Creating an Android App Using Android Studio and Setting up a UI Test Case

Name of Testers:
1. Anupam
2. Asha
3. Harshvardhan
4. Hrushikesh
5. Manish
6. Krishna
7. Tahi

Version: 1.7
Created: 05/31/2024
Last Updated: 05/31/2024 Status:
Complete

Revision Sheet

Version	Date	Author	Description of Change
1.1	5/30/2024	Krishna	Initial Draft: Step by step details to create project
1.2	5/30/2024	Asha	
1.3	5/30/2024	Harsha Vardhan	
1.4	5/30/2024	Hrushikesh	Revised Draft: with added pictures
1.5	5/30/2024	Tahir	
1.6	5/30/2024	Manish	
1.7	5/30/2024	Anupam	Final Draft: UI Testing

Introduction

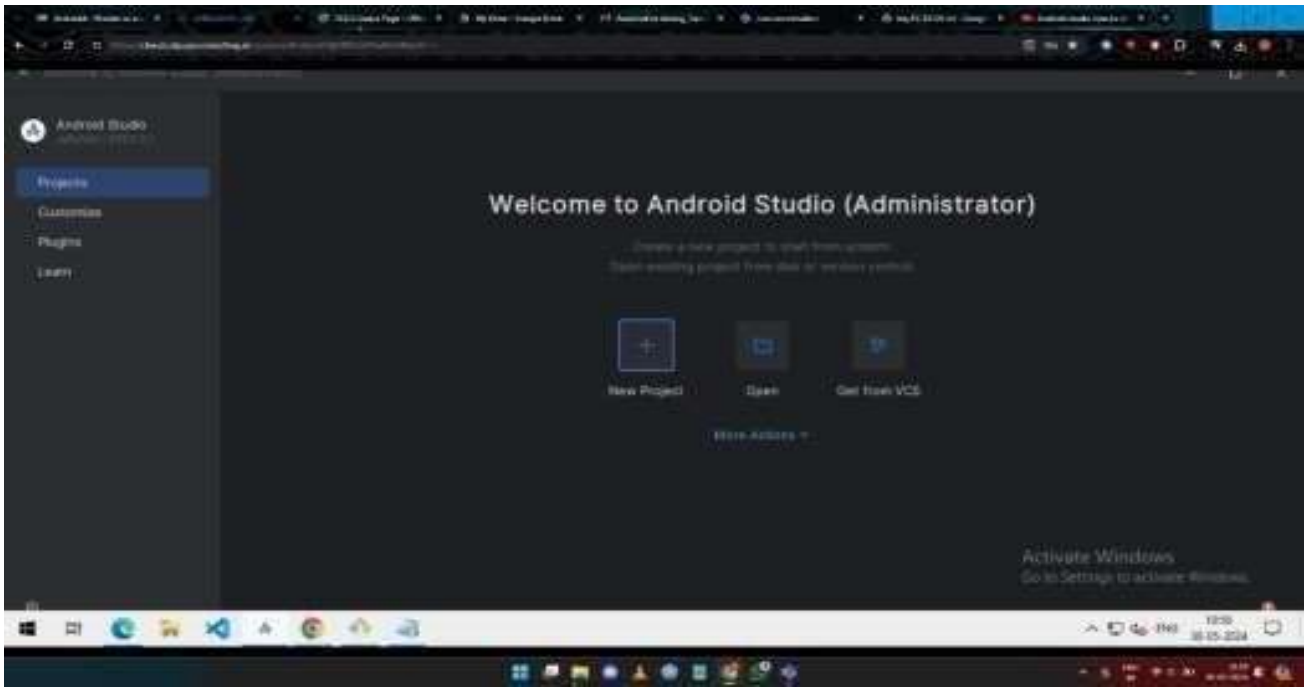
Creating a project in Android Studio is the first step towards building your own Android application. The step-by-step flow to create a project is as follows

1. Open Android Studio

Launch Android Studio on your computer. If you haven't installed it yet, you can download it from the official Android Developer website.

2. Start a New Project

Once Android Studio is open, you'll see a welcome screen. Click on Start a new Android Studio project or select File > New > New Project... from the menu.

