Day4-Automating Hybrid Apps Using Appium

Introduction to Hybrid Apps (30 minutes)

Setting Up Appium for Hybrid App Automation (60 minutes)

Q&A and Recap (30 minutes)

4.1 Introduction to Hybrid Apps(30 minutes)

Definition-

A hybrid app is a mobile application that combines elements of both native apps (built for a specific platform, such as iOS or Android) and web apps (built with web technologies like HTML, CSS, and JavaScript). Hybrid apps are designed to work on multiple platforms and devices, including smartphones and tablets

Hybrid apps are beneficial because they offer a cost-effective and efficient way for businesses to reach a wider audience. They can be developed using a single code base and can run on multiple platforms, which saves time and money compared to developing separate native apps for each platform.

Architecture of Hybrid Apps

Architecture: Hybrid apps have a clean architecture with a data layer, business layer, and presentation layer.
Front end: Hybrid apps use front-end technologies like HTML, CSS, and JavaScript.
Back end: Hybrid apps use native applications as the foundation for the back end.
Plugins: Hybrid apps use plugins like Ionic Capacitor or Apache Cordova to access native platform features.
Browser: Once installed, the app's native shell connects to the mobile platform's capabilities through a browser embedded in the app

Advantages and Disadvantges

Advantages of Hybrid Apps:

● Cross-platform compatibility: Hybrid apps can be developed to work on multiple platforms, which means that businesses can reach a wider audience with a single app.

● Cost-Effective: Developing a hybrid app can be less expensive than developing separate native apps for each platform.

● Faster Development Time: Since hybrid apps use a single code base, they can be developed faster than native apps.

● Easy To Maintain: Hybrid apps are easier to maintain since updates and changes can be made to a single code base.

● Access To Device Features: Hybrid apps can access device features like the camera, GPS, and accelerometer, which allows for a more engaging user experience.

Disadvantages of Hybrid Apps:

● Performance Issues: Hybrid apps can sometimes suffer from performance issues, especially when compared to native apps.

● Limited Access to Device Features: While hybrid apps can access device features, they may not have the same level of access as native apps.

● User Experience: Hybrid apps may not offer the same level of user experience as native apps, especially when it comes to animations and other graphics-intensive features.

● Development Limitations: Hybrid apps may have limitations when it comes to developing more complex or specialized features

Examples :-

Popular Hybrid Apps and Their Use Cases

Hybrid apps are increasingly popular due to their cost-effectiveness and cross-platform compatibility. They use a single codebase that runs across both iOS and Android, making them efficient to develop and maintain. Here are some popular hybrid apps, along with their use cases:

---

1. Instagram

Use Case: Social media platform for sharing photos, videos, and live streaming.
Why Hybrid?: Instagram uses hybrid technology for easy management of its interface across platforms. Its seamless performance allows for dynamic content like Stories, Reels, and direct messaging. By using hybrid technology, Instagram can quickly implement updates across devices, ensuring a consistent user experience.

2. Uber

Use Case: Ride-hailing app connecting drivers and riders.
Why Hybrid?: Uber's driver-side app is hybrid, enabling quick updates and maintenance across different devices that drivers may use. The hybrid approach also ensures that the app is lightweight, essential for quick loading and reliability, which is critical for real-time navigation and trip scheduling.

3. Twitter

Use Case: Social media platform for sharing short messages (tweets), photos, and videos.
Why Hybrid?: Twitter employs a hybrid structure to push updates quickly across its user base and ensure a consistent experience regardless of platform. Hybrid technology allows Twitter to retain its design and functionality consistency across different operating systems and devices.

4. Evernote

Use Case: Note-taking and task management app.
Why Hybrid?: Evernote uses hybrid technology to deliver a unified experience, allowing users to manage notes, to-do lists, and projects across mobile, desktop, and web platforms. Hybrid frameworks allow Evernote to provide real-time synchronization and consistent performance, making it convenient for users who switch between devices.

5. Amazon App Store

Use Case: E-commerce marketplace for browsing and purchasing products.
Why Hybrid?: Amazon's App Store leverages hybrid technology to offer a fast, consistent user experience, especially for handling dynamic content such as recommendations and product listings. This approach is cost-effective for Amazon, as it allows updates to be rolled out quickly while maintaining the app's core features across platforms.

6. Gmail (WebView Elements)

Use Case: Email service and communication tool.
Why Hybrid?: While Gmail is largely native, certain elements, particularly the email content display, leverage WebView for loading emails in a web-like interface. This hybrid approach allows for a familiar, browser-like experience for reading and formatting emails within the app.

