Mobile Automation Testing with Appium – Client-Server Architecture

Author Information

Author: Abdul Rashid
Designation: Software Engineer

Table of Contents

1. What is Automation Testing?

Automation Testing is a software testing technique where tests are executed automatically using specialized tools instead of manual effort. Test scripts are written once and reused multiple times. Tools interact with applications just like a real user (clicking buttons, typing text, validating output).

Example: Instead of manually opening an app and logging in 100 times, automation can do it in seconds.

2. Purpose of Automation Testing

The main goals of automation testing are:

• Save time & cost – Execute hundreds of test cases quickly.

• Increase accuracy – Avoid human errors in repetitive tests.

• Reusability – Same scripts can run on different devices/OS.

• Continuous Testing – Integrate with CI/CD pipelines for fast feedback.

• Scalability – Run parallel tests on multiple devices/emulators.

3. What is Mobile?

A mobile is a portable computing device (smartphone or tablet) that runs applications and connects to networks.

3.1 Types of Mobile

1. Feature Phones → Basic devices (calls, SMS).

2. Smartphones → Run advanced operating systems (Android, iOS).

3. Tablets → Larger screen, same OS as smartphones.

4. What is Mobile Automation?

Mobile Automation means testing mobile apps automatically using tools instead of manual testing. Apps tested include native apps, hybrid apps, and web apps. Platforms include Android (emulators/real devices) and iOS (simulators/real devices).

Example: Verify if a login button works on 20 devices without a tester manually clicking each one.

5. What is Appium?

Appium is an open-source automation testing tool for mobile apps. It supports Android and iOS, and automates native, hybrid, and web apps. Appium uses the WebDriver protocol, similar to Selenium, allowing the same test script to run across platforms.

Key drivers:

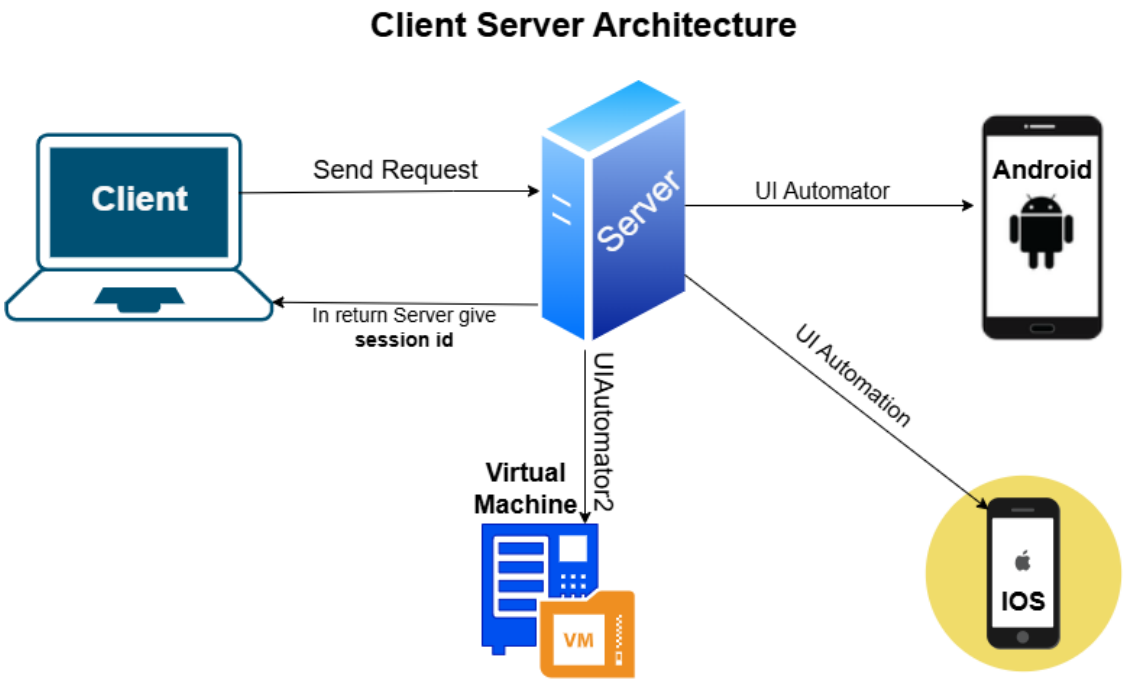
• UIAutomator2 → Android automation driver.