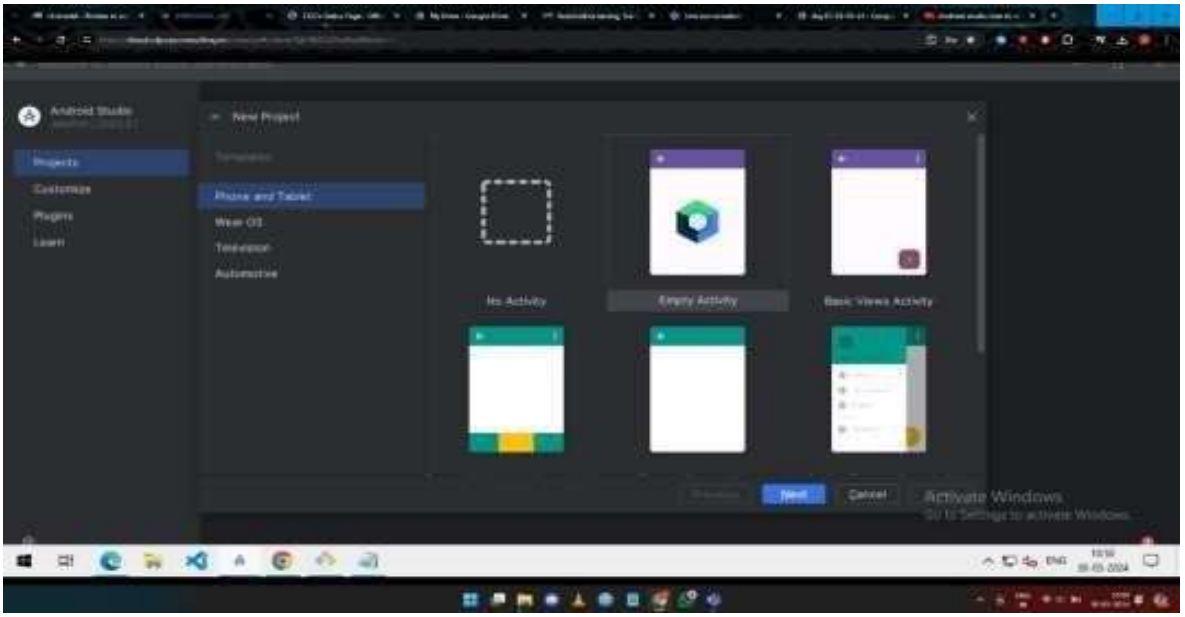


3. **Configure Your Project**

In the Create New Project window, you'll be prompted to configure your new project:

a. Select a Template

Choose the type of application you want to create (e.g., Phone and Tablet, Wear OS, TV, Automotive). Selecting Phone and Tablet is the most common choice for creating standard Android applications.



b. Choose a Form Factor

Select the device form factor you want your app to run on (e.g., Phone and Tablet, Wear, TV, Automotive).

c. Select a Design

Choose a design for your app's user interface (e.g. Empty Activity, Basic Activity, Navigation Drawer Activity). For this walkthrough, let's choose Empty Activity, which provides a starting point with a single blank screen.

d. Enter Details

Provide a name for your application (e.g. MyFirstApp). You can also specify the package name, save location, and language (Java or Kotlin).

