# Day 7 : Mobile Application Testing Architecture and Setup Using JavaScript

## Learning Objectives

By the end of this lesson, you will:

Understand the basics of mobile application testing using Appium.
Learn how to set up Appium for automated testing.
Write and execute test scripts for mobile applications using JavaScript with Appium.
Apply best practices for effective mobile application testing.

---

## Introduction

Appium is a popular open-source framework for automating mobile applications. It supports cross-platform testing on Android and iOS, making it versatile for modern app development. Using JavaScript with Appium, testers can write scripts to validate app functionality, UI, and performance across various devices and operating systems.

---

## Key Concepts and Definitions

1. Appium:
   A cross-platform testing tool that uses WebDriver to automate mobile apps.
   Works with native, hybrid, and mobile web applications.
2. Capabilities:
   Key-value pairs used to configure and customize test environments.
   Examples: `platformName`, `deviceName`, `app`, `automationName`.
3. Appium Server:
   Acts as a bridge between the test script and the mobile device.
   Receives commands from the client (test script) and executes them on the device.
4. WebDriver Protocol:
   A standard API used for interacting with browsers and mobile apps.
5. Appium Inspector:
   A GUI tool to identify elements in mobile applications.

---

# Step-by-Step Explanation

# 1. Set Up Appium

# Install Node.js

Appium requires Node.js as a runtime environment.
Verify installation:

```
bash
Copy code
node -v
npm -v
```

# Install Appium

Use the npm package manager to install Appium globally:

```
bash
Copy code
npm install -g appium
```

# Start Appium Server

Run the Appium server from the command line:

```
bash
Copy code
appium
```

## Install Appium Inspector

Download and install the Appium Inspector tool from Appium Inspector GitHub.

---

# 2. Configure the Environment

## Prepare the Mobile Device

For Android:

Enable Developer Options and USB Debugging.

Install Android Studio and configure the `ANDROID_HOME` environment variable.

For iOS:

Install Xcode and configure a simulator.

## Define Desired Capabilities

Desired capabilities tell Appium the details of the test environment.
Example configuration for an Android device:

```
javascript
Copy code
const capabilities = {
    platformName: "Android",
    platformVersion: "10.0",
    deviceName: "Pixel_3",
    app: "/path/to/your/app.apk",
    automationName: "UiAutomator2"
};
```

---

# 3. Write Test Scripts

# Install WebDriverIO

Appium integrates seamlessly with WebDriverIO for writing tests.
Install WebDriverIO and its dependencies:

```
bash
Copy code
npm init wdio .
npm install @wdio/cli @wdio/local-runner @wdio/mocha-framework @wdio/spec-reporter
```

# Sample Test Script

A basic script to test a login button:

```
javascript
Copy code
const { remote } = require("webdriverio");

const options = {
    path: "/wd/hub",
    port: 4723,
    capabilities: {
        platformName: "Android",
        platformVersion: "10.0",
        deviceName: "Pixel_3",
        app: "/path/to/your/app.apk",
        automationName: "UiAutomator2"
    }
};

(async () => {
    const driver = await remote(options);

    // Locate the login button and perform a click action
    const loginButton = await driver.$('~LoginButton');
    await loginButton.click();

    console.log("Login button clicked!");

    // End the session
    await driver.deleteSession();
})();
```

---

## 4. Debugging and Inspection

## Use Appium Inspector

Launch Appium Inspector to inspect the app and locate UI elements.
Use the element identifiers (`id`, `class`, `accessibilityId`, `xpath`) in your test scripts.

## Device Logs

Monitor logs for debugging using `adb logcat` (Android) or Xcode's logs (iOS).

---

## 5. Execute Test Scripts

Run the test script using Node.js:

```
bash
Copy code
node test-script.js
```

Verify the actions are performed on the mobile device or emulator.

---

# Examples

Example 1: Testing a Login Page

Input username and password fields and click login.
Validate the response message or navigation.

```
javascript
Copy code
```

```
const usernameField = await driver.$('~username');
const passwordField = await driver.$('~password');
const loginButton = await driver.$('~LoginButton');

await usernameField.setValue("testuser");
await passwordField.setValue("password123");
await loginButton.click();

const successMessage = await driver.$('~successMessage');
console.log(await successMessage.getText());
```

---

# Practice Exercises

1. Basic Setup:
   Set up Appium and configure desired capabilities for an emulator.
2. Test UI Elements:
   Write a script to validate the presence of buttons, text fields, and labels in a sample app.
3. Perform User Actions:
   Automate user actions like scrolling, swiping, and tapping.
4. Advanced Testing:
   Validate error messages for invalid login attempts.
   Test app behavior under different network conditions.

# Summary

Appium is a powerful tool for cross-platform mobile app testing.
Setting up the testing environment involves configuring Appium, desired capabilities, and the test script.
Appium Inspector simplifies element identification, making test scripting easier.
Writing reusable test scripts ensures efficient and consistent testing processes.