• XCUITest → iOS automation driver.

6. What is Client-Server Architecture?

Client-Server architecture is a model where the client (test script) sends requests and the server (Appium) receives and processes them. The Appium server forwards these requests to the appropriate driver, which then executes them on the mobile device. The result is returned back to the client.

7. Diagram



8. Explanation of the Diagram

The diagram below represents the Appium Client-Server architecture:

1. Client (Test Script): Written in Java, Python, JS, etc. Sends requests to Appium Server.
2. Server (Appium Server): Acts as a bridge between the test script and mobile devices. Receives requests, creates a session ID, and forwards commands to the correct driver.
3. Drivers: Translate Appium commands into OS-level instructions. UIAutomator2 for Android, XCUITest for iOS.
4. Virtual Machine (VM): Hosts Android emulators or iOS simulators.
5. Android Device: Executes commands using UIAutomator2 driver through ADB.
6. iOS Device: Executes commands using XCUITest driver through WebDriverAgent.
7. Response: Device executes the action and returns the result to Appium Server, which sends it back to the client.

9. Client Request Execution

The client sends a request that represents the activity it wants to perform. For example, if the client (test script) wants to click a button, type text, or swipe on the screen, that exact same action is performed on the connected mobile device (whether Android or iOS). In other words, whatever action the client requests, Appium ensures that the device performs it.

10. How the Request is Created?

A request in Appium is created automatically by the Appium Client libraries (Java, Python, JS, etc.) when you write test commands. Each command, such as finding an element or clicking a button, is converted into an HTTP JSON request according to the WebDriver protocol.

Example – Creating a Session Request:

```
POST /session HTTP/1.1
Content-Type: application/json
Host: 127.0.0.1:4723

{
  "capabilities": {
    "alwaysMatch": {
      "platformName": "Android",
      "automationName": "UIAutomator2",
      "deviceName": "emulator-5554",
      "app": "C:\\apps\\myApp.apk"
    }
  }
}
```

In this example:

- 'POST /session' tells the Appium server to create a new session.
- 'capabilities' describe the platform, automation driver, device, and application.
- The Appium Server receives this request, establishes a session, and returns a session ID.
- All subsequent actions (click, type, swipe) are sent as HTTP JSON requests with that session ID.

11. Step by Step: How a New User Can Perform Automation Testing

The following steps guide a beginner to start automation testing using Appium:

1. Step 1: Install Java Development Kit (JDK) and set JAVA_HOME environment variable.

2. Step 2: Install Android Studio and configure Android SDK and AVD (Android Virtual Device).
3. Step 3: Install Node.js (required for Appium).
4. Step 4: Install Appium Desktop or Appium Server CLI.
5. Step 5: Install Appium Client Library in your preferred programming language (Java, Python, JavaScript, etc.).
6. Step 6: Set environment variables (ANDROID_HOME, PATH for SDK tools).
7. Step 7: Start Appium Server (via Appium Desktop or command line).
8. Step 8: Create a new test project in IDE (e.g., IntelliJ, Eclipse, PyCharm, VSCode).
9. Step 9: Define Desired Capabilities (platformName, deviceName, automationName, app path).
10. Step 10: Write test scripts (e.g., open app, click button, enter text).
11. Step 11: Run the test script – Appium will connect with the emulator/device and perform the requested actions.
12. Step 12: Analyze the test results (pass/fail) and debug if required.