e. Choose Minimum API Leve

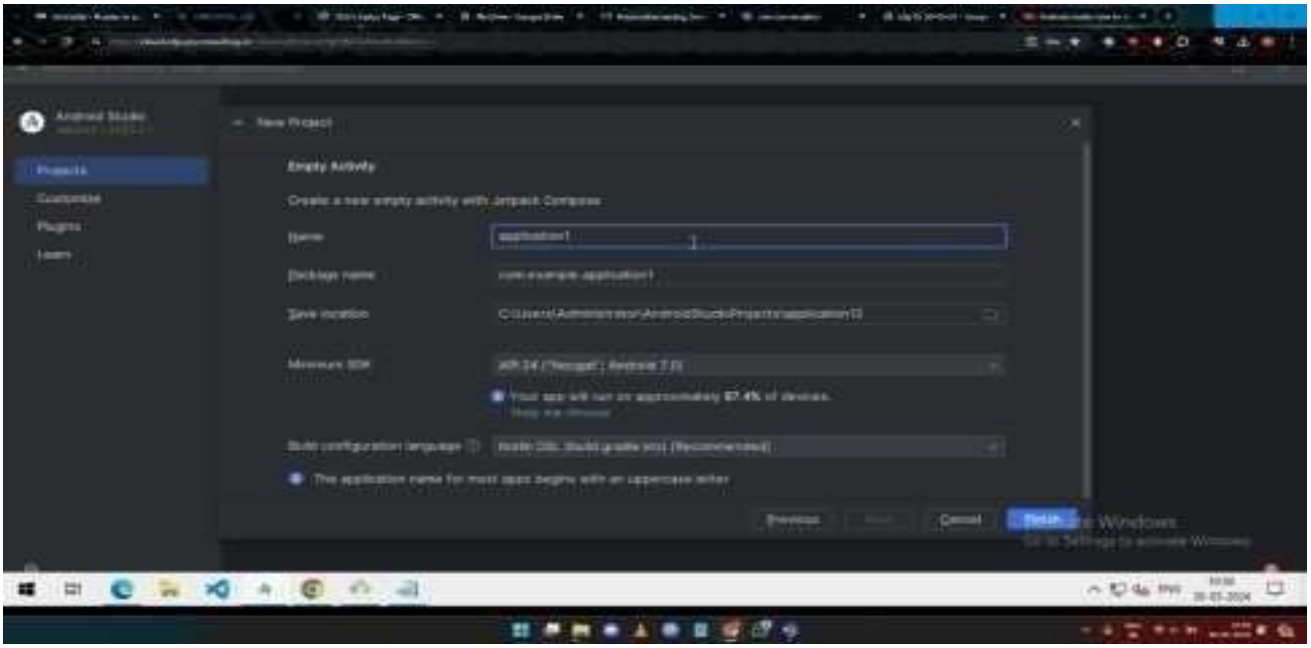
Select the minimum API level supported by your app. This determines the oldest Android version your app can run on. You can choose the default value or select a specific API level based on your requirements.

f. Add Kotlin Support (Optional)

If you prefer to use Kotlin as the programming language, you can enable Kotlin support by checking the box next to Include Kotlin support.