Frameworks (Hybrid App Frameworks)

Definition: Frameworks like Apache Cordova, Ionic, React Native, and Capacitor help streamline hybrid app development.
Function: They provide a development environment, support for plugins, and enable cross-platform compatibility by creating build files for both Android and iOS.
Usage: Hybrid frameworks simplify the development and packaging process, allowing developers to use one codebase to create and distribute apps across multiple platforms.

4.2 Setting Up Appium for Hybrid App Automation (60 minutes)

Content:

Prerequisites:

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

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

Configuring Desired Capabilities:

Example configuration for Android:

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("deviceName", "Android Emulator");
caps.setCapability("platformName", "Android");
caps.setCapability("app", "/path/to/hybrid_app.apk");
caps.setCapability("chromedriverExecutable", "/path/to/chromedriver");
```

Part 3: Writing Test Scripts for Hybrid Apps (60 minutes)

Content:

Switching Contexts in Hybrid Apps:

Understanding Contexts: Native and WebView contexts.

Switching between contexts:

```
Set<String> contextHandles = driver.getContextHandles();
for (String context : contextHandles) {
    System.out.println(context);
}
driver.context("WEBVIEW_com.example.hybridapp");
```

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 HybridAppTest {
    public static void main(String[] args) throws Exception {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setCapability("deviceName", "Android Emulator");
        caps.setCapability("platformName", "Android");
        caps.setCapability("app", "/path/to/hybrid_app.apk");
        caps.setCapability("chromedriverExecutable", "/path/to/chromedriver");

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

        Set<String> contextHandles = driver.getContextHandles();
        for (String context : contextHandles) {
            if (context.contains("WEBVIEW")) {
                driver.context(context);
                break;
```

```
            }
        }

        // Test script
        MobileElement el = driver.findElementByXPath("//input[@name='username']");
        el.sendKeys("testuser");
        driver.quit();
    }
}
```

Using different locators for web elements:

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

Importance of Switching Contexts in Hybrid Apps

In hybrid app automation testing, context switching is a critical operation that allows testers to interact with both native and web elements within a hybrid app. Hybrid apps often contain WebView components embedded within a native shell. For automation tools like Appium, switching contexts enables the tester to move between interacting with native elements (buttons, navigation bars) and web elements (HTML content within WebView) seamlessly.

Part 4: Advanced Automation Techniques for Hybrid Apps (45 minutes)

Content:

Handling Dynamic Elements:

Using XPath functions and CSS selectors to locate dynamic elements.

```
MobileElement element = driver.findElementByXPath("//input[contains(@name, 'user')]");
element.sendKeys("testuser");
```

Handling Multiple Windows and Tabs:

Switching between multiple windows and tabs in WebView

```
Set<String> windowHandles = driver.getWindowHandles();
for (String handle : windowHandles) {
driver.switchTo().window(handle);
}
```

Handling Alerts and Pop-ups:

Switching to and interacting with alerts and pop-ups:

```
Alert alert = driver.switchTo().alert();
alert.accept();
```

Debugging and Troubleshooting:

Common issues: Element not found, context switching issues, synchronization problems.

Solutions: Use explicit waits, verify context switches, check locator strategies.

Example of using explicit waits:

```
javaCopy code
WebDriverWait wait = new WebDriverWait(driver, 10);
MobileElement element =
wait.until(ExpectedConditions.visibilityOfElementLocated([By.id](http://by.id/)("com.example:i
d/button")));
element.click();
```

Advanced automation technique for hybrid app

Advanced Automation Techniques for Hybrid Apps

Handling Dynamic Elements:

Using XPath functions and CSS selectors to locate dynamic elements.

Handling Multiple Windows and Tabs:

Switching between multiple windows and tabs in WebView.

Handling Alerts and Pop-ups:

Switching to and interacting with alerts and pop-ups.

Debugging and Troubleshooting:

Common issues: Element not found, context switching issues, synchronization problems.

Solutions: Use explicit waits, verify context switches, check locator strategies.

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

Objective: Review the key concepts covered in the session and address any questions.
Content:

- Recap of key points:
  - Introduction to hybrid apps
  - Setting up Appium for hybrid app automation
  - Writing and running test scripts for hybrid apps
  - Advanced automation techniques for hybrid apps

Summary

This session plan provides a comprehensive understanding of automating hybrid apps using Appium. It covers the setup and configuration required for hybrid app automation, writing and running test scripts, and advanced automation techniques. The combination of theoretical knowledge, practical examples, and hands-on activities ensures that students gain a solid foundation in using Appium for hybrid app testing.