1. ViewModel (Khái niệm – Vì sao cần – Ví dụ)

1.1 Khái niệm

`ViewModel` là class thuộc Jetpack Architecture Components, được thiết kế để:

- Lưu giữ dữ liệu UI
- Sống lâu hơn Activity/Fragment
- Không bị mất khi xoay màn hình (configuration changes)
- Tránh recreate và tránh memory leak

Nó gắn với lifecycle của Activity/Fragment chứ không gắn với lifecycle UI.

1.2 Tại sao cần ViewModel?

Không dùng ViewModel → xoay màn hình (rotate) → Activity bị destroy → dữ liệu mất.

→ ViewModel giải quyết vấn đề này bằng cách sống lâu hơn UI.

1.3 Ví dụ ViewModel đơn giản

ViewModel

```
class MainViewModel : ViewModel() {
    val counter = MutableLiveData<Int>(0)

    fun increase() {
        counter.value = counter.value?.plus(1)
    }
}
```

Activity dùng ViewModel

```
class MainActivity : AppCompatActivity() {

    private val vm: MainViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        vm.counter.observe(this) { value ->
            findViewById<TextView>(R.id.tvCounter).text = value.toString()
        }
```

```
        findViewById<Button>(R.id.btnAdd).setOnClickListener {
            vm.increase()
        }
    }
}
```

1.4 Trả lời phỏng vấn ViewModel chuẩn

> "ViewModel là nơi lưu trữ và quản lý dữ liệu UI. Nó sống lâu hơn Activity/Fragment, do đó tránh mất dữ liệu khi xoay màn hình và giảm logic trong UI class. ViewModel kết hợp với LiveData/Flow để update UI theo lifecycle-safe."

---

2. LiveData vs StateFlow (Khác nhau – Khi dùng cái nào)

2.1 LiveData (Jetpack)

Đặc điểm:

- Là lifecycle-aware → chỉ update UI khi Activity/Fragment đang active
- Dễ dùng
- Thường dùng với ViewModel cổ điển

2.2 StateFlow (Kotlin Flow)

Đặc điểm:

- Modern, thuộc Kotlin coroutine
- Không lifecycle-aware → cần collect trong lifecycleScope
- StateFlow always has a **current value** (giống LiveData)
- Tối ưu cho reactive programming

2.3 Bảng so sánh

Thuộc tính	LiveData	StateFlow
Lifecycle aware	✔ Có	❌ Không
Multi-thread (suspend)	❌ Không	✔ Có
Giá trị mặc định	✔ Có	✔ Có
Best for UI MVVM	✔	✔ (hiện đại hơn)

Thuộc tính	LiveData	StateFlow
Backpressure	❌ Không	✔ Có
Nullable	✔ Cho phép	❌ Không cho phép null mặc định

2.4 Khi nào dùng cái nào?

- **Dự án cũ** → LiveData
- **Dự án hiện đại (Kotlin)** → StateFlow
- **Đòi hỏi streaming / reactive** → Flow/StateFlow
- **UI updates đơn giản** → LiveData

Code: StateFlow + collect

```
class MainViewModel : ViewModel() {
    private val _count = MutableStateFlow(0)
    val count = _count.asStateFlow()

    fun increase() {
        _count.value++
    }
}
```

Collect trong Activity:

```
lifecycleScope.launchWhenStarted {
    vm.count.collect {
        tvCounter.text = it.toString()
    }
}
```

---

3. Room Database (Entity – DAO – Database)

Room gồm 3 thành phần chính:

- **Entity** → bảng trong database
- **DAO** → truy vấn dữ liệu
- **Database** → quản lý database + tạo instance

3.1 Entity (tạo bảng)

```
@Entity(tableName = "users")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String,
    val age: Int
)
```

3.2 DAO (CRUD)

```
@Dao
interface UserDao {

    @Insert
    suspend fun insert(user: User)

    @Query("SELECT * FROM users")
    fun getAll(): Flow<List<User>>

    @Delete
    suspend fun delete(user: User)
}
```