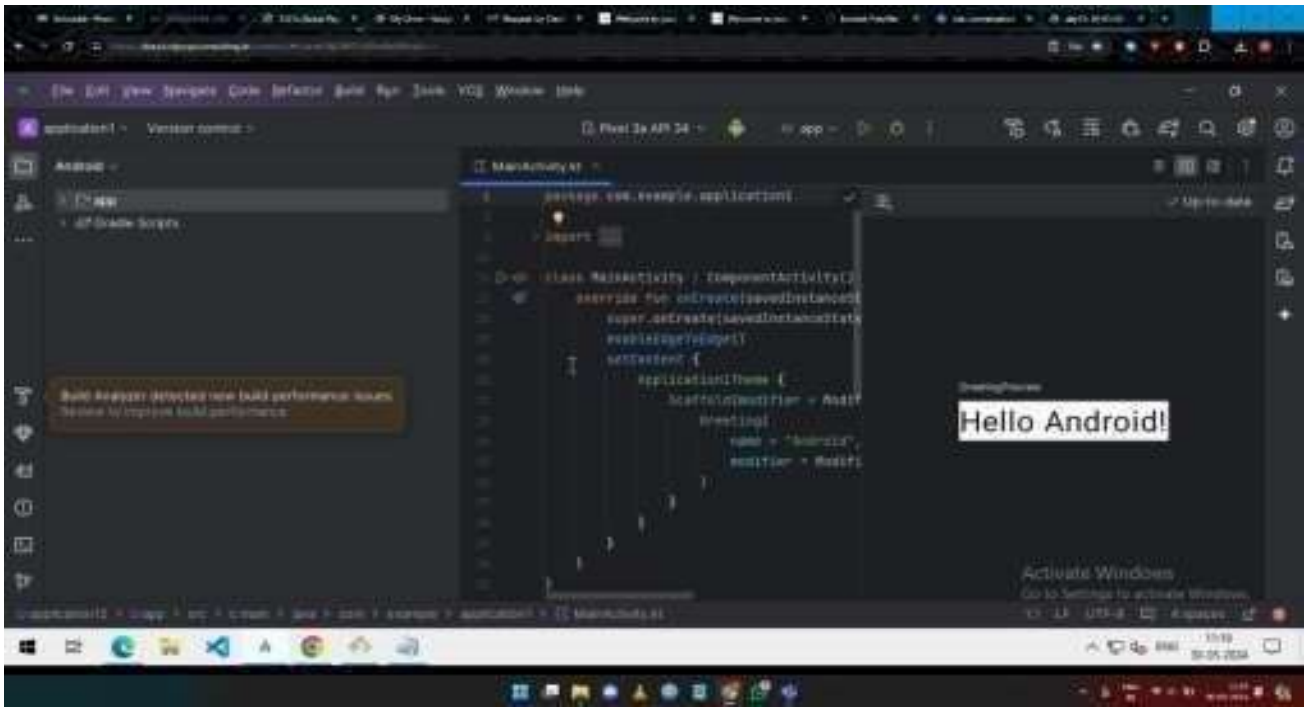
4. **Create the Project**

After configuring your project settings, click on the Finish button. Android Studio will then create the project structure and files based on your selections.



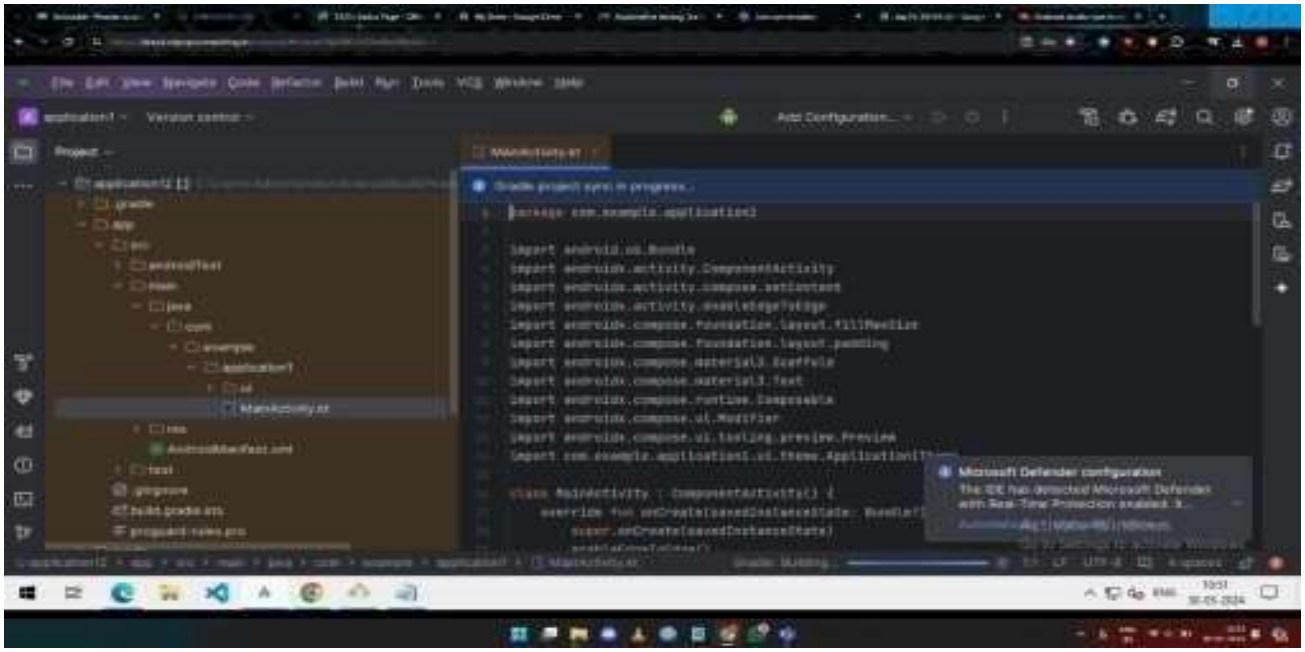
5. **Wait for Gradle Sync**

Android Studio will start syncing your project with Gradle, the build system used for Android projects. This process may take a few moments as it downloads dependencies and sets up the project environment.



6. **Project Setup Complete**

Once Gradle sync is complete, you'll see your project loaded in Android Studio. You'll have a project structure on the left-hand side, including folders like app, res, and Gradle Scripts. The main code editor will display the contents of your project's main activity file.

