable/logo.png hoặc res/mipmap/ic\_launcher.png

## 5️⃣ drawable **- Khai báo background, shape, vector**

📌 **Dùng để tạo hình ảnh, nền, hoặc hiệu ứng động**  
📍 **Vị trí:** res/drawable/bg\_button.xml

## 🎯 **Tóm lại**

| **Loại Resource** | **Vị trí** | **Chức năng** |
| --- | --- | --- |
| **dimen.xml** | res/values/dimens.xml | Định nghĩa kích thước (dp, sp) |
| **colors.xml** | res/values/colors.xml | Định nghĩa màu sắc |
| **font-family** | res/font/ | Quản lý font chữ |
| **image** | res/drawable/ hoặc res/mipmap/ | Lưu trữ hình ảnh PNG, JPG, SVG |
| **drawable.xml** | res/drawable/ | Định nghĩa hình dạng (shape), hiệu ứng (gradient) |

Activity trong Android là gì

**Activity** là một **thành phần (component) chính trong Android**, đại diện cho **một màn hình giao diện người dùng**. Nó là nơi người dùng có thể tương tác với ứng dụng.

### 📌 ****Đặc điểm của Activity****

✅ Mỗi ứng dụng có thể có nhiều Activity.  
✅ Một Activity thường chứa **giao diện (UI)** gồm các View như Button, TextView, RecyclerView,...  
✅ Một ứng dụng có **một Activity chính** (MainActivity) được khởi chạy đầu tiên khi ứng dụng mở.  
✅ Các Activity có thể liên kết với nhau bằng **Intent**.

## 🔄 **Vòng đời của Activity**

Activity trong Android có vòng đời được quản lý bởi hệ thống, gồm các trạng thái sau:

### 🏷 ****1. Trạng thái và phương thức vòng đời chính****

| **Trạng thái** | **Phương thức** | **Mô tả** |
| --- | --- | --- |
| **Khởi tạo** | onCreate() | Được gọi khi Activity được tạo. |
| **Hiển thị** | onStart() | Activity xuất hiện trên màn hình. |
| **Đang chạy** | onResume() | Người dùng có thể tương tác với Activity. |
| **Bị dừng tạm thời** | onPause() | Activity bị che bởi một Activity khác. |
| **Bị ẩn hoàn toàn** | onStop() | Activity không còn hiển thị nhưng chưa bị hủy. |
| **Bị hủy** | onDestroy() | Activity bị hủy để giải phóng bộ nhớ. |

📍 **Sơ đồ vòng đời Activity**  
📌 **Trình tự thực thi:**

scss

Sao chépChỉnh sửa

onCreate() → onStart() → onResume() → (Hoạt động) → onPause() → onStop() → onDestroy()

💡 **Lưu ý:**

* Nếu Activity bị che bởi một Activity khác → gọi onPause()
* Nếu người dùng quay lại Activity → gọi onResume()
* Nếu Activity bị hủy hoàn toàn → gọi onDestroy()

🔄 **Chuyển đổi giữa các Activity bằng Intent**

Các view cơ bản trong Android

+ ViewGroup: LinearLayout, RelativeLayout, ConstraintLayout, ...

+ View: TextView, EditText, ImageView, View, ...

**🏗 1. ViewGroup – Nhóm chứa các View**

📌 **ViewGroup** là các thành phần dùng để **chứa và sắp xếp** các **View con** bên trong nó.  
📌 Một số **ViewGroup phổ biến**:

| **Loại ViewGroup** | **Mô tả** |
| --- | --- |
| LinearLayout | Sắp xếp các View theo hàng ngang (horizontal) hoặc dọc (vertical). |
| RelativeLayout | Sắp xếp các View dựa trên vị trí của các View khác. |
| ConstraintLayout | Cho phép thiết kế linh hoạt, tối ưu hiệu suất. |
| FrameLayout | Chỉ chứa một View, phù hợp cho giao diện đơn giản. |
| ScrollView | Cho phép cuộn nội dung nếu vượt quá kích thước màn hình. |
| RecyclerView | Dùng để hiển thị danh sách dữ liệu lớn với hiệu suất cao. |

