Native Mobile Automation Lite Framework

Solution Accelerator

[DATE]

Objective

In mobile application development, most of the applications are developed in multiple platforms, commonly on iOS and Android. When developers try to automate the mobile application functionality they need to write the same automation script for each platform. It leads to the code duplication. The code duplication may also be for the features like -- taking screenshots, video recording and logs capturing. With this duplication, any change in the application will affect the automation code in multiple locations, as developers need to update code separately for all the supported platforms.

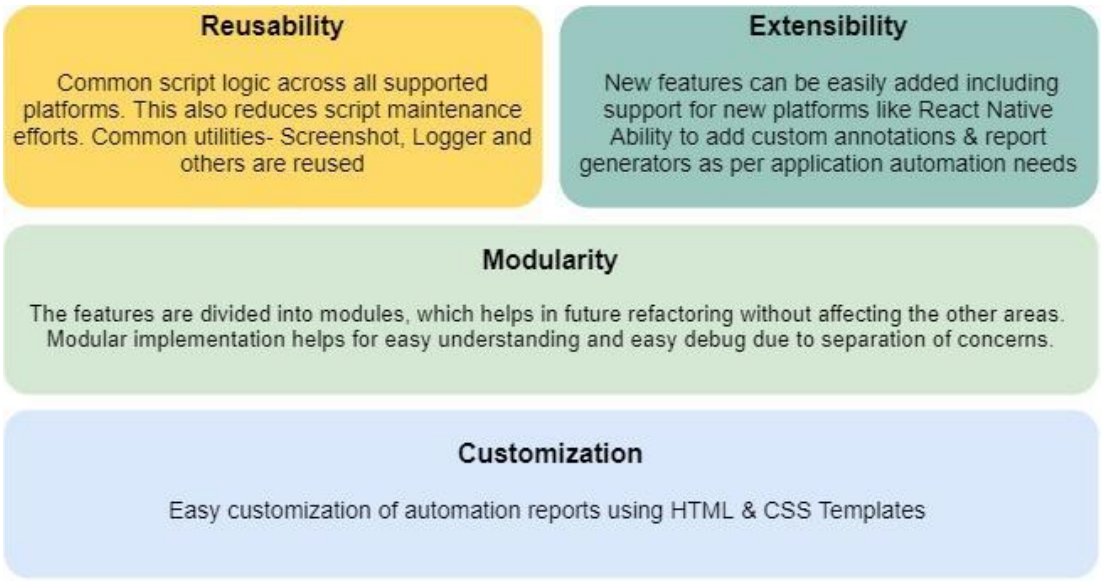
This solution accelerator addresses this issue by providing a reference framework to write a single script for features, for multiple platforms. This helps in reducing the developer's effort for developing multiple scripts and maintaining it.

Apart from the core objective of reducing redundancy of code, the framework provides boilerplate code, which speeds up the automation script development.

