Mobile Testing

- **Step by Step Emulator setup process**

Step 1: Install Java Development Kit (JDK)

- **Download and Install JDK**:
  - Download the latest version of the JDK from Oracle's JDK page or install OpenJDK.
  - Follow the installation instructions based on your operating system (Windows, macOS, Linux).
- **Set up JAVA_HOME**:
  - On **Windows**:
    - Right-click on "This PC" -> Properties -> Advanced System Settings -> Environment Variables.
    - Under "System Variables", click "New" and set `JAVA_HOME` to the path of your JDK installation (e.g., `C:\\Program Files\\Java\\jdk-11`).

On **macOS/Linux**: Add the following line to your `.bashrc` or `.zshrc`:Then, run `source ~/.bashrc` (or `source ~/.zshrc`) to apply the changes.

bash
Copy code
```
export JAVA_HOME=$(/usr/libexec/java_home)
export PATH=$JAVA_HOME/bin:$PATH
```

  -
- **Verify Installation**:

Open a terminal or command prompt and type:

bash
Copy code
```
java -version
```

        -
        - This should show the version of the JDK that you installed.

- ---

Step 2: Install Android Studio and Set Up Emulator

    - **Download Android Studio**:
        - Download and install Android Studio.
    - **Set up Android SDK**:
        - During the installation of Android Studio, ensure the Android SDK is selected for installation.
        - After installation, open Android Studio and configure the SDK through the SDK Manager:
            - Open Android Studio > Preferences (or Settings on Windows) > Appearance & Behavior > System Settings > Android SDK.
            - Install the latest SDK and platform tools (e.g., Android 11 or higher).
    - **Set up Emulator**:
        - Open Android Studio and go to **Tools > AVD Manager**.
        - Click **Create Virtual Device** to set up a new emulator.
        - Choose a device (e.g., Pixel 4) and an Android version (make sure it's one you want to test on).
        - Follow the prompts to finish setting up the emulator.
        - Once set up, click the green **Play** button in AVD Manager to start the emulator.

- ---

Step 3: Install Appium

    - **Install Node.js**:

        - Appium is built on Node.js, so you must have it installed. Download and install Node.js.
    - **Install Appium using npm**:

Open a terminal or command prompt and run the following command to install Appium globally:

bash
Copy code
npm install -g appium

        -

- **Verify Installation**:

After installation, verify by typing:

bash
Copy code
appium -v

-
- This will return the installed Appium version.

Step 4: Install Appium Desktop (Optional)

Appium Desktop is a GUI tool for inspecting elements and starting Appium servers easily.

- **Download Appium Desktop**:
  - Go to the Appium Desktop GitHub Releases page and download the appropriate version for your OS.
- **Install Appium Desktop**:
  - Install and launch the Appium Desktop app.
  - Click on the **Start Server** button in Appium Desktop to launch the Appium server.

Step 5: Set Up Android Device for Automation

- **Enable Developer Options**:
  - On the emulator, go to **Settings > About phone**.
  - Tap on the **Build number** 7 times to enable Developer Options.
- **Enable USB Debugging**:
  - Go to **Settings > Developer options**.
  - Enable **USB debugging**.
- **Install Appium's Android Driver**:
  - Appium supports Android automation through the **UiAutomator2** driver, which is installed by default.

If not, you can manually install it via:

bash
Copy code

appium driver install uiautomator2

- ---

Step 6: Create a Test Script for Appium

Create a basic test script in Java (or any other supported language) to interact with the emulator.

- **Install Appium Client for Java**:

Use Maven or Gradle to install the Appium Java client. If using Maven, add the following dependency in `pom.xml`:

xml
Copy code

```xml
<dependency>
  <groupId>io.appium</groupId>
  <artifactId>java-client</artifactId>
  <version>7.6.0</version>
</dependency>
```

- **Write the Test Script**:

Create a Java file (e.g., `AppiumTest.java`) with the following content:

java
Copy code

```java
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.RemoteWebDriver;
import java.net.URL;

public class AppiumTest {
    public static void main(String[] args) throws Exception {
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability("platformName", "Android");
```

```
        capabilities.setCapability("deviceName", "emulator-5554");  // Emulator ID
        capabilities.setCapability("platformVersion", "11.0");  // Android version
        capabilities.setCapability("appPackage", "com.android.calculator2");  // App Package
        capabilities.setCapability("appActivity", "com.android.calculator2.Calculator");  // App
Activity

        URL url = new URL("<http://127.0.0.1:4723/wd/hub>");  // Appium server URL
        WebDriver driver = new AndroidDriver<AndroidElement>(url, capabilities);

        // Example test: Perform a basic calculation
        driver.findElementById("com.android.calculator2:id/digit_7").click();
        driver.findElementById("com.android.calculator2:id/op_add").click();
        driver.findElementById("com.android.calculator2:id/digit_3").click();
        driver.findElementById("com.android.calculator2:id/eq").click();

        String result = driver.findElementById("com.android.calculator2:id/result").getText();
        System.out.println("Calculation result: " + result);

        driver.quit();
    }
}
```

-
- This script opens the Android emulator, launches the Calculator app, performs a calculation, and prints the result.

