**Building automation framework around Appium**

In the automation world be it web application or mobile application or any windows application for that matter, framework establishment if quite necessary. In this discourse let’s understand how important it is to build a potential framework.

**What is Framework?**

In a summary framework is all about set up a protocol for any project i.e. test automation in this case. Enforcing the rules for the team to write and maintain the code base and helps us integrate the different components to make the project more enriched so that it helps us to perform seamless execution and serve a better results.

**Following are the some of the benefits of having a robust framework**

* The entire team is aware of developing and maintaining the code base as per the set standards since the code is organized in well-defined packages and properly structured
* We can integrate different components that helps for the purpose of running the test cases, reporting the results and logging the logs..etc
* Thanks to the framework as it helps us to avoid the redundant code to be written and reduce the maintenance
* Framework can be re used for any similar projects internally within the teams

**What is Appium?**

Appium is an open source tool that’s helps for mobile application automation, it helps us automate all the genre of the applications such as web, native and hybrid application. Appium works on top of selenium, it internally uses the webdriver commands to before it talks to the mobile application

IOS

IOS driver

Test scripts

Appium server

Android

Json wire protocol Androiddriver

Windows

Windows driver

From the figure we understood the architecture of the Appium as a tool, if we deep dive into the technical internals we can see it uses different components to help us achieving the tasks. Given that lets understand how to build a best appium framework to build robust and efficient scripts

**Components of an Appium framework:**

Below are the tools required to be installed on the desktop in which the tests are run. Let’s look at each one of them in detail as how that is useful? What are the vivid tools exist for each of them

**Node Modules**

Installing node.js in the computer should be the starting point, because all the other packages such as appium , appium-doctor can be installed through the npm install

**Appium Server**

Appium server is made available in node modules, we can install this by executing the command

Npm install –g appium

**Build Tool**

Build tool helps us building the code then execute.

Build tool helps us to manage the required dependencies needed for our poject such as testNG, appium client, selenium..etc.

Build tool helps us to run the tests from the command line.

We have a few build tools such as maven, gradle, ant. We can choose any one of them based on the suggestion from the team.

**Selenium-webdriver**

As discussed in the above archirecture Appium uses selenium to talk to browser that runs on mobile device, hence having selenium jar files loaded into our project is must.

**Test runner**

One must run the test cases once they are developed, test runner does exactly the same for us. We have a few test runners such as J-unit, N-unit, testNG..etc. testNG is best used in test automation as it has several annotations to help with running at a sequence.

**JDK**

Java development kit is must in order to develop any java application. Hence this should be installed from oracle official site. Appium has java support until java 12

**Android SDK**

Android SDK is in general used to develop the android applications, but from the test automation perspective we don’t need the entire studio rather extraction of a few platform tools would be sufficient. With these tools we can actually connect the mobile device to the computer and understand the several properties of it.

**UI Automator2**

This tool is used to set the capabilities of the device therefore the device can be connected and the UI of it can be accessed. This will help us connecting both emulator and the real device. We can capture the application elements (locators) with the help of this tool.

**Extent reports**

Extent reports are used to project the test results in more meaningful format. We can include the jar files into our project by adding these dependencies to our pom.xml file

**VCS system**

Version controlling system should go hand in hand with any software development or software test automation projects. VCS systems will greatly help the team in maintaining the new revision of code by including everyone’s changes to the code base. With the help of this tool back tracking is pretty much possible and we can integrate with CI/CD too

GIT hub, GIT lab, bit bucket..are a few examples of VCS system

**CI/CD system**

In the recent era in order to achieve continuous testing integration of test automation code base with CI/CD tools. This will help us to execute the tests automatically without any manual intervention. Jenkins, Teamcity, Azure develops are some good examples for this

**Framework architecture**

By far in this discourse we have leant the components that make an appium framework more versatile. Now we will focus on how to organize the code base, these will in turn be considered as best practices

**Folder structure**

When any maven project is created, we get the below shown project template to add the code. So we need to ensure that we define proper guidelines to maintain the code base

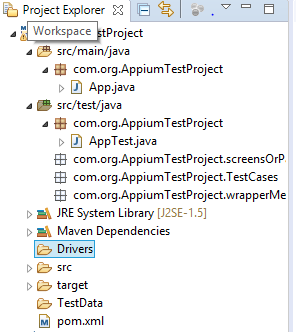
Since the project is test automation one, it is okay to keep all the required java class files under src/test/java folder