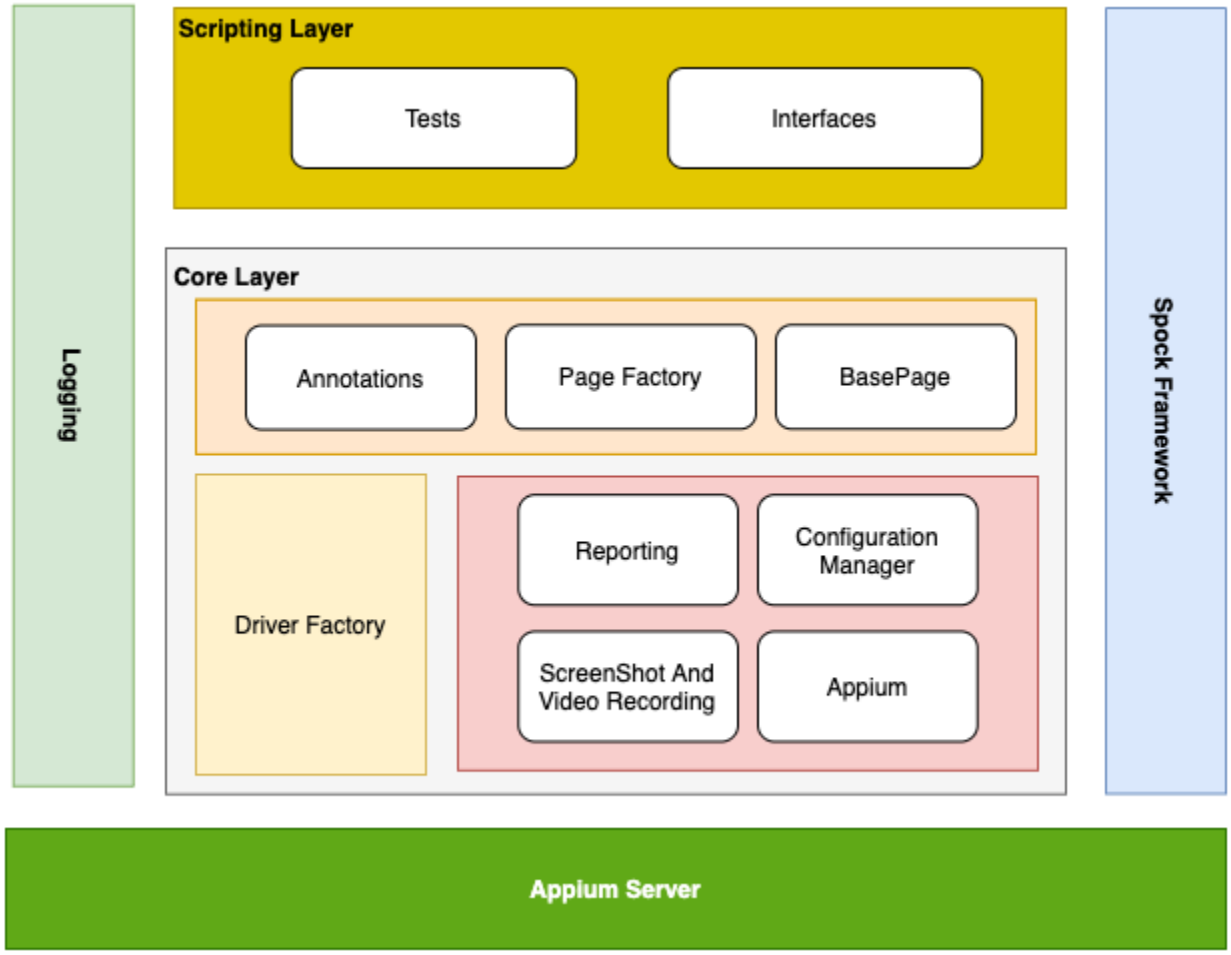
Features

- Custom Report Generation
- Screenshot Capture
- Video Capture (Right now supported for Android only)
- Logger
- Annotation support for Device types (Real/Simulator)
- Annotation support for Platforms (iPhone/Android)
- Custom Annotation Support
- Selective Test Execution

NFR Coverage



Framework Overview



Framework consists of different components, which are grouped into two layers – Scripting layer and the Core layer.

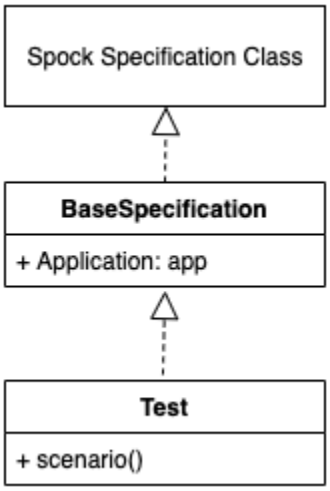
Scripting Layer

The Scripting layer contains the project specific automation scripts, which use the framework provided interfaces. As per the application automation requirements, the developer can create additional interfaces.

.

