Android Development Guide

How to Install Android Studio, Set Up, and Clone an Android Project on macOS and Windows

1. Prerequisites

For macOS:

- macOS Mojave (10.14) or later
- At least 8 GB RAM and 20 GB free disk space
- Xcode (optional, but recommended for emulator dependencies)

For Windows:

- Windows 10 or 11 (64-bit)
- At least 16 GB RAM and 20 GB free disk space
- Virtualization enabled in BIOS (for emulator support)

2. Installing Android Studio

For macOS:

- Visit: https://developer.android.com/studio
- Click Download Android Studio and accept the terms.
- Open the downloaded .dmg file.
- Drag the Android Studio icon to the Applications folder.
- Launch Android Studio from Launchpad or Spotlight.
- Grant permissions if prompted and complete the setup wizard.

For Windows:

- Visit: https://developer.android.com/studio
- Download the .exe installer.
- Run the installer and follow the setup steps.
- Leave default options checked (Android SDK, emulator, etc.).
- Finish the setup and launch Android Studio.

3. Initial Configuration

– Select Standard installation during the first-time setup.

– Optionally sign in with your Google account.

– Android Studio will download SDKs and emulator images.

– After setup, you'll see the Android Studio Welcome screen.

4. Cloning an Android Project

Method A:

– Using Android Studio UI

– On the Welcome screen, click "Get from VCS".

– Choose Git as the version control system.

– Paste your repository URL, for example:
https://github.com/your-username/your-android-project.git

– Choose the destination folder on your computer.

– Click Clone.

– Android Studio will download the project and sync Gradle.

Method B:

– Using Git Command Line

– Open Terminal (macOS) or Command Prompt (Windows).

– Run the following command, for example:
git clone https://github.com/your-username/your-android-project.git

– Open Android Studio > Open > Select the cloned project folder.

4. Build and Run the Project:

– Android Studio may automatically sync the Gradle files. If not, go to File > Sync Project with Gradle Files. If prompted, update the Gradle plugin and SDK version.

– Connect a physical Android device or create a virtual device.

– Press the green Run ▶ button or hit Shift + F10 to run the app.

Project Structure Overview

```
com.example.demo/
├── app/                # Core utilities
│   ├── base/           # Base classes (ViewModel, State, etc.)
│   ├── component/      # Reusable UI components (e.g., buttons, cards)
│   ├── constants/      # Global constant values
│   ├── states/         # State definitions for Compose UIs
│   └── widget/         # Custom widgets and Composables
├── data/               # Data layer (source of truth)
│   ├── local/          # Room DB, DAOs, SharedPrefs
│   ├── model/          # DTOs, API models
│   ├── remote/         # Retrofit, APIs, network logic
│   └── repository/     # Repository implementations
├── domain/
│   └── useCases/       # Business logic grouped by feature
│       ├── admin/      # feature1
│       └── user/       # feature2
├── presentation/       # UI + ViewModel per feature
│   ├── home/
│   ├── login/
│   ├── register/
│   ├── main/
│   ├── navHost/            # Navigation controller
├── provider/               # Wrappers for context-level services
│   ├── resource/           # Resource helpers (e.g., localization)
│   └── sharedpreference/
```

✅ Responsibilities by Layer

See table above for structure

✅ Data Flow Summary

UI → ViewModel → UseCase → Repository → Remote/Local

✅ Rules & Best Practices

– Single Responsibility

– No Logic in Composables

– Testable UseCases

– Immutable UI State

– DI Everywhere (Hilt)

– Map DTO to Domain Models

✅ Coding Guidelines

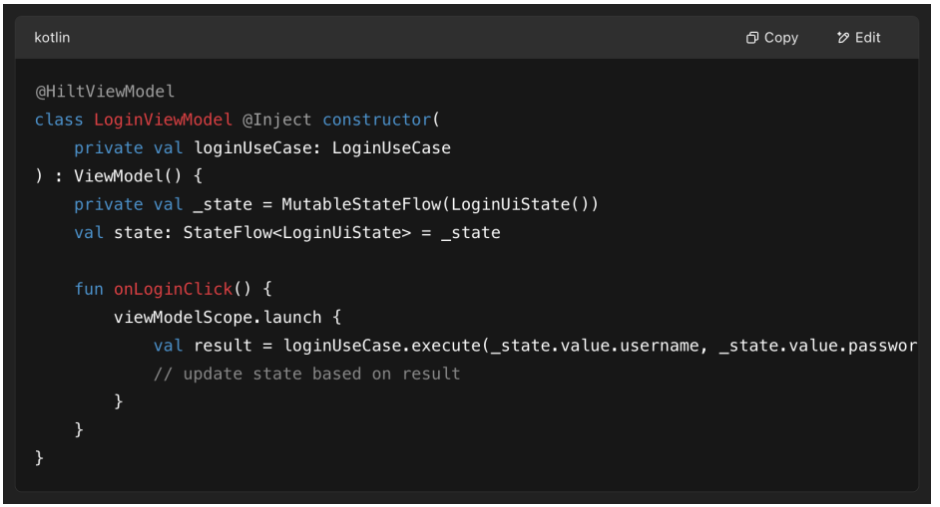
Details on UI, ViewModel, UseCase, Repository layers

1. **UI Layer (Jetpack compose):**

- Pure Ui logic
- Never call repository or usecase directly
- observe viewmodel via state.

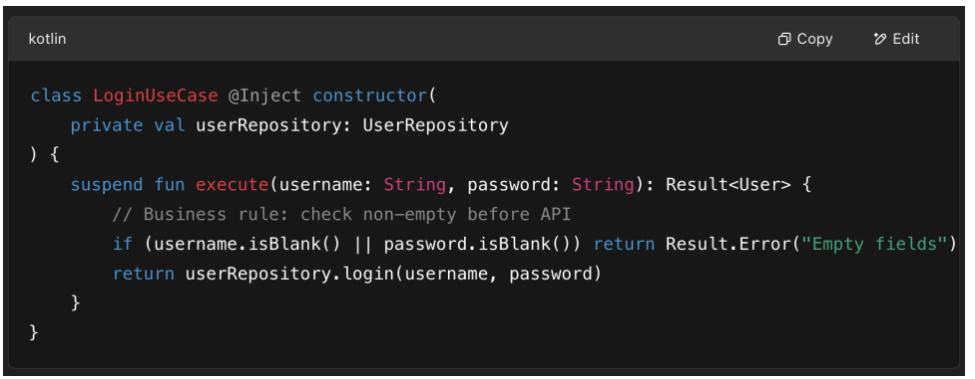
2. ViewModel Layer

- One viewModel per screen.
- Should never know implementation details (only UseCases).
- ViewModel owns the screen state.



3. UseCases

- Perform business logic
- Only depend on repository interface
- Independent from Android/Jetpack Compose.



4. Repository

- implementation belogs in data/repository
- should seperate remote and local operations

kotlin Copy Edit

```kotlin
class UserRepositoryImpl @Inject constructor(
    private val api: UserApi
) : UserRepository {
    override suspend fun login(username: String, password: String): Result<User> {
        val dto = api.login(username, password)
        return Result.Success(dto.toDomain())
    }
}
```

5. Nav-graph Usage

- NavGraphUtil.gotoScreen(navController, Constants.NavDestinationScreens.REGISTER_SCREEN, true)

```
Spacer(modifier = Modifier.height(16.dp))
TextButton(onClick = {
    NavGraphUtil.gotoScreen(navController, Constants.NavDestinationScreens.REGISTER_SCREEN, true)
) {
    Text(text = "Create New Account.")
}
```