Automation Framework

Ostan Dsouza
CodeCraft Technologies
19/12/2020

Table of Contents

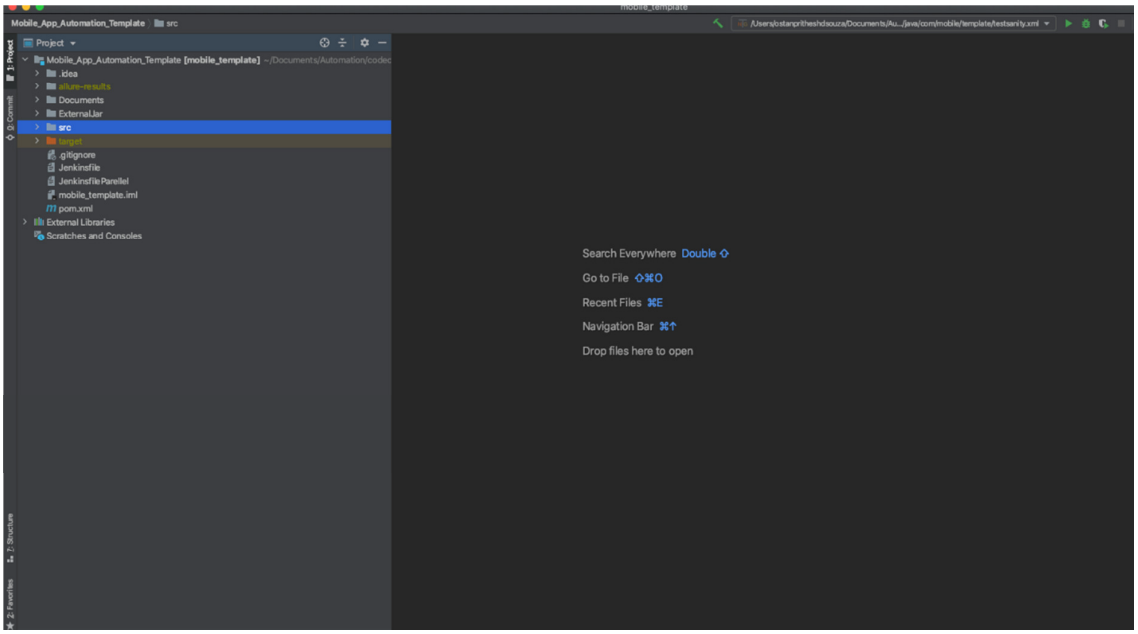
Chapter 1: **Introduction**

Test Automation Framework represents a framework used for test automation. It provides some core functionality (e.g. logging and reporting) and allows its testing capabilities to be extended by adding new test libraries. A test automation framework defines common functions such as handling external files, GUI interaction, provides templates for test structure that helps in reusing the same piece of code throughout the project.

Chapter 2: **Our Automation Framework**

For Automating the testcases for this application we are primarily using Hybrid Appium Framework which is an open source Automation framework to write the testcases and use open source libraries to interact with Application running on browsers, real devices or cloud platforms. Our current framework is based on maven and we use page objects for maintaining

```
.gitignore

ExternalJar/

pom.xml

ReadMe.md/

Jenkinsfile

src/
```



Files

`pom.xml`

Since this is an maven project pom file will have all dependencies that our project requires and downloads them whenever they are added.

`.gitignore`

This file contains all the directory paths that must not be pushed to the git. eg: Allure-reports, target files etc

`Jenkinsfile`

This file will contain a pipeline script that will execute a pipeline when called upon.

Folders

`ExternalJar/`

ExternalJar folder will contain jar file that is used for JWT token authentication for Zephyr integration.

`src/`

The folder is named src contains all the tests that are built into the final project. Each folder within corresponds to helper methods and tests themselves.

Here is a basic example layout to get an idea how this can look like:

assembly/
.idea/
main/

assembly/
java/
resources/

test/

java/

Main

Pages

Main folder has a page object model structure where-in for every screen in an app we create a separate package inside this folder and all the locators and methods corresponding to that page will go in there.

API's

Api's folder has base definition for all api calls that are being used in current framework.