The Scripting layer uses the annotations to configure the way tests should execute. It uses the interfaces to interact with the UI element of the application through their respective pages.

Test Specification Class Diagram



The scenarios are in the Test class, which extends from the Base Specification class. The Base Specification class extends from the Specification class provided by Spock Framework. The Spock framework provides special methods for— Scenario execution start and Scenario execution end. This provides a hook to perform additional tasks at the start or at the end of the scenario. One of such examples is generating a report.

Core Layer

Core Layer consists of the various components. Some of these components are available to the user for scripting purposes. The components available to the user are as below:

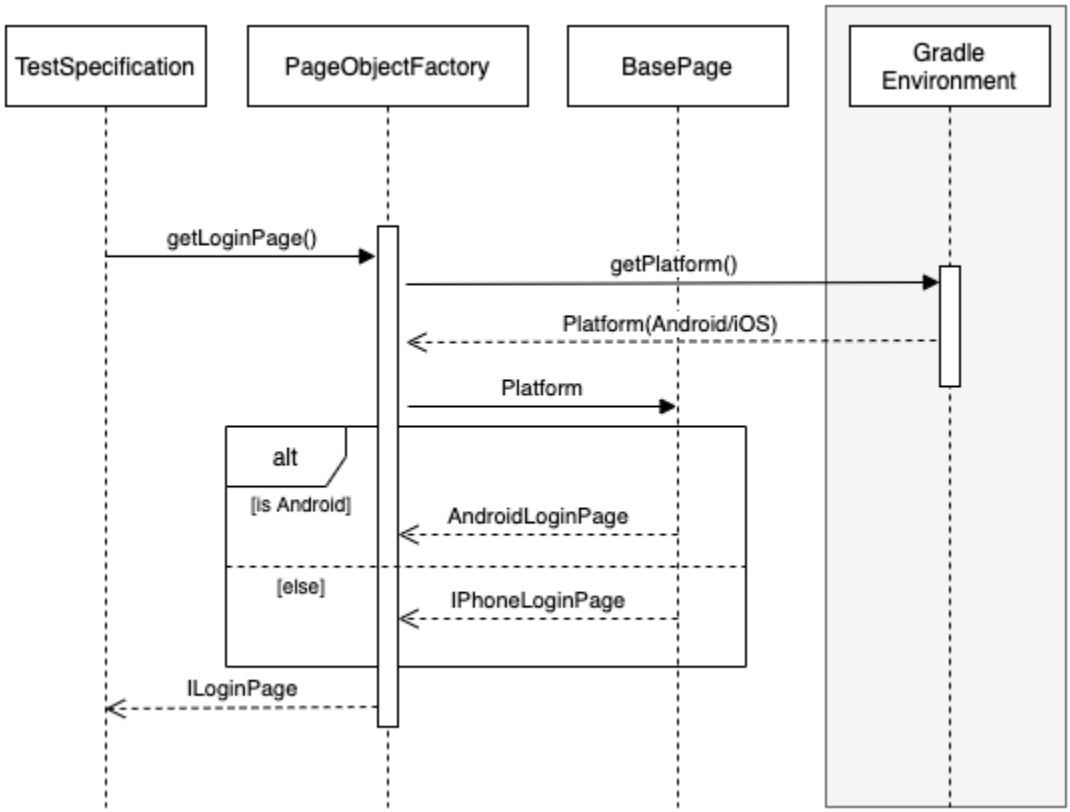
Annotations

Framework consists of Predefined annotations, used to specify different conditions like, tests specific to particular platform or device. It is also useful to tag (identify) the tests with ID and generate necessary information for reporting purposes. Annotations are also internally used to detect the test failures and. The annotation module allows developer to add custom annotation.

Page Factory

Page factory is allowed to create page objects based on the provided platform. The page object methods for different returns the interface which are common for all supported platforms. Developers need to add a getter method in this factory class for any new page.

