Trella Test Automation Challenge

Part 2: Mobile Automation Case Study

Requirement 1

Recommended Testing Tool/Technology

Given the application is Android-only, there are several tools suitable for implementing a test automation framework:

1. **Appium**
   - Supports Android, iOS, and Windows.
   - Cross-platform: Write tests using the same APIs for multiple platforms.
   - Supports multiple programming languages: Java, Python, Ruby, C#, etc.
2. **Espresso**
   - Designed for Android UI testing.
   - Part of Android Jetpack.
   - Lightweight and fast for native apps.
3. **MonkeyRunner**
   - An Android-specific tool for UI automation.
4. **LambdaTest**
   - A cross-platform mobile app testing tool.

Given the application is Android-only, we will focus on comparing **Appium** and **Espresso**.

Criteria for Selecting an Automation Tool

Key factors to consider:

1. Platform Support
2. Application Type
3. Programming Language Support
4. Ease of Integration
5. Speed and Performance
6. Ease of Setup and Use
7. Community and Support
8. Device Testing Options
9. Cost
10. Reporting and Analytics

Appium

- **Cross-Platform Support**: Facilitates testing across multiple platforms, including Android and iOS, enabling code reuse.
- **Multiple Programming Languages**: Supports Java, Python, Ruby, C#, and more.
- **No Need for App Modification**: Does not require access to the application's source code, making it suitable for black-box testing.
- **Slower Execution**: The client-server architecture may result in slower test execution compared to Espresso.

Espresso

- **Android-Specific**: Tailored exclusively for Android applications, with deep integration into the Android SDK.
- **Faster Execution**: Tight integration with Android Studio ensures quicker execution without server communication.
- **Limited Language Support**: Primarily supports Java and Kotlin, which may limit testers familiar with other languages.
- **Requires Access to Source Code**: Ideal for white-box testing but necessitates source code access.

Choosing Between Appium and Espresso

- **Use Appium if:**
  - You need to test across both Android and iOS platforms.
  - Your team uses diverse programming languages.
  - The application’s source code is unavailable.
- **Use Espresso if:**
  - Testing is focused solely on Android.
  - You prioritize faster execution and stability.
  - Your team is proficient in Java or Kotlin.
  - Source code access is available.

Since the application is designed exclusively for Android users, **Espresso** is the recommended tool.

Requirement 2

Automating the Uploading POD Scenario

Yes, the scenario can be automated. However, it involves a step where the camera opens, the user takes a photo, and uploads it. **Espresso** does not directly support camera interactions but provides workarounds for simulating or automating this process.

Approaches

1. **Mocking the Camera**
2. **Using Intents to Simulate Camera Behavior**
3. **Using UIAutomator for Camera Interaction**

Steps for Automation

1. **Launch the App**
   - Ensure the app launches successfully and the login screen is displayed.
2. **Log in to the App**
   - Locate the Mobile Number field and enter valid credentials.
   - Locate the Password field and enter valid credentials.
   - Click the Login button.
3. **Grant Permissions**
   - Allow the application to access location services.
4. **Navigate to My Loads**
   - Click on "My Loads" in the navigation menu.
5. **Select the Past Shipment Tab**
   - Scroll through the list of shipments.
   - Select the relevant shipment marked as "PODs not uploaded."
6. **Upload the Proof of Delivery (POD)**
   - Locate the "Documents" button.
   - Simulate selecting or uploading a file by capturing an image of the document.
   - Confirm and upload the POD document by clicking the "Submit" button.
7. **Verify Upload Success**
   - Check the "Documents" section and verify that the file appears successfully.
   - Confirm that the shipment status updates in the Past Loads tab.