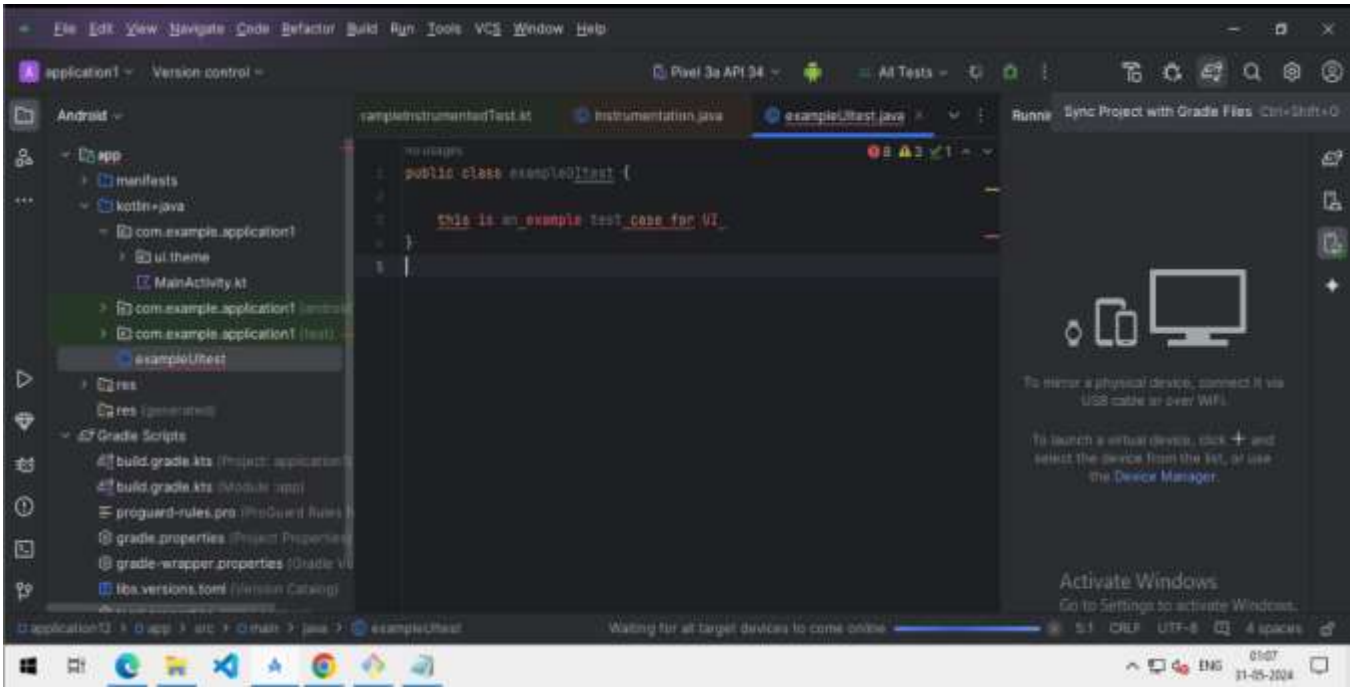


At this point, we've successfully created an empty project in Android Studio. We can now start developing your Android application by adding code, resources, and other components to achieve your desired functionality.

