Day-5 : Automating Web Apps Using Appium and Implementing a Testing Framework

Introduction to Web Apps Automation Using Appium (30 minutes)

Setting Up Appium for Web App Automation (45 minutes)

Writing Test Scripts for Web Apps (45 minutes)

Implementing a Web App Testing Framework (60 minutes)

Q&A and Recap (30 minutes)

5.1: Introduction to Web Apps Automation Using Appium (30 minutes)

A web application, commonly referred to as a "web app," is a software application that runs on a web server and is accessed through a web browser. Unlike traditional desktop applications, which must be downloaded and installed on a device, web applications are hosted on a network (usually the internet) and can be used across different devices and operating systems as long as a browser is available.

Web applications have become integral to everyday life and business operations, from social media platforms to online banking, shopping, productivity tools, and even complex enterprise systems. The flexibility and accessibility of web apps make them popular in a world that increasingly relies on the internet for communication, transactions, and information sharing.

Definition: Web apps are applications that run in a web browser, designed to work across multiple devices, including desktops, tablets, and mobile phones. Unlike native apps, they do not require installation and can be accessed directly via URLs.

Why Automate Web Apps?:

Automating web apps provides significant benefits to development and QA teams, enabling them to deliver high-quality applications efficiently and consistently across platforms. Here's a breakdown of the key reasons for automating web app testing:

1. Cross-Browser and Cross-Device Testing

Web apps are accessed on various browsers (e.g., Chrome, Firefox, Safari) and devices (desktop, tablet, mobile), each with different configurations.
Automation helps to run test scripts across multiple browsers and devices, ensuring consistent performance and appearance, and uncovering any browser-specific issues early.

2. Repetitive Task Automation

Some test cases, like login processes, form submissions, and data validations, are repetitive but essential.
Automating these repetitive tasks saves time and allows testers to focus on more complex and high-value test scenarios, improving overall productivity.

3. Improved Accuracy and Reliability

Manual testing is prone to human error, especially in complex or repetitive tasks.
Automated tests run the same way each time, providing reliable, consistent results and reducing the chances of missing critical issues.

4. Faster Feedback Loop

Automation accelerates the feedback cycle, allowing teams to run tests continuously, especially in CI/CD pipelines, and receive immediate feedback on code changes.
This fast feedback helps teams address issues early, reducing the cost and time of fixing bugs later in the development cycle.

5. Enhanced Test Coverage

Automated testing enables QA teams to test a broader range of scenarios that would be time-consuming and costly to perform manually.
It allows teams to automate both functional (UI) and non-functional (performance, load) testing, ensuring all aspects of the web app are thoroughly validated.

6. Scalability in Testing

Automated tests can be scaled up to run parallel tests across different environments, devices, and browsers.
This scalability is crucial for web apps that expect high traffic or diverse usage scenarios, enabling testers to simulate various conditions and verify the app's robustness.

7. Regression Testing Made Easy

Every time new features are added or code is updated, regression tests are necessary to ensure existing functionality isn't broken.
Automation simplifies regression testing, allowing for rapid execution of test suites to validate that the app remains stable after changes.

8. Cost-Effectiveness in the Long Run

While the initial setup of automated testing requires investment in time and tools, automation saves money and effort in the long run by reducing manual labour.
With automated test scripts, teams can achieve high test execution frequency without additional resources, making it highly cost-effective over time.

Browser Support for web apps

Benefits of web apps

Part 2: Setting Up Appium for Web App Automation (45 minutes)

Objective: Learn to set up Appium for automating web apps.
Content:

Prerequisites:

Java, Node.js, Appium Desktop, Chrome Driver (for Android), and Safari Driver (for iOS).

Installation Steps:

```
npm install -g appium
npm install -g appium-doctor
appium-doctor --android
appium-doctor --ios
appium
```

Configuring Desired Capabilities:

Example configuration for Chrome browser on Android:

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("deviceName", "Android Emulator");
caps.setCapability("platformName", "Android");
caps.setCapability("browserName", "Chrome");
```

Part 3: Writing Test Scripts for Web Apps (45 minutes)

Objective: Learn to write and run test scripts for web apps using Appium.

Content:

Writing Test Scripts:

Importing necessary libraries and initializing the driver:

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

public class WebAppTest {
    public static void main(String[] args) throws Exception {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setCapability("deviceName", "Android Emulator");
```

```
        caps.setCapability("platformName", "Android");
        caps.setCapability("browserName", "Chrome");

        AndroidDriver<MobileElement> driver = new AndroidDriver<>(new
URL("http://localhost:4723/wd/hub"), caps);

        driver.get("https://www.google.com");
        MobileElement searchBox = driver.findElementByName("q");
        searchBox.sendKeys("Appium testing");
        searchBox.submit();
        driver.quit();
    }
}
```

Using different locators for web elements:

By ID, By Name, By Class Name, By XPath, By CSS Selector.

Example:

```
MobileElement searchBox = driver.findElementByName("q");
searchBox.sendKeys("Appium testing");
```

Part 4: Implementing a Web App Testing Framework (60 minutes)

Content:

Framework Components:

Test Organization: Structure for organizing test cases and test data.

Page Object Model (POM): Design pattern to create object repositories for web elements.

TestNG: Testing framework for running tests and generating reports.

Utility Classes: Common utility methods for reuse across test scripts.

Setting Up the Framework:

Directory Structure:

```
src/test/java/
  └── pages/
  └── tests/
  └── utils/
```

Example of a Page Object Model class:

```
package pages;

import io.appium.java_client.MobileElement;
import io.appium.java_client.pagefactory.AndroidFindBy;
import io.appium.java_client.pagefactory.AppiumFieldDecorator;
import org.openqa.selenium.support.PageFactory;

public class GoogleHomePage {
    public GoogleHomePage(AndroidDriver<MobileElement> driver) {
        PageFactory.initElements(new AppiumFieldDecorator(driver), this);
    }

    @AndroidFindBy(name = "q")
    private MobileElement searchBox;

    public void search(String query) {
        searchBox.sendKeys(query);
        searchBox.submit();
    }
}
```

Example of a TestNG test class:

```
package tests;

import io.appium.java_client.MobileElement;
import io.appium.java_client.android.AndroidDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;
import pages.GoogleHomePage;

import java.net.URL;

public class GoogleSearchTest {
    private AndroidDriver<MobileElement> driver;
    private GoogleHomePage googleHomePage;

    @BeforeClass
    public void setUp() throws Exception {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setCapability("deviceName", "Android Emulator");
        caps.setCapability("platformName", "Android");
        caps.setCapability("browserName", "Chrome");
```

```
        driver = new AndroidDriver<>(new URL("http://localhost:4723/wd/hub"), caps);
        googleHomePage = new GoogleHomePage(driver);
    }

    @Test
    public void testSearch() {
        driver.get("https://www.google.com");
        googleHomePage.search("Appium testing");
    }

    @AfterClass
    public void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```

Part 5: Q&A and Recap (30 minutes)

Recap of key points:

- Basics of web app automation
- Setting up Appium for web app automation
- Writing and running test scripts for web apps
- Implementing a testing framework for web apps