Sequence diagram of Page object creation

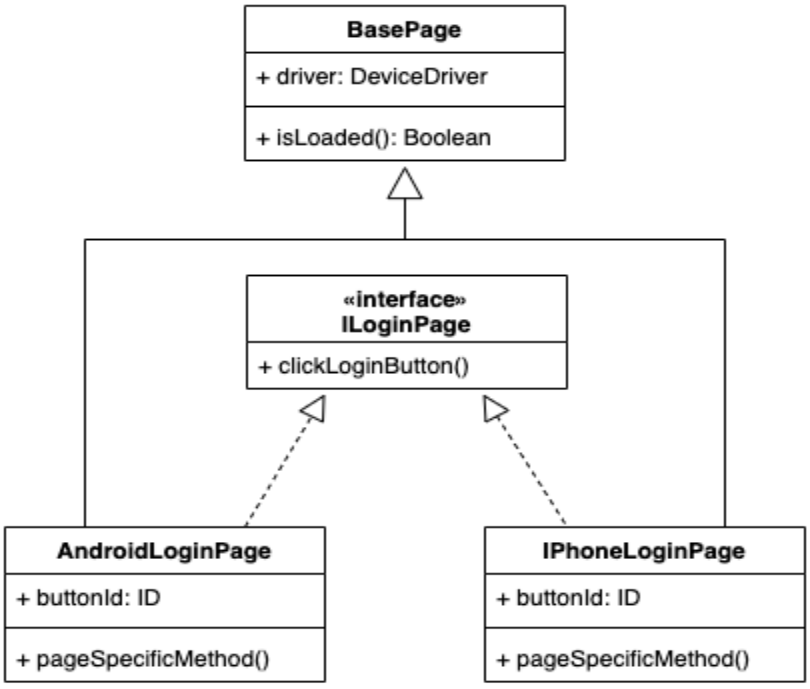


All page objects are created through PageObjectFactory. In a page object factory, platform specific page objects are created that can only be accessed through a common interface. The platform value is passed through the gradle environment using the execution command.

Page Object and Base Page

Page object represents the UI element map for a particular screen of the application and also provides methods to take action on them. All the pages extend the Base page, which provides common functionalities like logging, device driver access, method to find elements based on ID and many more.

Page Class Diagram

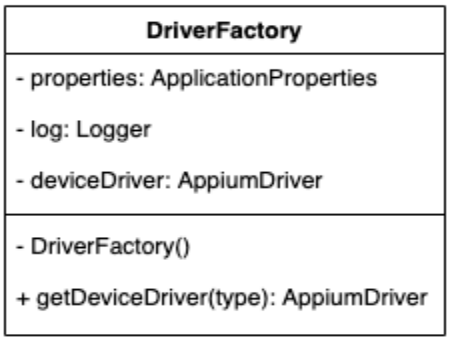


Pages of a specific screen also have to implement a common interface on both the platforms. Each page has platform specific IDs to access UI elements and perform action on them. In the page object, developers can combine the few steps into a single method to improve readability of the script e.g. for login, developer can create a single method for entering user name, password and clicking the login button.