- ---

Step 7: Start the Appium Server

  - **Start Appium Server**:

You can start the Appium server from the command line by typing:

```
 bash
Copy code
appium
```

-
- If you're using **Appium Desktop**, click on **Start Server** to start the server.

- ---

  ### Step 8: Run the Test Script

  - **Execute the Test**:
    - Run the test script from your IDE (e.g., IntelliJ IDEA, Eclipse).
    - Appium will interact with the emulator, execute the test, and output the results.

- ---

  ### Troubleshooting Tips:

  - **Emulator Not Found**: Ensure the emulator is running and check the emulator ID by running `adb devices` from the terminal.
  - **Appium Server Not Starting**: Ensure the correct Java version is set in `JAVA_HOME` and that no other services are using port `4723`.

General Mobile Testing Concepts

1. What is mobile testing, and why is it important?

- **Mobile Testing** refers to the process of testing mobile applications (apps) to ensure they function as intended across various devices and platforms (iOS, Android).
- **Importance**:
  - Ensures the app works seamlessly across different mobile devices, operating systems, and network conditions.
  - Mobile apps are critical for user engagement and business operations, so testing ensures a positive user experience, reliability, and performance.

2. What are the differences between mobile application testing and web application testing?

- **Mobile Application Testing** involves testing apps on mobile devices, ensuring compatibility with various hardware and operating systems (e.g., Android/iOS). It also considers factors like screen sizes, battery consumption, and touch gestures.
- **Web Application Testing** focuses on testing applications in a browser environment, checking compatibility with different browsers (Chrome, Firefox, etc.), and network conditions.
- **Key Differences**:
  - Mobile testing must consider touch interaction, hardware-specific features (GPS, camera), network fluctuations, battery performance, and device fragmentation.
  - Web testing mainly focuses on browser compatibility and user interaction through a mouse/keyboard.

3. What are the different types of mobile applications?

- **Native Apps**: Built for a specific platform (iOS/Android) using platform-specific programming languages (Swift/Java for iOS/Android). Example: Instagram, WhatsApp.
- **Web Apps**: Accessed via a web browser, responsive to different screen sizes but not installed on the device. Example: Mobile versions of Facebook, Twitter.
- **Hybrid Apps**: Combines elements of both native and web apps. Built using web technologies (HTML, CSS, JavaScript) and wrapped in a native container to run on a device. Example: Uber, Instagram.

4. What are the major challenges in mobile testing?

- **Device Fragmentation**: Multiple devices with different screen sizes, resolutions, and operating systems can complicate testing. Solution: Use a combination of real devices and emulators for broad coverage.
- **Network Issues**: Testing mobile apps under varying network conditions (e.g., 3G, 4G, Wi-Fi, low signal) is critical. Tools like **Network Link Conditioner** can simulate network issues.
- **Battery Performance**: Testing how an app affects battery life. Overuse of resources can drain the battery quickly. Solution: Use tools like **Battery Historian** to track battery usage.
- **Touch and Gesture Testing**: Simulating gestures like swiping, pinching, tapping, etc., on a range of devices to ensure proper handling.

5. How do you ensure compatibility testing for mobile apps?

- **Test on Multiple Devices**: Perform testing on various mobile devices (real and emulators) with different screen sizes, OS versions, and hardware configurations.
- **Automated Compatibility Testing Tools**: Use tools like **Sauce Labs**, **BrowserStack**, or **Firebase Test Lab** to run tests on a variety of devices and OS combinations.
- **Cross-Browser and Cross-Platform Testing**: Ensure the app is compatible with different versions of Android and iOS and works well across browsers in mobile web applications.

Mobile Testing Types

1. What are the key types of mobile testing?

- **Functional Testing**: Ensures that the app's core functionality works as expected (login, user registration, navigation, etc.).
- **UI/UX Testing**: Verifies the app's user interface is intuitive, responsive, and visually appealing. Ensures a smooth user experience (swiping, scrolling, zooming).
- **Performance Testing**: Measures how well the app performs under different conditions (load, memory usage, CPU utilization).

