Day-3 Native Apps and Web Apps Automation Using Appium

Introduction to Native Apps and Web Apps (30 minutes)

Native Apps Automation Using Appium (75 minutes)

Web Apps Automation Using Appium (75 minutes)

Q&A and Recap (30 minutes)

**3.1 Introduction to Native Apps and Web Apps (30 minutes

Analogy-**

Favorite Shoes: A native app is like your favorite pair of shoes that fit your feet perfectly. They're made just for your size and feel super comfy, so you can run and play without any trouble.

A native app is a software application developed specifically for a particular operating system (OS) and platform, such as iOS, Android, or Windows. Native apps are built using the official programming languages and tools of that platform, allowing them to take full advantage of the device's features and capabilities.

Characteristics of Native Apps:

1. Platform-Specific Languages and Tools:
   iOS: Developed using Swift or Objective-C in Xcode.
   Android: Developed using Kotlin or Java in Android Studio.
   Windows: Developed using C# or C++ with Visual Studio.
2. High Performance:
   Optimized for the specific OS, leading to faster loading times, smoother animations, and better responsiveness.
3. Direct Access to Device Features:
   Full access to the device's hardware (e.g., camera, GPS, microphone, sensors) and software features (e.g., notifications, contact lists), which enhances functionality and interactivity.
4. Consistent User Experience:

Native apps follow platform-specific design guidelines (such as Apple's Human Interface Guidelines or Google’s Material Design) to provide a familiar and seamless user experience.

5. Offline Functionality:
   Can store data locally, allowing parts of the app to function even when there is no internet connection.
6. Enhanced Security:
   Native apps can leverage built-in security features, such as biometric authentication and platform-specific encryption, to protect user data.
7. Regular Updates and Maintenance:
   Since they’re developed for a single platform, native apps are easier to update and optimize for new OS versions or hardware features.

**Web apps

Analogy-**

Library Book: A web app is like a library book you borrow and read wherever you are. You don’t need to own it—it stays in the library (the internet), and you can use it whenever you visit.

A web app is an application accessed and used via a web browser over the internet. Unlike native apps, web apps are not platform-specific and run on any device with a compatible browser, making them cross-platform by default. Web apps are typically built with web technologies like HTML, CSS, and JavaScript and can mimic the look and functionality of native apps to some extent.

Characteristics of Web Apps:

1. Cross-Platform Compatibility:
   Web apps can run on any device or operating system with a modern browser, including desktops, tablets, and smartphones.
2. No Installation Required:
   Users don’t need to download or install web apps; they’re accessed through a URL. This reduces storage usage and simplifies user access.
3. Easier Updates:
   Updates are made on the server side, meaning all users automatically have the latest version without needing to update manually.
4. Cost-Effective Development:
   Since they’re developed once and can work across multiple platforms, web apps are typically less expensive and faster to build than native apps.
5. Dependency on Internet Connection:
   Web apps rely on an internet connection for most functionalities, though some features can work offline with advanced caching (e.g., Progressive Web Apps).
6. Reduced Access to Device Features:

Web apps have limited access to device features like GPS, camera, or Bluetooth, although Progressive Web Apps (PWAs) can use some hardware capabilities.

7. Responsive Design:
   Built to adapt to different screen sizes and orientations, ensuring a consistent user experience across various devices.
8. Scalability:
   Web apps are easily scalable because they're centrally hosted and can accommodate large numbers of users as server capacity is increased.

Hybrid apps

Analogy→

Lunchbox with Favorite Snacks: A hybrid app is like a lunchbox that has some of your favorite snacks from home (native app features) and some snacks you get from school (web app features). It combines the best of both!

A hybrid app is a software application that combines elements of both native apps and web apps. Hybrid apps are developed using web technologies like HTML, CSS, and JavaScript but are wrapped in a native container, allowing them to be installed on devices and distributed through app stores (similar to native apps). They rely on frameworks like Apache Cordova, Ionic, and React Native to access native device features.