Driver Factory

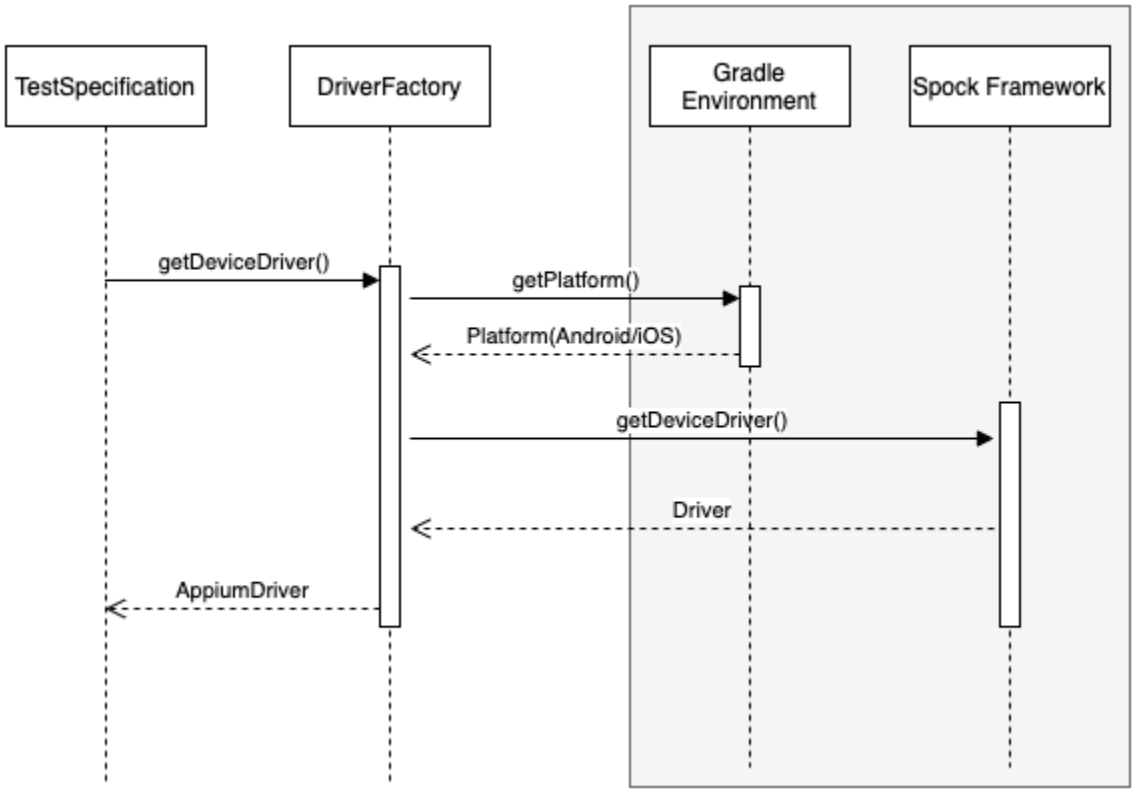
Driver factory is responsible for creating device drivers based on the platform value passed. The additional configuration values required to create a driver are read from the properties file.
Driver Factory returns the base class of the device driver so that it can be commonly used for both iOS and Android.

Driver Class Diagram



DriverFactory is a Singleton class and used to get the Device Driver based on the provided platform. It returns the value as an AppiumDriver, which is Base Class of device drivers so that the returned value should be independent of the platform.