- **Compatibility Testing**: Ensures the app works across different devices, OS versions, and screen sizes.
- **Security Testing**: Verifies that the app is protected from vulnerabilities like data leaks, unauthorized access, etc.

2. What is the difference between functional and non-functional testing for mobile apps?

- **Functional Testing**: Focuses on the core functionalities of the app like login, payment, data processing, etc.
- **Non-Functional Testing**: Focuses on performance, usability, security, and other aspects not directly related to functionality.

3. What is usability testing, and how is it conducted for mobile apps?

- **Usability Testing** evaluates the ease of use and user experience (UX) of the app. This is conducted by real users who perform tasks within the app.
- **Steps**:
    1. Identify test scenarios (e.g., signing up, using a specific feature).
    2. Observe users interacting with the app and gather feedback.
    3. Analyze if users face any issues or difficulties.

4. What is regression testing in mobile applications?

- **Regression Testing** ensures that new updates or changes to the app have not negatively impacted existing functionalities.
- Example: After adding a new feature (e.g., payment integration), regression tests verify that existing features like login or profile updates still work correctly.

5. What is interruption testing in mobile apps?

- **Interruption Testing** checks how well the app handles interruptions like incoming calls, SMS, low battery warnings, or network issues.
- Example: Verify that when the app is interrupted by an incoming call, it resumes from where the user left off after the call ends.

Mobile Testing Tools

1. What tools have you used for mobile testing?

- **Appium**: Open-source tool for automating mobile apps (both Android and iOS).
- **Espresso**: Android-specific testing framework for writing automated tests.

- **XCUITest**: iOS-specific testing framework from Apple for UI testing.
- **Selendroid**: A test automation framework for Android applications.
- **Robotium**: A tool for automated testing of Android apps.

2. How do you automate mobile app testing?

- **Use of Automation Frameworks**: Tools like **Appium**, **Espresso**, and **XCUITest** can automate UI and functional testing of mobile apps.
- **Test Scripts**: Write scripts using Java, Python, or JavaScript to simulate user interactions (e.g., tapping, swiping, text input).
- **Parallel Execution**: Use cloud-based tools like **Sauce Labs** or **BrowserStack** to run tests on multiple devices simultaneously.

3. What is the difference between Appium and Espresso?

- **Appium**: Cross-platform tool that supports Android and iOS. It allows writing tests in various languages (Java, Python, JavaScript).
- **Espresso**: Android-specific framework by Google, used to write native UI tests for Android apps in Java.

4. How do you use Appium for mobile testing?

- **Setup Appium Server**: Install Appium and start the Appium server.
- **Desired Capabilities**: Set the desired capabilities (e.g., platform name, device name, app path) for the mobile app.
- **Write Test Scripts**: Use Appium's client libraries (Java, Python, etc.) to create test scripts that interact with the mobile app.

5. What is the role of ADB (Android Debug Bridge) in mobile testing?

- **ADB** is a command-line tool used for interacting with Android devices. It allows actions like installing and uninstalling apps, copying files to and from devices, running tests, and more.

Example:

bash
Copy code
adb install app.apk

-

Device and Environment Testing

1. How do you perform testing on different devices and operating systems?

- **Test on Real Devices**: Use a variety of devices (both iOS and Android) to test compatibility across different screen sizes and OS versions.
- **Emulators/Simulators**: Use for initial testing or when real devices are not available.
- **Cloud Platforms**: Use platforms like **BrowserStack** or **Sauce Labs** for testing on a wide variety of devices remotely.

2. What is device fragmentation, and how do you handle it during testing?

- **Device Fragmentation**: Refers to the existence of a large variety of devices with different screen sizes, resolutions, OS versions, and hardware configurations.
- **Handling Fragmentation**: Test on multiple devices (real and emulators), prioritize devices based on user data (top market share devices), and use cloud-based testing platforms.

3. What is the difference between testing on an emulator and a real device?

- **Emulator/Simulator**: Software-based, simulates the device environment. Can't fully replicate device-specific features like GPS, camera, etc.
- **Real Device**: Physical testing on actual devices ensures real-world performance and handles device-specific scenarios like battery consumption, network conditions, and sensors.

4. How do you test a mobile application in a low network environment?

- Use network simulation tools like **Charles Proxy** or **Network Link Conditioner** to simulate low network conditions (e.g., 2G, 3G).
- Test how the app behaves when network connectivity is slow or unavailable.

5. What is battery consumption testing, and how is it performed?

- **Battery Consumption Testing** measures how an app impacts battery life during normal usage.
- Use tools like **Battery Historian** (for Android) to track battery usage and ensure the app does not drain the battery excessively.