Characteristics of Hybrid Apps:

1. Cross-Platform Compatibility:
   A single codebase can be used across multiple platforms (iOS, Android, etc.), reducing development time and costs.
2. Web Technologies in a Native Shell:
   Built with web technologies (HTML, CSS, JavaScript) and packaged within a native container, allowing them to be distributed through app stores like native apps.
3. Access to Device Features:
   Hybrid apps have more access to device features (camera, GPS, contacts, etc.) than web apps through plugins, though this access may be somewhat limited compared to purely native apps.
4. Offline Functionality:
   Hybrid apps can store data locally within the device, enabling offline functionality for certain features, depending on the design and requirements.
5. Easier Maintenance and Updates:
   Since they're built on a single codebase for multiple platforms, updates and bug fixes can be applied universally, streamlining the maintenance process.
6. Performance Limitations:

Hybrid apps are typically slower than fully native apps because they rely on a web view to render content, but performance can be optimized using frameworks like React Native or Flutter, which enable more native-like experiences.

7. Access to App Stores:
   Like native apps, hybrid apps can be distributed through the Apple App Store, Google Play Store, and other app marketplaces, allowing users to download and install them directly.
8. Enhanced UI/UX Capabilities:
   While hybrid apps can mimic native app designs, achieving a fully native look and feel can be challenging, but modern frameworks help narrow the gap significantly.

Examples of native app

Popular Native Apps and Their Use Cases

1. Instagram:
   Platform: iOS and Android
   Use Case: Social media, photo and video sharing, messaging.
   Why Native: Instagram utilizes native app features like the camera, push notifications, and smooth UI transitions to deliver a highly interactive, media-rich experience.
2. Spotify:
   Platform: iOS, Android
   Use Case: Streaming music, creating playlists, downloading music for offline listening.
   Why Native: Spotify offers seamless audio playback, advanced caching, and offline functionality, benefiting from the optimized performance and audio features native development provides.
3. WhatsApp:
   Use Case: Instant messaging, voice and video calling, media sharing.
   Why Native: WhatsApp needs to provide secure, real-time messaging and calls while integrating with contacts, storage, and notifications for a smooth, reliable user experience.
   Platform: iOS, Android
4. Uber:
   Platform: iOS, Android
   Use Case: Ridesharing, location-based services, real-time notifications.
   Why Native: Uber requires precise GPS tracking, real-time updates, and payment integration, all of which benefit from native capabilities.

Popular Web Apps and Their Use Cases

1. Google Docs:
   Use Case: Document creation, editing, and collaboration.
   Why Web App: Users can access and edit documents from any browser without installation, making it easy to collaborate and work across devices.
2. Trello:
   Use Case: Project management, task tracking, team collaboration.
   Why Web App: Trello allows easy access on any device with real-time updates, ideal for remote teams who need a lightweight, cross-platform solution.
3. Netflix (Web Version):
   Use Case: Streaming movies and TV shows.
   Why Web App: Netflix's web app provides flexibility for users who want to stream on any device without downloading, ideal for users on shared or public computers.
4. Canva:
   Use Case: Graphic design, photo editing.
   Why Web App: Canva allows users to create, store, and access designs from any browser, making it easy for casual designers and businesses to create media on the go.

Choosing Between Native and Web Apps:

Native apps are preferred when high performance, access to device features, and offline functionality are critical.
Web apps are ideal for users who need platform flexibility, ease of access, and lightweight, non-processor-intensive applications.

Each app type has strengths depending on use cases, user expectations, and development resources.

3.2 Native Apps Automation Using Appium

Setting Up Appium for Native App Automation:

Prerequisites: Java, Android SDK, Node.js, Appium Desktop.

Installation steps:

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

Configuring desired capabilities:

```
DesiredCapabilities dc = new DesiredCapabilities();

               dc.setCapability(MobileCapabilityType.DEVICE_NAME, "AD");

               dc.setCapability(MobileCapabilityType.PLATFORM_NAME, "Android");

               dc.setCapability(AndroidMobileCapabilityType.APP_PACKAGE,
"io.appium.android.apis");

               dc.setCapability(AndroidMobileCapabilityType.APP_ACTIVITY,
"io.appium.android.apis.ApiDemos");


               AndroidDriver<AndroidElement> driver=new AndroidDriver<AndroidElement>(new
URL("http://0.0.0.0:4723/wd/hub") , dc);

               return driver;
```