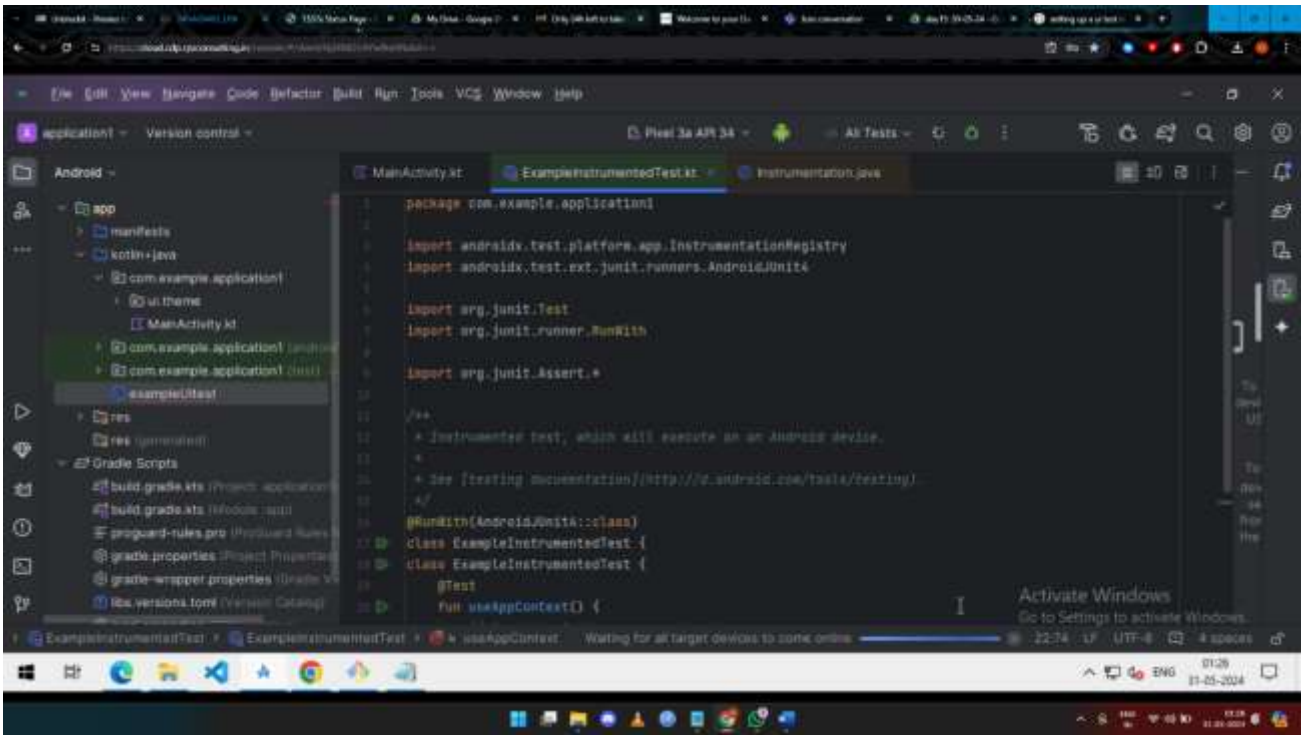
UI Testing

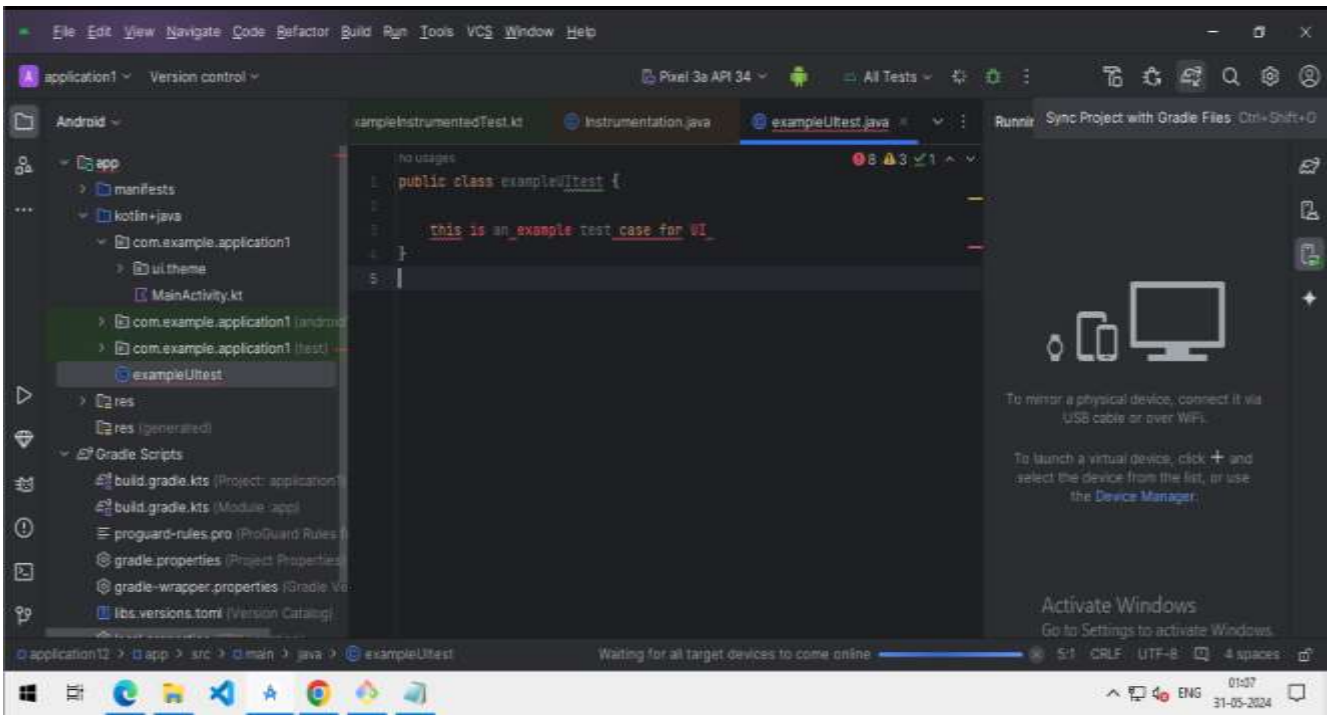
Setting up a UI test in Android Studio involves creating a new test class, configuring the test dependencies, and writing test code to interact with the UI elements of your Android application. Here's a step-by-step guide to set up a UI test (We have only created a UI test file and navigated as to where do we have to click once all the test scripting has been done). It has been done because we have yet to understand how test case scripting is done.

