Group_1:

Hrushikesh Pawar
Anupam Pandey
HarshWardhan Sharma
Manish Singh
Syed Mohammed Tahir
Krishna Zanwar
Asha Julupalli

Appium supports the automation of native, hybrid, and web applications on various platforms (iOS, Android, Windows) using a single API. This allows for a unified approach to testing across different types of applications. Here's how Appium supports each type of application:

1. Native Applications

Definition: Native apps are built specifically for a particular platform using platform-specific programming languages and SDKs (e.g., Swift or Objective-C for iOS, Java or Kotlin for Android).

Appium Support:

iOS: Appium uses the XCUITest framework (previously UIAutomation) to automate iOS applications.

Android: Appium uses the UIAutomator/UiAutomator2 or Espresso frameworks to automate Android applications.

Capabilities:

Access to all native UI elements.

Interaction with device features like GPS, camera, and notifications.

2. Hybrid Applications

Definition: Hybrid apps are built using web technologies (HTML, CSS, JavaScript) and are wrapped in a native container, allowing them to be installed on a mobile device. They use WebView to render web content.

Appium Support:

Context Switching: Appium can switch between native and web contexts within the same app.

Access to WebView: By switching to the WebView context, Appium can automate interactions within the web component of the app.

Capabilities:

Access to both native and web elements.

Ability to interact with the DOM and execute JavaScript.

3. Web Applications

Definition: Web apps are accessed through mobile browsers and do not need to be installed on the device.

Appium Support:

Mobile Browsers: Appium can automate mobile browsers like Chrome on Android and Safari on iOS.

Capabilities:

Access to web elements in the browser.

Interaction with web pages similar to Selenium.

Common Features

Cross-Platform Testing: Use the same test script for both iOS and Android platforms by abstracting the platform-specific details.

Multi-Language Support: Appium supports various programming languages like Java, Python, Ruby, JavaScript, and C#.

WebDriver Protocol: Appium uses the WebDriver protocol to interact with mobile applications, making it similar to Selenium for web automation.

Inspector Tools: Appium provides inspector tools to identify elements within the app, aiding in the creation of automation scripts.

Native Apps: Full access to native UI elements and device features using platform-specific automation frameworks (XCUITest for iOS, UIAutomator/Espresso for Android).

Hybrid Apps: Ability to switch contexts between native and web views, allowing interaction with both types of elements.

Web Apps: Automation of mobile web browsers using the same techniques as Selenium, enabling interaction with web elements in the browser.

Appium's versatility and cross-platform capabilities make it a powerful tool for mobile application automation, providing comprehensive support for native, hybrid, and web applications.

Programming Example Using Java :

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

public class AppiumTest {
  public static void main(String[] args) {
    // Set the Desired Capabilities
    DesiredCapabilities caps = new DesiredCapabilities();
    caps.setCapability("deviceName", "Android Emulator");
    caps.setCapability("platformName", "Android");
    caps.setCapability("appPackage", "com.example.myapp");
    caps.setCapability("appActivity", "com.example.myapp.MainActivity");
    caps.setCapability("automationName", "UiAutomator2");

    // Initialize the Android Driver
    AndroidDriver<MobileElement> driver = null;
    try {
      driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), caps);
    } catch (MalformedURLException e) {
      System.out.println("Invalid URL for Appium Server");
      e.printStackTrace();
    }
```

```
        // Add a small wait to ensure the app has loaded
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Find an element and interact with it
        MobileElement myButton = driver.findElementById("com.example.myapp:id/button");
        myButton.click();

        // Validate the result
        MobileElement resultText = driver.findElementById("com.example.myapp:id/result");
        if (resultText.getText().equals("Expected Result")) {
            System.out.println("Test Passed!");
        } else {
            System.out.println("Test Failed!");
        }

        // Close the app
        if (driver != null) {
            driver.quit();
        }
    }
}
```

This example demonstrates:

Setting Up Desired Capabilities: Configures the test environment, such as the device name, platform, app package, and activity.

Initializing the Driver: Connects to the Appium server and starts the session.

Interacting with the App: Locates UI elements and performs actions (like clicking a button).

Validating the Result: Checks if the app behavior is as expected.

Closing the App: Ends the session and closes the app.

Thank You