Writing Test Scripts for Native Apps:

```
package nativeapp;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import javax.lang.model.element.Element;

import org.openqa.selenium.WebElement;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
import static java.time.Duration.ofSeconds;
import com.github.dockerjava.api.model.Driver;

import io.appium.java_client.MobileBy;
import io.appium.java_client.TouchAction;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import io.appium.java_client.android.nativekey.AndroidKey;
import io.appium.java_client.android.nativekey.KeyEvent;
import io.appium.java_client.touch.LongPressOptions;

import io.appium.java_client.touch.offset.ElementOption;

// static import for longpress
import  static io.appium.java_client.touch.LongPressOptions.longPressOptions;

import static io.appium.java_client.touch.offset.ElementOption.element;

public class Apidemo extends Apidemos {

       AndroidDriver<AndroidElement> driver;
```

```
	@BeforeTest

	public void bt() throws MalformedURLException {
		driver= cap();

		driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);


	}
	@Test(priority = 1)

	public void tc() throws InterruptedException {
		System.out.println("API demos opened");
		driver.findElement(MobileBy.AccessibilityId("Accessibility")).click();

		driver.findElement(MobileBy.AccessibilityId("Accessibility Node
Querying")).click();
		driver.pressKey(new KeyEvent(AndroidKey.BACK));
		Thread.sleep(2000);
		driver.pressKey(new KeyEvent(AndroidKey.BACK));
		//driver.pressKey(new KeyEvent(AndroidKey.BACK));

	}

	@Test(priority = 2)
	public void performance() throws InterruptedException {

		driver.findElement(MobileBy.AccessibilityId("Preference")).click();

		driver.findElement(MobileBy.AccessibilityId("3. Preference
dependencies")).click();

		driver.findElement(MobileBy.id("android:id/checkbox")).click();


		driver.findElement(MobileBy.AndroidUIAutomator("UiSelector().text(\"WiFi
settings\")")).click();

		driver.findElement(MobileBy.id("android:id/edit")).sendKeys("vaishnavi");

	driver.hideKeyboard();

		driver.findElement(MobileBy.id("android:id/button2")).click();

		//this is for going back

		driver.pressKey(new KeyEvent(AndroidKey.BACK));

		Thread.sleep(5000);

		driver.pressKey(new KeyEvent(AndroidKey.BACK));
		//driver.pressKey(new KeyEvent(AndroidKey.BACK));


	}
	@Test(priority = 3)

	public void notification() {
		driver.openNotifications();

driver.findElements(MobileBy.className("android.widget.ImageView")).get(4).click();


driver.findElements(MobileBy.className("android.widget.ImageView")).get(4).click();
		driver.pressKey(new KeyEvent(AndroidKey.BACK));

		//driver.pressKey(new  KeyEvent(AndroidKey.HOME));


	}
	@Test(priority = 4)

	public void scroll() {
		driver.findElement(MobileBy.AccessibilityId("Views")).click();
		driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new
UiSelector()).scrollIntoView(text(\"WebView\"))")).click();
		driver.pressKey(new KeyEvent(AndroidKey.BACK));
	}

	@Test(priority = 5)

	public void longpress() throws InterruptedException {
		//driver.findElement(MobileBy.AccessibilityId("Views")).click();
		driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new
UiSelector()).scrollIntoView(text(\"Expandable Lists\"))")).click();

		//driver.findElement(MobileBy.AccessibilityId("Expandable Lists")).click();

		driver.findElement(MobileBy.AccessibilityId("1. Custom Adapter")).click();

		AndroidElement cat =
driver.findElement(MobileBy.AndroidUIAutomator("UiSelector().text(\"Cat Names\")"));

		TouchAction tc= new TouchAction(driver);


tc.longPress(longPressOptions().withElement(element(cat)).withDuration(ofSeconds(3))).release(
).perform();
		driver.pressKey(new KeyEvent(AndroidKey.BACK));
		Thread.sleep(3000);
		driver.pressKey(new KeyEvent(AndroidKey.BACK));

	}
	@Test(priority = 6)
```

```
        public void swipe() {

                driver.findElement(MobileBy.AccessibilityId("Views")).click();

                driver.findElement(MobileBy.AccessibilityId("Date Widgets")).click();

                driver.findElement(MobileBy.AccessibilityId("2. Inline")).click();

                AndroidElement ele1 = driver.findElement(MobileBy.AccessibilityId("12"));

                AndroidElement ele2 = driver.findElement(MobileBy.AccessibilityId("5"));

                TouchAction tc= new TouchAction(driver);


tc.longPress(longPressOptions().withElement(element(ele1)).withDuration(ofSeconds(3))).moveTo(
element(ele2)).release().perform();

        }

        @Test(priority = 7)
        public void switchapp() throws InterruptedException {
                driver.findElement(MobileBy.AccessibilityId("OS")).click();
                driver.findElement(MobileBy.AccessibilityId("SMS Messaging")).click();
                Thread.sleep(2000);
                //sdriver.findElement(MobileBy.AccessibilityId("Enable SMS broadcast
receiver")).click();


driver.findElement(MobileBy.id("io.appium.android.apis:id/sms_recipient")).sendKeys("(650)
555-1212");


driver.findElement(MobileBy.id("io.appium.android.apis:id/sms_content")).sendKeys("Hello
world..");

                //driver.hideKeyboard();

                driver.findElement(MobileBy.AccessibilityId("Send")).click();



                driver.activateApp("com.google.android.apps.messaging");
                Thread.sleep(5000);

                String msg =
driver.findElement(MobileBy.id("com.google.android.apps.messaging:id/conversation_snippet")).g
etText();

                System.out.println(msg);

        }
```

```
}
```

