☑ 1. SharedPreferences vs DataStore

1.1 SharedPreferences

📌 Khái niệm

- SharedPreferences là cơ chế lưu trữ **key–value**, dạng **XML** trong bộ nhớ thiết bị.
- Dùng để lưu dữ liệu **nhỏ, đơn giản**, như setting app, token, boolean flags…
- Chỉ phù hợp cho dữ liệu nhỏ (≤ vài KB).

📌 Đặc điểm

SharedPreferences	Ghi chú
Đồng bộ (synchronous)	Ghi file trực tiếp → dễ gây **ANR** nếu file lớn
API lâu đời	Bị khuyến nghị **không dùng** cho logic quan trọng
Không đảm bảo an toàn khi đọc/ghi đa luồng	Có thể ghi đè hoặc mất dữ liệu
Lưu dạng XML trong Internal Storage	Không mã hóa

📌 Cách dùng

Kotlin

```kotlin
val shared = context.getSharedPreferences("USER_PREF", Context.MODE_PRIVATE)

// Write
shared.edit().putString("username", "Quang").apply()

// Read
val name = shared.getString("username", "")
```

Java

```java
SharedPreferences shared = getSharedPreferences("USER_PREF", MODE_PRIVATE);

// Write
shared.edit().putInt("age", 20).apply();

// Read
int age = shared.getInt("age", 0);
```

1.2 DataStore (Jetpack)

DataStore là API mới thay thế SharedPreferences. Có hai loại:

- **Preferences DataStore** → Key-value giống SharedPreferences
- **Proto DataStore** → Dữ liệu có schema, mạnh mẽ, dùng protobuf

📌 Ưu điểm

DataStore	SharedPreferences
Asynchronous (coroutines / Flow)	Synchronous → dễ ANR
An toàn đa luồng	Không an toàn
Không bị mất dữ liệu	Dễ bị lỗi/ghi đè
Tối ưu hơn cho dữ liệu lớn	Không phù hợp

📌 Cách dùng (Preferences DataStore)

1. Khai báo

```
val Context.dataStore: DataStore<Preferences> by preferencesDataStore("settings")
```

2. Ghi dữ liệu

```
suspend fun saveToken(token: String) {
    val TOKEN_KEY = stringPreferencesKey("token")

    context.dataStore.edit { prefs ->
        prefs[TOKEN_KEY] = token
    }
}
```

3. Đọc dữ liệu

```
val TOKEN_KEY = stringPreferencesKey("token")

val tokenFlow: Flow<String> = context.dataStore.data
    .map { prefs -> prefs[TOKEN_KEY] ?: "" }
```

☑ 2. **Room Database (ORM)**

2.1 Room là gì?

Room là **ORM (Object Relational Mapping)** của Android giúp làm việc với SQLite dễ dàng hơn. Thay vì viết SQL thủ công, Room tự map dữ liệu → object.

📌 Các thành phần chính

1. **Entity**
    - Đại diện 1 bảng trong database
2. **DAO (Data Access Object)**
    - Chứa các hàm tương tác DB: insert, update, delete, query
3. **RoomDatabase**
    - Tạo database instance

2.2 Ví dụ hoàn chỉnh

➤ Bước 1: Entity

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

➤ Bước 2: DAO

```
@Dao
interface UserDao {

    @Insert
    suspend fun insert(user: User)

    @Query("SELECT * FROM user_table")
    fun getAllUsers(): Flow<List<User>>

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

➤ Bước 3: Database

```
@Database(entities = [User::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
```

```
    abstract fun userDao(): UserDao
}
```

➤ Bước 4: Khởi tạo DB

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

val userDao = db.userDao()
```

➤ Bước 5: Dùng trong ViewModel

```
viewModelScope.launch {
    userDao.insert(User(name = "Quang", age = 22))
}

val userFlow = userDao.getAllUsers()
```

❗ Room so với SQLite truyền thống

Room	SQLite
Compile-time SQL validation	Không kiểm tra lỗi SQL lúc compile
Dễ dùng, code ít	Cần viết SQL thủ công
Tích hợp LiveData/Flow	Không hỗ trợ
Tự động map object	Tự parse Cursor thủ công

☑ 3. Internal Storage vs External Storage

Android có 2 loại bộ nhớ chính để lưu file.

3.1 Internal Storage

📌 Đặc điểm

- Chỉ app truy cập → **private**
- File bị xoá khi app bị uninstall

- An toàn hơn (không cần permission)
- Dùng cho: token, cache, avatar, file nhạy cảm

📌 Ghi file (Kotlin)

```
val filename = "data.txt"
context.openFileOutput(filename, Context.MODE_PRIVATE).use {
    it.write("Hello Android".toByteArray())
}
```

📌 Đọc file

```
val text = context.openFileInput("data.txt")
    .bufferedReader().use { it.readText() }
```

3.2 External Storage

📌 Đặc điểm

- Người dùng có thể xem được bằng File Manager
- Cần quyền:
  - READ_EXTERNAL_STORAGE
  - WRITE_EXTERNAL_STORAGE
- Không an toàn → bị xoá khi format thẻ SD
- Dùng cho file lớn: ảnh, video, file xuất Excel/PDF

📌 Ghi file vào external

```
val file = File(context.getExternalFilesDir(null), "note.txt")
file.writeText("This is external data")
```

📌 Đọc file

```
val file = File(context.getExternalFilesDir(null), "note.txt")
val text = file.readText()
```

🔎 Internal vs External (so sánh)

Nội dung	Internal	External
Quyền	Không cần	Cần request permission
Bảo mật	Cao	Thấp
Người dùng truy cập	Không	Có
Dùng cho	Dữ liệu nhỏ, private	File lớn, media