**🖼 2. View – Các Thành Phần Cơ Bản của UI**

📌 **View** là thành phần hiển thị trực tiếp trên giao diện.  
📌 Một số **View phổ biến**:

| **Loại View** | **Mô tả** |
| --- | --- |
| TextView | Hiển thị văn bản. |
| EditText | Ô nhập dữ liệu văn bản. |
| Button | Nút bấm. |
| ImageView | Hiển thị hình ảnh. |
| CheckBox | Hộp kiểm tra (check/uncheck). |
| RadioButton | Nút chọn một trong nhiều lựa chọn. |
| Switch | Công tắc bật/tắt. |
| SeekBar | Thanh kéo giá trị. |
| ProgressBar | Hiển thị tiến trình. |

Activity, vòng đời của Activity

Activity là một thành phần quan trọng trong Android, đại diện cho một màn hình giao diện người dùng. Mỗi Activity thường chứa UI và tương tác với người dùng.

### 🔄 ****Vòng đời của Activity trong Android****

Vòng đời của một Activity trong Android bao gồm các trạng thái sau:

1. **onCreate()** 🔹
   * Được gọi khi Activity được tạo lần đầu tiên.
   * Dùng để khởi tạo giao diện, khai báo biến, tải dữ liệu, gán sự kiện.
2. **onStart()** 🔹
   * Activity sắp hiển thị lên màn hình nhưng chưa có focus.
   * UI đã được vẽ nhưng người dùng chưa tương tác được.
3. **onResume()** 🔹
   * Activity đã ở trạng thái hoạt động và có thể tương tác với người dùng.
4. **onPause()** 🔹
   * Được gọi khi Activity bị gián đoạn (chẳng hạn khi có cuộc gọi đến).
   * Không nên thực hiện tác vụ nặng vì có thể quay lại nhanh.
5. **onStop()** 🔹
   * Activity không còn hiển thị trên màn hình (chuyển sang Activity khác).
   * Thích hợp để lưu trạng thái hoặc giải phóng tài nguyên.
6. **onDestroy()** 🔹
   * Activity bị hủy (do người dùng thoát hoặc hệ thống cần giải phóng bộ nhớ).
   * Dọn dẹp tài nguyên trước khi Activity bị xóa.
7. **onRestart()** 🔹
   * Activity quay lại từ trạng thái onStop() (chẳng hạn khi người dùng nhấn Back).
   * Dùng để cập nhật dữ liệu nếu cần.

### 📊 ****Sơ đồ vòng đời của Activity****

📌 **Sơ đồ mô tả vòng đời của Activity theo thứ tự:**  
🔹 onCreate() → onStart() → onResume() → **(Activity đang chạy)**  
🔹 onPause() → onStop() → onDestroy() (thoát hẳn)  
🔹 onRestart() (nếu quay lại sau khi onStop())

Fragment, vòng đời của Fragment

Fragment là một phần của Activity, giúp chia nhỏ giao diện để tái sử dụng. Một Activity có thể chứa nhiều Fragment, và một Fragment có thể được sử dụng trong nhiều Activity.

### 🔄 ****Vòng đời của Fragment trong Android****

Vòng đời của Fragment có phần giống Activity, nhưng được quản lý bởi Activity chứa nó.

#### **📌 Các trạng thái quan trọng của Fragment**

1. **onAttach()** 🔹
   * Fragment được gắn vào Activity.
   * Dùng để lấy tham chiếu Activity chứa nó.
2. **onCreate()** 🔹
   * Khởi tạo Fragment, nhưng UI chưa được tạo.
   * Dùng để khởi tạo biến, lấy dữ liệu từ Bundle.
3. **onCreateView()** 🔹
   * Dùng để tạo và trả về View của Fragment.
   * Dùng LayoutInflater để tạo UI từ file XML.
4. **onViewCreated()** 🔹
   * Được gọi sau khi onCreateView().
   * Dùng để thiết lập dữ liệu cho các thành phần UI.
5. **onStart()** 🔹
   * Fragment bắt đầu hiển thị lên màn hình.
6. **onResume()** 🔹
   * Fragment hoạt động, có thể tương tác với người dùng.
7. **onPause()** 🔹
   * Fragment bị gián đoạn (do Activity chứa nó bị tạm dừng hoặc một Fragment khác chồng lên).
8. **onStop()** 🔹
   * Fragment không còn hiển thị.
9. **onDestroyView()** 🔹
   * Hủy View của Fragment, nhưng không hủy Fragment.
10. **onDestroy()** 🔹
    * Hủy Fragment, giải phóng tài nguyên.
11. **onDetach()** 🔹
    * Fragment bị tách khỏi Activity.