It is highly recommended that to create a sub package for the wrapper functionalities and for the test cases as shown in below snap, so that we can have a clean differentiation of test cases and the build and functions

All the driver exe files such as Android driver, IOS driver and windows drivers are supposed to be kept in the driver folder, so that the team can have a common understanding of where are those files reside and it would easy for anyone to update them to the latest if required

It is one of the best practices to keep the test data, which is being maintained in csv, excel in a separate folder called test data, so that we know where it is update as per the need

For reports when the test case is executed, testNG will create a test-output folder inside which we can see he testNG html results files



**Design patterns**

Despite the test automation project, design patterns must be adopted to make the framework more reusable and reduce the maintenance effort, below are the useful techniques to achieve the same

**POM implementation**

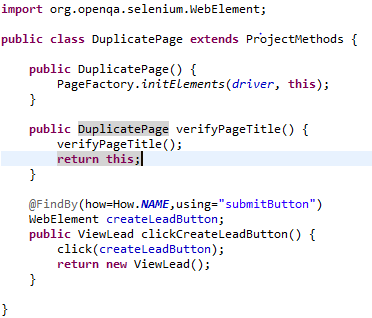
Page object model is one of the design technique of separating the application page code from the test case code. This will avoid doing re-work when there are any changes done to the application screens.

POM class file will have page factory method implementation

Generally we will define each page as separate java class

We also find the needed locators that are associated to the particular page with the help of @findBy annotation

Sample POM class file



**Dependency injection**

This is the useful technique of passing the value between the class file without having them coupled. The dependent object receives the object value from the class object that it depends on. This ca be helpful in case when we integrate cucumber (BDD) with our appium test. Since the hooks class can’t be extended by any other class we use this concept to have the driver object shared to any step definition class that is available in the project

**Retry logics**

While running the appium test cases we tend to observe lot of events, a test can possibly be failed, skipped and ignored. We can have a test that is failed to be run again. testNG listeners are greatly helpful to achieve this. We can try writing a logic to re-run the test case under the method (On test failure), we can also take snaps of the screens in which the test is failed

**Use of property files**

We give lot of inputs to our appium test cases such as platform, device name, device OS, UDID, app package name, app activity, path of the apk/.ipa file..etc. rathe than giving thee details directly in the test case it is always good practice to fetch this information from a property file . this helps all the tests use the same details when we are running the test cases against a particular device . also when ever tests are running this property file need not be built as it’s not a class file. Whenever we need some change, changing the value in that particular location would be enough.

public static void main(String[] args) throws MalformedURLException {

// Step01:- Set Desired Capabilities

DesiredCapabilities dc = new DesiredCapabilities();

dc.setCapability("appPackage", "com.android.dialer");

dc.setCapability("appActivity", "com.oneplus.contacts.activities.OPDialtactsActivity");

dc.setCapability("deviceName", "OnePlus 7T Pro");

dc.setCapability("platformName", "Android");

dc.setCapability("automationName", "UiAutomator2");

dc.setCapability("noReset", true);

// Step02:- Create the driver and hit the appium server

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

From the above sample code we can store all the values of the capabilities in a property file and can use them, whenever we want to execute the tests against another device we can tweak the values in the property file so that all tests are now pointed to the new device platform

**Test runner annotations usage for sequential and timely execution**

At a high level in order for any appium test case to be executed we need the appium server up and running, code should be loaded and the driver object should be loaded with all the desired capabilities and the test case will run then finally report results and close server if needed

In a conventional framework we will have numerous test cases, albeit we have many of them starting appium server once started at the beginning is enough . similar way we usually close in the end.

There would be some more events such as opening and closing of broser window (in case of web app) opening and closing of an app (for native app) are required to be executed may be just before and just after the test cases

Hence the recommendation here is to use the testNG annotations wisely.

Eg:

@BeforeSuite- start the apium server in here

@BeforeSuite- configure the driver (Android driver/ IOS driver)

@Before method – open tha app/ browser

@Test- run the actual test scenario

@AfterMethod- close the app or browser

@ AfterSuite – close the server.

Thank you very much for reading this through the blog. Hoping that this has been very useful for an individual to understand how to design a good framework for a mobile test automation project which is appium in this case.

Happy learning ☺