Using different locators:

By ID, By Name, By Class Name, By XPath, By Accessibility ID, By Android UIAutomator.

Example:

```
MobileElement elById = driver.findElementById("com.example:id/button");
elById.click();
```

Advanced Native App Automation Techniques:

Handling gestures (tap, swipe, scroll).

```
driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new
UiSelector()).scrollIntoView(text(\"Expandable Lists\"))")).click();

              //driver.findElement(MobileBy.AccessibilityId("Expandable Lists")).click();

              driver.findElement(MobileBy.AccessibilityId("1. Custom Adapter")).click();

              AndroidElement cat =
driver.findElement(MobileBy.AndroidUIAutomator("UiSelector().text(\"Cat Names\")"));

              TouchAction tc= new TouchAction(driver);


tc.longPress(longPressOptions().withElement(element(cat)).withDuration(ofSeconds(3))).release(
).perform();
              driver.pressKey(new KeyEvent(AndroidKey.BACK));
              Thread.sleep(3000);
              driver.pressKey(new KeyEvent(AndroidKey.BACK));
```

Interacting with device features (camera, GPS).

```
driver.openNotifications();
```

3.3 Web Apps Automation Using Appium (75 minutes)

Setting Up Appium for Web App Automation:

- Prerequisites: Java, Android SDK, Node.js, Appium Desktop.
- Configuring desired capabilities:

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

- 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 {
        driver.get("https://www.google.com/");

driver.findElement(MobileBy.cssSelector("[class=\"gLFyf\"]")).sendKeys("Masai",Keys.ENTER);
               driver.navigate().back();
               driver.navigate().forward();
               driver.pressKey(new KeyEvent(AndroidKey.HOME));
               driver.findElement(MobileBy.AccessibilityId("OS")).click();
               driver.findElement(MobileBy.AccessibilityId("SMS Messaging")).click();
               driver.findElement(MobileBy.AccessibilityId("Enable SMS broadcast
receiver")).click();

driver.findElement(By.id("io.appium.android.apis:id/sms_recipient")).sendKeys("(650)
555-1212");
               driver.findElement(By.id("io.appium.android.apis:id/sms_content")).sendKeys("Hi
everyone");
               driver.hideKeyboard();
               driver.findElement(MobileBy.AccessibilityId("Send")).click();

    }
}
```

Using Different Locators for Web Apps:

- By ID, By Name, By Class Name, By XPath, By CSS Selector.
- Example:

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

Advanced Web App Automation Techniques:
Handling multile windows/tabs

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

Summary

This session plan provides a comprehensive understanding of automating native apps and web apps using Appium. It covers the setup and configuration required for both types of apps, writing 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 mobile application testing.