### 📊 ****Sơ đồ vòng đời của Fragment****

📌 **Trình tự gọi các phương thức phổ biến:**  
🔹 onAttach() → onCreate() → onCreateView() → onViewCreated() → onStart() → onResume() → **(Fragment đang chạy)**  
🔹 onPause() → onStop() → onDestroyView() → onDestroy() → onDetach()

Bottom Navigation Bar

Bottom Navigation Bar là một thành phần UI ở cuối màn hình, giúp người dùng chuyển đổi giữa các màn hình chính trong ứng dụng.

## 🛠 **1. Cài đặt Bottom Navigation Bar trong Android (Kotlin)**

### ****Bước 1: Thêm dependencies (nếu chưa có)****

Mở build.gradle (Module: app) và thêm:

gradle

Sao chépChỉnh sửa

dependencies {

implementation 'com.google.android.material:material:1.10.0'

}

### ****Bước 2: Thêm**** BottomNavigationView ****vào**** activity\_main.xml

xml

Sao chépChỉnh sửa

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<FrameLayout

android:id="@+id/frameLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:layout\_above="@id/bottomNavigationView"/>

<com.google.android.material.bottomnavigation.BottomNavigationView

android:id="@+id/bottomNavigationView"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_alignParentBottom="true"

app:menu="@menu/bottom\_nav\_menu"/>

</RelativeLayout>

### ****Bước 3: Tạo menu cho Bottom Navigation (****res/menu/bottom\_nav\_menu.xml****)****

xml

Sao chépChỉnh sửa

<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item

android:id="@+id/nav\_home"

android:icon="@drawable/ic\_home"

android:title="Home"/>

<item

android:id="@+id/nav\_search"

android:icon="@drawable/ic\_search"

android:title="Search"/>

<item

android:id="@+id/nav\_profile"

android:icon="@drawable/ic\_profile"

android:title="Profile"/>

</menu>

### ****Bước 4: Xử lý chọn item trong**** MainActivity.kt

kotlin

Sao chépChỉnh sửa

class MainActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

val bottomNavigationView = findViewById<BottomNavigationView>(R.id.bottomNavigationView)

bottomNavigationView.setOnNavigationItemSelectedListener { item ->

when (item.itemId) {

R.id.nav\_home -> {

replaceFragment(HomeFragment())

true

}

R.id.nav\_search -> {

replaceFragment(SearchFragment())

true

}

R.id.nav\_profile -> {

replaceFragment(ProfileFragment())

true

}

else -> false

}

}

// Chọn mặc định tab Home

bottomNavigationView.selectedItemId = R.id.nav\_home

}

private fun replaceFragment(fragment: Fragment) {

supportFragmentManager.beginTransaction()

.replace(R.id.frameLayout, fragment)

.commit()

}

}

### ****Bước 5: Tạo các Fragment (Home, Search, Profile)****

Ví dụ HomeFragment.kt:

kotlin

Sao chépChỉnh sửa

class HomeFragment : Fragment() {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_home, container, false)

}

}

Tương tự tạo SearchFragment.kt và ProfileFragment.kt.

## 🎨 **Tùy chỉnh giao diện**

### 🌟 ****1. Đổi màu nền và màu icon khi chọn****

Thêm vào res/values/colors.xml:

xml

Sao chépChỉnh sửa

<color name="nav\_selected">#FF5722</color>

<color name="nav\_unselected">#757575</color>

Thêm vào res/values/styles.xml:

xml

Sao chépChỉnh sửa

<style name="BottomNavStyle" parent="Widget.MaterialComponents.BottomNavigationView">

<item name="itemIconTint">@color/nav\_selected</item>

<item name="itemTextColor">@color/nav\_selected</item>

<item name="backgroundTint">@color/white</item>

</style>

Áp dụng trong activity\_main.xml:

xml

Sao chépChỉnh sửa

app:itemIconTint="@color/nav\_selected"

app:itemTextColor="@color/nav\_selected"

- Mô hình MVVM:

+ Data Binding

+ ViewModel

MVVM (Model-View-ViewModel) là một kiến trúc phổ biến trong Android, giúp tách biệt logic xử lý (ViewModel) với giao diện (View), giúp dễ bảo trì và kiểm thử.