Sequence Diagram of Driver creation

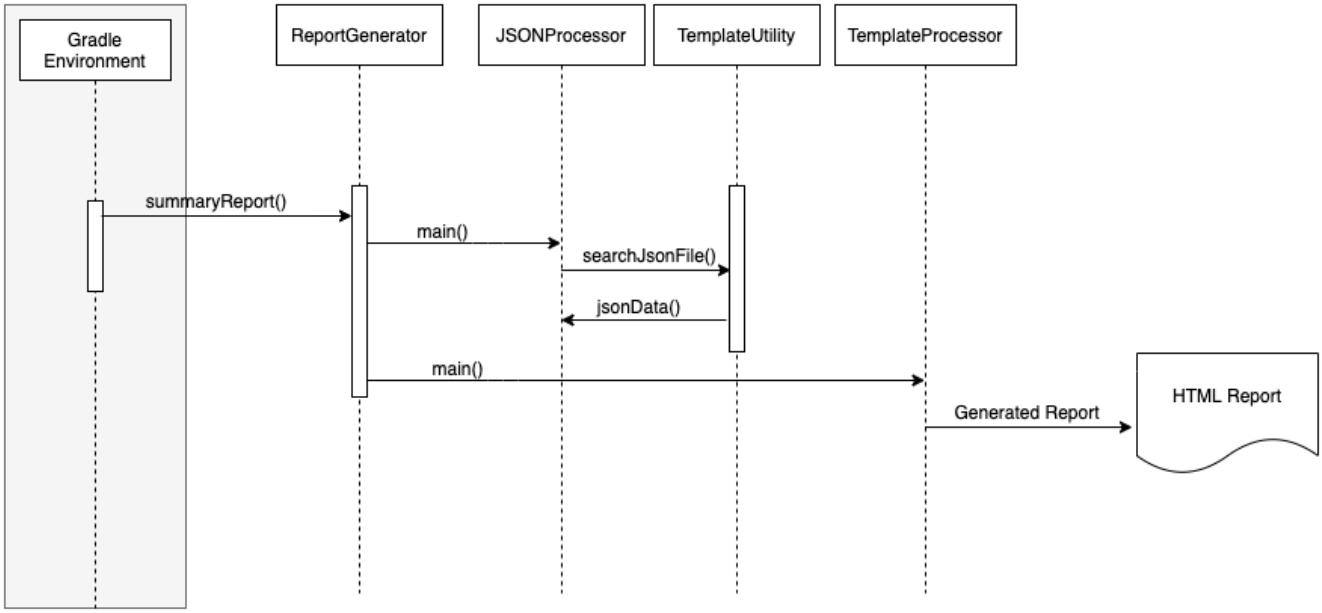


Reporting

Reporting is used to generate custom reports using JSON data generated by Spock framework and HTML/CSS template. The report is generated incrementally, so in case of interruption, the report will still be generated for the executed tests.

Report Generation

Report is generated using the template HTML(StructureTemplate.html) and the CSS(StructureTemplate.css).
For basic modification like color or placement of the data, developers can update these two files but for major modification, it is required to update the tool classes present in ReportProcessingTool package.



Snapshot of the generated report

Summary of Automated Test(s) Executed on Thu Dec 03 15:03:58 IST 2020 by shrikant.purohit

Executed By	Environment	Total Test	Test Passed	Test Failed	Pass %	Total Time
shrikant.purohit	Android - Real	3	3	0	100	0 minutes 43.707 seconds

Tests

Test ID	Test Name	Status
1	About Test	PASSED
2	Converter Test	PASSED
3	Survey Test	PASSED

Tests Details

Test Name: About Test

Test ID: 1

Scenario: Ability to show app version
Given User is on logged in: PASSED
When User is on About Tab: PASSED

Screenshot and Video Recording Utility
Provides an ability to capture screen and record the application screen video during the failure of the scenario. This can be further extended to perform the Visual testing

Appium Wrapper

it provided an ability to start and stop the appium server which is required for test execution. It also manages the logs generated by appium .It helps in avoiding the manual process of starting appium server from terminal.

Logging

Provides to log the messages. It supports the standard log levels – Info, Debug and Error and is accessible across all the layers of the framework. The generated logs are redirected to a separate file, which helps for post execution analysis.

External Entities used in Framework

Spock Framework

This is the core of the automation framework, which provides Specification class to handle the different events of test execution. It also helps in generating default execution reports and the data, which can be used for creating custom reports. It accepts the test description in Gherkin format .

Appium Server

Appium Server is the core component of the Appium architecture. It is written in Node.js and runs on the machine or in the cloud. The Appium Server receives requests from Appium client libraries via the JSON Wire Protocol and invokes the mobile driver (Android driver/iOS driver) to connect to the corresponding native testing automation frameworks to run client operations on the devices. The test results are then received and sent to the clients.

Quick Start Guide

1. Environment Setup For Mac

Appium Setup

```
$ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
$curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash
$brew install gradle
$brew install node
$brew install ideviceinstaller
$npm install -g appium
$sudo npm install -g appium-doctor
```

Android Setup

Download and install Android Studio, from the link below and move to the applications folder of MAC.

```
https://developer.android.com/studio
```

JAVA Home Setup

Type following command in terminal to open nano and set JAVA_HOME.

```
$ nano .bash_profile
```

Set Java home variable as below in .bash_profile.

```
export JAVA_HOME=$(/usr/libexec/java_home)
```

Save and close the opened .bash_profile file.