1. **Create a Test Class**: Right-click on the androidTest folder (usually located under the app module) in the Project view. Navigate to "New" > "Java Class" and then name your test class (e.g., ExampleUITest) and click "OK" to create it.

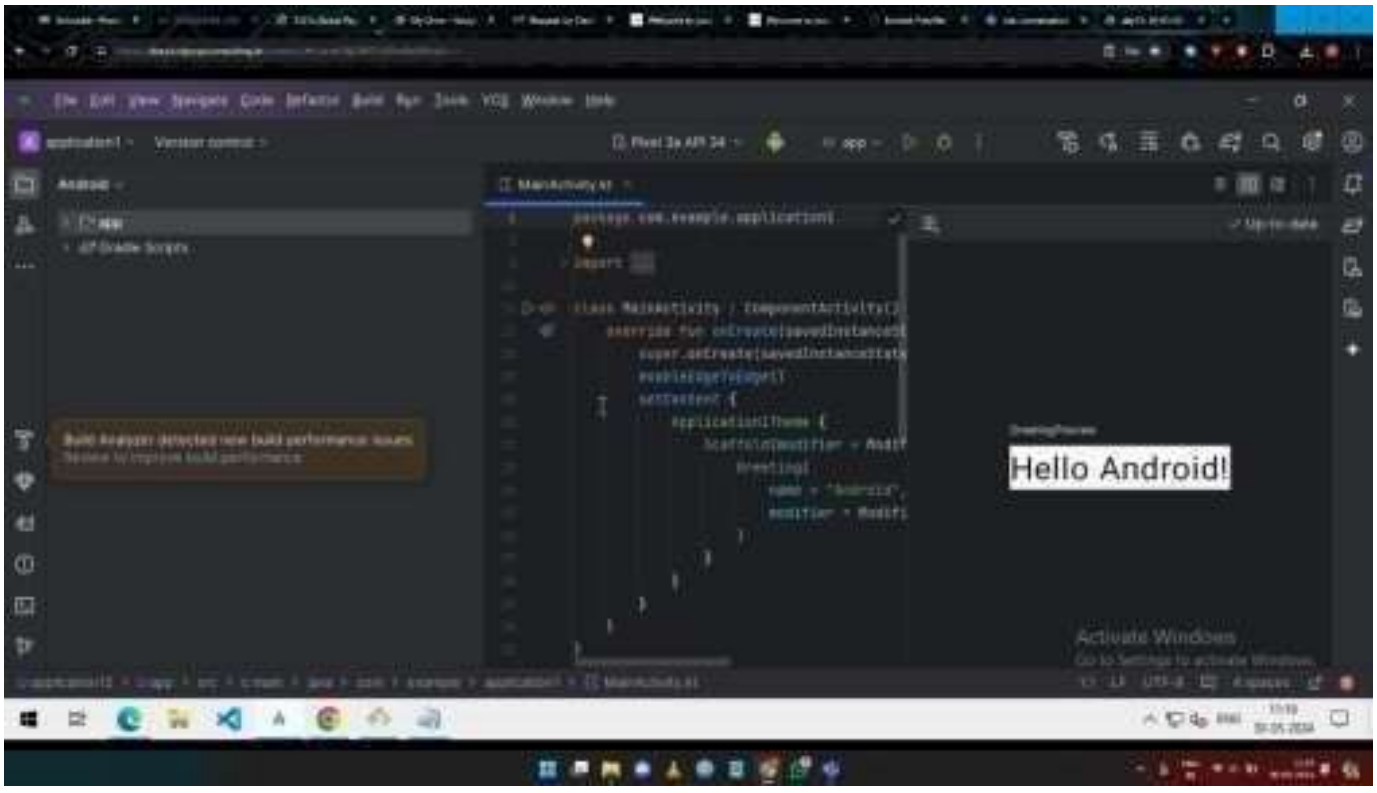


2. **Sync Gradle:** After adding the dependencies, click on "Sync Now" at the top right corner of the screen to sync your project with Gradle.





3. **Write Your UI Test:**
    - Open the test class you created (`ExampleUITest.java`) in the `androidTest` package.
    - Write your UI test code using Espresso, the UI testing framework for Android. Below is a simple example of a UI test that verifies if a TextView with the id "textView" displays the text "Hello,Android!":



4. **Run Your UI Test:**
    - Right-click on your test class (`ExampleUITest.java`) or the test package in the Project view.
    - Select "Run 'ExampleUITest'" or "Run 'Tests in 'com.example.app'" from the context menu.
    - Android Studio will launch an emulator or connect to a physical device to run the UI test. You'll see the test results in the Run panel at the bottom of the screen.

Methods for UI testing

1. Espresso

What it is: Espresso is an open-source testing framework specifically designed for Android UI testing. It's developed and maintained by Google and known for its simplicity and reliability.
What it does: Espresso allows you to write automated tests that simulate user interactions with your app's UI elements. This includes tapping buttons, entering text, swiping through screens, and verifying the expected outcomes.
Benefits:
Faster and more reliable testing: Compared to manual testing, Espresso automates repetitive tasks, reducing the time and effort required for testing.
Improved test coverage: You can write tests for various functionalities and UI elements, ensuring more comprehensive testing.
Early bug detection: By automating tests, you can catch bugs early in the development lifecycle, making them easier and cheaper to fix.
Focus: Espresso primarily focuses on testing the functionality and behavior of your app's UI elements.
Example: You can write an Espresso test to verify that clicking a "Login" button takes the user to a successful login screen with a welcome message.