3.3 Database

```
@Database(entities = [User::class], version = 1)
abstract class AppDB : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

3.4 Tạo instance Room

```
val db = Room.databaseBuilder(
    context,
    AppDB::class.java,
    "app.db"
).build()

db.userDao().insert(User(name = "Minh", age = 20))
```

3.5 Room Database

> "Room là một ORM của Jetpack giúp tương tác SQLite dễ dàng, type-safe và tránh lỗi runtime SQL. Gồm 3 component: Entity (bảng), DAO (query), Database (quản lý DB). Room kết hợp rất tốt với Flow/LiveData để auto-update UI."

4. WorkManager vs Coroutine vs Service (phân biệt cực chi tiết)

4.1 Coroutine scope

Chạy tác vụ **ngắn**, liên quan UI hoặc logic tạm thời. → Bị cancel khi UI bị destroy.

Dùng cho:

- Gọi API
- Load data
- Xử lý background ngắn

4.2 Service

Chạy **ngầm**, sống lâu dù user thoát app.

Dùng cho:

- Music player
- GPS tracking
- Long-running task

4.3 WorkManager

WorkManager là API chạy background **đảm bảo thực thi**, kể cả:

- App bị kill
- Reboot
- Thời gian delay
- Ràng buộc (network, battery)

Dùng cho:

- Upload log
- Sync data
- Backup dữ liệu
- Job định kỳ

4.4 Bảng so sánh

Loại	Khi dùng	Sống lâu	Đảm bảo hoàn thành	Bị OS kill?
Coroutine	Tác vụ ngắn	❌	❌	✔ Dễ bị kill
Service	Tác vụ dài	✔	❌	⚠ Từ Android O bị hạn chế
WorkManager	Tác vụ lâu dài, đảm bảo hoàn thành	✔	✔	❌ Không bị kill

> "Nếu tác vụ chạy ngắn → Coroutine. Nếu chạy dài và cần foreground → Service. Nếu muốn chắc chắn hoàn thành kể cả app bị kill → WorkManager."

---

5. Navigation Component (Navigation Graph – Safe Args – Backstack)

5.1 Khái niệm

Navigation Component là Jetpack library giúp:

- Điều hướng giữa các Fragment
- Quản lý back stack tự động
- Tránh lỗi Fragment transaction
- Dễ truyền dữ liệu giữa màn hình

Đặc biệt phù hợp với **Single Activity Architecture**.

5.2 Thành phần chính

Thành phần	Vai trò
NavHostFragment	Container chứa các fragment
Navigation Graph	File XML mô tả luồng điều hướng
NavController	Thực hiện điều hướng

Thành phần	Vai trò
Safe Args	Truyền data type-safe giữa Fragment

5.3 Ví dụ Navigation Graph

```
<navigation ...>
    <fragment
        android:id="@+id/homeFragment"
        android:name="com.example.HomeFragment">
        <action
            android:id="@+id/action_home_to_detail"
            app:destination="@id/detailFragment" />
    </fragment>

    <fragment
        android:id="@+id/detailFragment"
        android:name="com.example.DetailFragment" />
</navigation>
```

5.4 Điều hướng bằng Kotlin

```
findNavController().navigate(R.id.action_home_to_detail)
```

5.5 Truyền data bằng Safe Args

HomeFragment:

```
val action = HomeFragmentDirections.actionHomeToDetail(userId = 10)
findNavController().navigate(action)
```

DetailFragment nhận data:

```
val id = arguments?.let { DetailFragmentArgs.fromBundle(it).userId }
```

> “Navigation Component giúp quản lý luồng điều hướng rõ ràng bằng Navigation Graph, tránh lỗi fragment transaction thủ công. Safe Args hỗ trợ truyền dữ liệu type-safe. Đây là lựa chọn chuẩn với Single Activity Architecture.”