Re-Open Terminal and run the source command to apply the changes.

```
$ source ~/.bash_profile
```

Now we can check the value of the JAVA_HOME variable.

```
$ echo $JAVA_HOME
```

The result should be the path to the JDK installation.Preferred JDK version is 1.8.221

ANDROID Home Setup

Type following command in terminal to open nano editor and set Android Home along with the path to below-required tools. platform-tools. tools. build-tools.

1. platform-tools.
2. tools.

```
$ nano .bash_profile
```

```
export ANDROID_HOME=/Users/{username}/Library/Android/sdk

export PATH=$ANDROID_HOME/platform-tools:$PATH

export PATH=$ANDROID_HOME/tools:$PATH

export JAVA_HOME=$(/usr/libexec/java_home)
```

Save and close the opened .bash_profile file.

Re-Open Terminal and run the source command to apply the changes.

```
$ source ~/.bash_profile
```

Now we can check the value of the ANDROID_HOME variable.

```
$ echo $ANDROID_HOME
```

The result should be the path to the Android SDK installation.

iOS Setup

Developers can skip this part if they are planning to run automation on iOS devices only.

1. Install Xcode before executing commands.
2. Open Xcode and let it load additional libraries.

Select xcode:

```
$xcode-select --install
```

Please refer to the provided reference link for detailed steps installing appium and the other necessary packages.

Libraries packages for communicating with iOS device:

```
$brew install ios-webkit-debug-proxy
$brew install libimobiledevice
$brew install ideviceinstaller
$brew upgrade ideviceinstaller
```

Web Driver creation:

```
$brew install carthage
$sudo npm install xcpretty
If Error: package.json not created in root folder, then run the command "npm init --yes".

$npm install -g deviceconsole
$sudo npm install ll -g ios-deploy
If Error: run this command sudo npm install --global --unsafe-perm ios-deploy

$sudo gem install
$sudo chown -R $(whoami): /usr/local/lib/node_modules
$cd /usr/local/lib/node_modules/appium/node_modules
$sudo chmod -R 777 . appium-xcuitest-driver/*
```

Web Driver installation:

```
$cd /usr/local/lib/node_modules/appium/node_modules/appium-webdriveragent
$npm i -g webpack
$sudo ./Scripts/bootstrap.sh -d
$bash Scripts/build.sh -d (eslint-config-appium not found error)
$Open .
Double click WebDriverAgent.xcodeproj
```

```
Goto Xcode> Preferences... > Accounts > Add Apple Developer Account
Click Manage Certificates.. > + sign Drop Down > click iOS Development and close
Select WebDriverAgent > General > Automatic manage signing login and select Account from
Drop Down.
Select WebDriverAgentRunner > General > Automatic manage signing login > and select
Account from Drop Down.
- If signing error - change build identifier from com.facebook.xxxx to
com.$(COMPANY_DOMAIN_NAME).xxxx.
Where COMPANY_DOMAIN_NAME is the domain used in provisioning profiles for device signing.
For example: com.globallogic.xxxx.
Build the code.
Open New Terminal window and run appium.
Note : Add device to Apple Developer Account if not already added.

For xcode path Error :
$sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
$sudo xcode-select -s /Applications/Xcode.app/Contents/Developer
```

2. Environment Setup for Windows

Android Setup

Download and install Android Studio, from the link below.

```
https://developer.android.com/studio
```

JAVA Home Setup

Download and install JDK 8 version 1.8.0_261 from the official website.
Set JAVA_HOME environment variable. If downloaded on the default path the value would be C:\Program Files\Java\jdk1.8.0_261

ANDROID Home Setup

Use Android studio SDK Manager for downloading Android SDK.It will also help in locating the path SDK with developer need to set as ANDROID_HOME as environment variable.
Developers also need to set the path of platform_tools and tools.