## 🛠 **1. Thành phần chính trong MVVM**

### ****📌 1.1. Model (Dữ liệu & logic)****

* Chứa dữ liệu của ứng dụng (có thể lấy từ API, Database).
* Không biết gì về UI.
* Ví dụ: Repository xử lý dữ liệu từ Room, Retrofit.

### ****📌 1.2. ViewModel (Xử lý logic & dữ liệu)****

* Chứa logic nghiệp vụ, cung cấp dữ liệu cho View.
* Dữ liệu được lưu trữ ngay cả khi màn hình bị xoay.
* Sử dụng LiveData để cập nhật UI một cách tự động.

### ****📌 1.3. View (Giao diện người dùng)****

* Lắng nghe dữ liệu từ ViewModel và hiển thị lên UI.
* Không chứa logic nghiệp vụ.

## 🎯 **2. Cài đặt MVVM trong Android với ViewModel & Data Binding**

### ****📌 Bước 1: Thêm dependencies vào**** build.gradle

gradle

Sao chépChỉnh sửa

dependencies {

implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2"

implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.6.2"

implementation "androidx.databinding:databinding-runtime:7.0.0"

}

### ****📌 Bước 2: Tạo Model (****User.kt****)****

kotlin

Sao chépChỉnh sửa

data class User(val name: String, val age: Int)

### ****📌 Bước 3: Tạo ViewModel (****UserViewModel.kt****)****

kotlin

Sao chépChỉnh sửa

class UserViewModel : ViewModel() {

private val \_user = MutableLiveData<User>()

val user: LiveData<User> get() = \_user

fun updateUser(name: String, age: Int) {

\_user.value = User(name, age)

}

}

### ****📌 Bước 4: Tạo layout**** activity\_main.xml ****và sử dụng Data Binding****

xml

Sao chépChỉnh sửa

<?xml version="1.0" encoding="utf-8"?>

<layout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools">

<data>

<variable

name="viewModel"

type="com.example.app.UserViewModel" />

</data>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:padding="16dp"

tools:context=".MainActivity">

<TextView

android:id="@+id/tvUserName"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@{viewModel.user.name}" />

<EditText

android:id="@+id/etName"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:hint="Enter name" />

<EditText

android:id="@+id/etAge"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:hint="Enter age"

android:inputType="number" />

<Button

android:text="Update User"

android:onClick="@{() -> viewModel.updateUser(etName.text.toString(), Integer.parseInt(etAge.text.toString()))}"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

</LinearLayout>

</layout>

### ****📌 Bước 5: Cấu hình**** MainActivity.kt ****để sử dụng ViewModel & Data Binding****

kotlin

Sao chépChỉnh sửa

class MainActivity : AppCompatActivity() {

private lateinit var binding: ActivityMainBinding

private lateinit var viewModel: UserViewModel

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

binding = DataBindingUtil.setContentView(this, R.layout.activity\_main)

viewModel = ViewModelProvider(this).get(UserViewModel::class.java)

binding.viewModel = viewModel

binding.lifecycleOwner = this

}

}

LiveData

LiveData là một **thành phần của Jetpack** giúp **quan sát dữ liệu** trong ứng dụng. Nó tự động cập nhật UI khi dữ liệu thay đổi và giúp **tránh rò rỉ bộ nhớ** (memory leaks).

## 🚀 **1. Đặc điểm chính của LiveData**

✅ **Tuân theo vòng đời (Lifecycle-aware)**: Chỉ cập nhật dữ liệu khi Activity hoặc Fragment đang ở trạng thái **active**.  
✅ **Tự động cập nhật UI**: Khi dữ liệu thay đổi, tất cả các Observer (người quan sát) sẽ tự động nhận giá trị mới.  
✅ **Không gây rò rỉ bộ nhớ**: Khi Activity bị hủy, LiveData tự động **ngừng quan sát**, không cần xóa thủ công.  
✅ **Dễ dàng kết hợp với ViewModel** để **lưu dữ liệu ngay cả khi xoay màn hình**.