2. UI Automator

What it is: UI Automator is a more comprehensive testing framework built into the Android SDK. While similar to Espresso, it offers more power and flexibility.
What it does: UI Automator allows you to automate interactions not just with your app, but with the entire Android system. This includes interacting with system settings, launching other apps, and performing complex gestures.
Benefits:
Advanced system-wide testing: You can automate tasks that involve interaction with multiple apps or system settings.
Greater control: UI Automator offers more control over the testing environment, allowing for more complex test scenarios.
Focus: UI Automator focuses on automating interactions with the entire Android system, including your app and other elements.
Example: You can write a UI Automator test to simulate enabling Wi-Fi, launching your app, and then performing specific actions within the app.

3. Snapshot Testing

What it is: Snapshot testing is a different approach to app testing. It doesn't involve interacting with the app, but rather focuses on how the app looks visually.
What it does: Snapshot testing captures screenshots of your app's screens at different stages and compares them to previously stored reference images. Any unexpected visual changes will trigger a test failure.
Benefits:
Snapshot testing helps identify unintended changes in the app's layout, colors, fonts, or other visual elements.
Maintaining consistent UI: By detecting visual regressions early, developers can ensure a consistent user experience across app updates.
Focus: Snapshot testing focuses solely on the visual appearance of the app's UI elements.
Example: You can capture a snapshot of the login screen and store it as a reference image. Later tests will compare the current login screen with the reference image to detect any visual changes.

Conclusion

Espresso and UI Automator automate user interaction with the app or system, while Snapshot testing focuses on the app's visual appearance.
Espresso is simpler and ideal for UI testing, while UI Automator offers more power for complex system-wide testing.
All three techniques work together to ensure a well-functioning, visually appealing, and bug-free Android app.

THANK YOU