Appium Setup

Download and install Node.js, from the link below.

```
https://nodejs.org/en/download/
```

Install the latest version of appium using the following command on command prompt.

```
npm install -g appium
```

3. Checkout The Framework

Checkout and download source code from here and open it in intelliJ IDEA CE.

4. Run Existing Tests

Samples Overview

Sample consist of 3 scripts, each representing the different scenarios.

Converter Test

It shows how a single script covers a scenario in which a particular screen on both the platform (Android and iOS) has the same UI and same business logic. For such cases, the developers only need to maintain platform specific UI element ID in the page object.

Survey Test

It shows how a single script covers a scenario in which a particular screen on both the platform (Android and iOS) has a different UI but has the same business logic. For such casee, platform specific page objects need to handle UI actions separately as per UI component, like in this example for selecting survey option, in iOS application has Radio button but on Android, it has drop down menu.

About Test

It shows how a single script covers a scenario in which a particular screen on both the platform (Android and iOS) has the same UI but has different business logic. In that case verification is done based on the platform and might require condition check at the script level which set of steps should be performed for the given platform.

Configuration

Developers need to put the android app apk/ios app ipa in the root folder on the same level as the "src" folder.
Developers also need to update following properties as per their need for execution:

1. **apk_path** / **ipa_path** - Executable name.
2. **Android_appPackage** / **bundleId** - Application package id.
3. **Android_appActivity** - Android startup activity

Also developers might need to update the gradle version of distributionUrl in **gradle-wrapper.properties** file based on their configuration.

Run And Test

Following command need to execute the test:

1. For whole execution:

./gradlew cleanTest test -info -Dplatform=iPhone -DdeviceType=Simulator

2. For running particular test based on ID

./gradlew cleanTest testwithIDs -Dplatform=Android -DdeviceType=Real -DtestIDs=TEST_ID1,TEST_ID2,..

Supported parameters

- platforms : iPhone, Android
- deviceType : Real, Simulator

5. Adding New Test

1. Creation of Test Class

For automating any scenario, developers need to create a new groovy test class in acceptance test package, this class should extends BaseSpecification class

2. Adding Required Annotations

Developers should add following required annotations:

- **VersionTraces** - To track the test based on ID
- **Details** - To describe the scenario.It is used for the reporting purpose.

There are other optional annotations are also available:

- **iPhoneOnly/AndroidOnly** - To specify the platform specific test.

- **Real** - To specify that the test can only be run on real devices (iOS only. For Android, test will run on real device only)
- **IgnoreRest** - This will ignore other scenarios present in the test file and execute only the scenario which has this annotation.

```
@VersionTraces("1,1")
class AboutTest extends BaseSpecification {

   @Details ('Ability to show app version')
   def "Ability to show app version" () {
   }
}
```

3. Adding Page Object and its factory method

Developers need to create the page object as per the app screen for UI interaction. Page class should be derived from BasePage to get the essential functionality. Also need to add the getter method of the newly created page object in the PageObjectFactory class to get the platform specific page object which can be accessed through interface.Please refer to the given sample in the code.

Android Specific Page:

```
public class AndroidFirstTabPage extends BasePage implements IFirstTabPage {
}
```

iPhone Specific Page

```
public class IPhoneFirstTabPage extends BasePage implements IFirstTabPage {
}
```

Page Object Factory Getter Method

```
public ILoginPage getLoginPage() {
   switch (mPlatform) {
       case Android:
           return new AndroidLoginPage();
       case iPhone:
           return new IPhoneLoginPage();
```

```
        default:
            log.error("Platform not found");
            throw new IllegalArgumentException("Platform not found");
    }
}
```

Interface for Page Object

```
public interface ILoginPage extends IBaseInterface {
    void loginWithUserName(String username);
}
```

6. Troubleshoot

If any test fails in the generated report, analyse the logs and the screenshot to identify the root cause of test failure.
Screenshot will be captured for the screen in which that test failed.

References

http://spockframework.org/