## 🎯 **2. Cách sử dụng LiveData**

### ****👉 Bước 1: Tạo ViewModel chứa LiveData****

kotlin

Sao chépChỉnh sửa

class MainViewModel : ViewModel() {

val count = MutableLiveData(0) // MutableLiveData: Có thể thay đổi dữ liệu

fun increaseCount() {

count.value = (count.value ?: 0) + 1

}

}

### ****👉 Bước 2: Sử dụng LiveData trong Activity****

kotlin

Sao chépChỉnh sửa

class MainActivity : AppCompatActivity() {

private lateinit var viewModel: MainViewModel

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

val tvCount = findViewById<TextView>(R.id.tvCount)

val btnIncrease = findViewById<Button>(R.id.btnIncrease)

// Khởi tạo ViewModel

viewModel = ViewModelProvider(this).get(MainViewModel::class.java)

// Quan sát LiveData -> Tự động cập nhật UI khi dữ liệu thay đổi

viewModel.count.observe(this) { count ->

tvCount.text = count.toString()

}

// Nút tăng số

btnIncrease.setOnClickListener {

viewModel.increaseCount()

}

}

}

## 🎯 **3. Các loại LiveData**

### ****✅ MutableLiveData****

* Cho phép **thay đổi giá trị** bên trong ViewModel.
* Dùng .value hoặc .postValue() để cập nhật dữ liệu.

kotlin

Sao chépChỉnh sửa

val count = MutableLiveData(0)

// Cập nhật giá trị

count.value = 10 // Cập nhật ngay

count.postValue(20) // Cập nhật bất đồng bộ

### ****✅ LiveData (chỉ đọc)****

* **Không thể thay đổi giá trị** bên ngoài ViewModel.
* Dùng khi bạn muốn dữ liệu **chỉ có thể được thay đổi bên trong ViewModel**.

kotlin

Sao chépChỉnh sửa

private val \_count = MutableLiveData(0) // Biến nội bộ (có thể thay đổi)

val count: LiveData<Int> = \_count // Chỉ đọc từ bên ngoài

fun increaseCount() {

\_count.value = (\_count.value ?: 0) + 1

}

## 🎯 **4. Khi nào nên dùng LiveData?**

✅ Khi cần **quan sát dữ liệu từ ViewModel** và tự động cập nhật UI.  
✅ Khi cần **lưu dữ liệu ngay cả khi xoay màn hình**.  
✅ Khi muốn dữ liệu **tuân theo vòng đời của Activity/Fragment** để tránh rò rỉ bộ nhớ.

## 🔥 **Tóm tắt**

| **Tính năng** | **MutableLiveData** | **LiveData** |
| --- | --- | --- |
| **Có thể thay đổi giá trị** | ✅ Có | ❌ Không |
| **Tự động cập nhật UI** | ✅ Có | ✅ Có |
| **Tuân theo vòng đời** | ✅ Có | ✅ Có |
| **Dùng trong ViewModel** | ✅ Có | ✅ Có |

Cơ sở dữ liệu trong thiết bị:

+ Sqlite, Room

+ SharedPreferences

Android cung cấp nhiều cách để lưu trữ dữ liệu trong thiết bị, trong đó phổ biến nhất là:  
✅ **SQLite** & **Room Database** (Cơ sở dữ liệu quan hệ)  
✅ **SharedPreferences** (Lưu trữ dữ liệu dạng key-value)

## **🚀 1. SQLite & Room Database**

### ****🔹 SQLite là gì?****

* SQLite là hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) **tích hợp sẵn** trong Android.
* Sử dụng cú pháp **SQL chuẩn** để thao tác dữ liệu.
* Cần viết nhiều **code thủ công** (mở kết nối, truy vấn, đóng kết nối).

#### **👎 Nhược điểm của SQLite**

❌ Code dài, dễ lỗi.  
❌ Không hỗ trợ kiểm tra lỗi compile-time.  
❌ Không hỗ trợ xử lý bất đồng bộ mặc định.

### ****🔹 Room Database (Thư viện ORM của Jetpack)****

Room là một **ORM (Object-Relational Mapping)** giúp đơn giản hóa việc thao tác với SQLite trong Android.

✅ **Tự động chuyển đổi dữ liệu từ Object thành SQL**.  
✅ **Kiểm tra lỗi SQL ngay tại compile-time**.  
✅ **Hỗ trợ LiveData và Flow** để xử lý bất đồng bộ.

### ****👉 Cách sử dụng Room****

#### **Bước 1: Thêm dependencies vào** build.gradle

gradle

Sao chépChỉnh sửa

dependencies {

implementation "androidx.room:room-runtime:2.6.1"

kapt "androidx.room:room-compiler:2.6.1"

}

#### **Bước 2: Tạo Entity (Bảng trong SQLite)**

kotlin

Sao chépChỉnh sửa

@Entity(tableName = "user")

data class User(

@PrimaryKey(autoGenerate = true) val id: Int = 0,

val name: String,

val age: Int

)

#### **Bước 3: Tạo DAO (Data Access Object)**

kotlin

Sao chépChỉnh sửa

@Dao

interface UserDao {

@Insert

suspend fun insertUser(user: User)

@Query("SELECT \* FROM user")

fun getAllUsers(): LiveData<List<User>>

}

#### **Bước 4: Tạo Database**

kotlin

Sao chépChỉnh sửa

@Database(entities = [User::class], version = 1)

abstract class AppDatabase : RoomDatabase() {

abstract fun userDao(): UserDao

}

#### **Bước 5: Khởi tạo Database trong ViewModel**

kotlin

Sao chépChỉnh sửa

val db = Room.databaseBuilder(

applicationContext,

AppDatabase::class.java, "app\_database"

).build()

val userDao = db.userDao()

### ****🚀 Khi nào nên dùng Room Database?****

✅ Khi cần lưu trữ dữ liệu có **mối quan hệ**.  
✅ Khi cần **tìm kiếm, lọc, truy vấn dữ liệu** nhanh chóng.  
✅ Khi dữ liệu có **dung lượng lớn**.

## **🚀 2. SharedPreferences**

### ****🔹 SharedPreferences là gì?****

* SharedPreferences giúp lưu trữ **dữ liệu nhỏ** dưới dạng **key-value**.
* Phù hợp để lưu **cấu hình, cài đặt, trạng thái đăng nhập**.
* Không phù hợp để lưu **dữ liệu lớn** hoặc **dữ liệu quan hệ**.

### ****👉 Cách sử dụng SharedPreferences****

#### **Bước 1: Lưu dữ liệu**

kotlin

Sao chépChỉnh sửa

val sharedPreferences = getSharedPreferences("my\_prefs", Context.MODE\_PRIVATE)

val editor = sharedPreferences.edit()

editor.putString("username", "NguyenVanA")

editor.putInt("age", 25)

editor.apply()

#### **Bước 2: Đọc dữ liệu**

kotlin

Sao chépChỉnh sửa

val sharedPreferences = getSharedPreferences("my\_prefs", Context.MODE\_PRIVATE)

val username = sharedPreferences.getString("username", "Guest")

val age = sharedPreferences.getInt("age", 0)

### ****🚀 Khi nào nên dùng SharedPreferences?****

✅ Khi cần lưu trữ **cấu hình ứng dụng** (ví dụ: chế độ tối/sáng).  
✅ Khi lưu **token đăng nhập** hoặc **trạng thái đăng nhập**.  
✅ Khi cần **truy xuất dữ liệu nhanh** mà không cần truy vấn phức tạp.

## **🔥 So sánh Room Database vs SharedPreferences**

| **Tiêu chí** | **Room Database** | **SharedPreferences** |
| --- | --- | --- |
| **Kiểu dữ liệu** | Dữ liệu có quan hệ (bảng) | Key-Value đơn giản |
| **Dung lượng** | Lưu trữ dữ liệu lớn | Chỉ lưu dữ liệu nhỏ |
| **Tốc độ** | Tìm kiếm nhanh nhưng phức tạp | Truy xuất cực nhanh |
| **Ứng dụng** | Dữ liệu phức tạp (người dùng, sản phẩm) | Cài đặt, token đăng nhập |