Utils

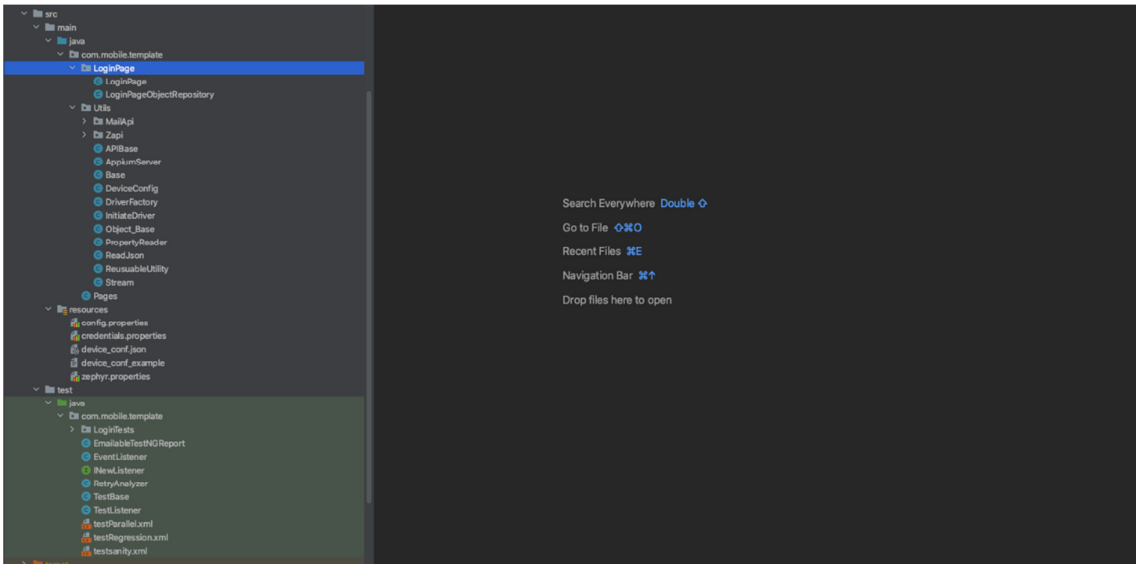
All our core framework is in utils folder. From all driver setup and respective capabilities for all supported platforms to tear down method can all be found inside utils folder

Test

Consists of all tests files packaged into individual folders for every screen in an app. Along with the test files it also has core test class consists of all generic methods from which other test can inherit.

Section 2.1: **Source Directory**

Base template will contain a parent pom which will hold all the dependencies and plugins needed by the template. Parent pom is also configured with two sub modules, one of which is core template. Core template is core part of a framework, which will contain all the dependencies, set up files, utility files that every project needs. Let first look at the folder structure of this.



```
|Mobile_Automation_template/
    |src/
        |main.java.com.template/
            |Utils/
              |MailApi/
              |Zapi/
              |ApiBase.java
              |AppiumServer.java
              |Base.java
              |DeviceConfig.java
              |DriverFactory.java
              |PropertyReader.java
              |ReadJson.java
              |Resuability.java
              |Stream.java
              |AllureLogger.java
            |Pages/
              |LoginPage.java
              |LoginPageObjRepo.java
```

Section 2.1.1: **Utils**

Utils is one of the modules in our framework which will contains all the base classes, helper methods, reusable methods that we will be using in our automation project. Since core template is module, all the dependencies from parent pom will be inherited to child modules as well.

Section 2.1.1.1: **MailApi**

This package will contain mail api implementations for mail services like mailinator, maildrop and gmail. Disposable inbox like mailiantor service provide their websocket

connection to listener to their inbox where as for other disposable inbox like maildrop provides their own api for this purpose. As far as gmail is concerned we make use for java mail dependency to connect to gmail SMTP server. Above mentioned scenarios are implemented and can be accessible with this directory.

Section 2.1.1.2: **ZAPI**

This package contains all files that are required for integrating zapi with our framework. ZAPI basically exposes Zephyr for JIRA data via REST APIs which allows you to build your own integration with Zephyr for JIRA. ZAPI (or Zephyr API) is an Add-on to Zephyr for JIRA that allows access to its testing data including the ability to view and upload information programmatically.

Section 2.1.2.3: **Managers**

These classes are backbone of our framework. It mainly contains all base methods which we are going to be inherited in our sample project. These files include TestManager which is an interface depicting the structure of TestBase which is going to be extended across test class. ApiBase class will have Okttp service implementation which helps in making api call. Base class will have all the helper methods that will be required by page class and lastly, we have objectBase which will help in initializing all page objects.

Section 2.1.2.4: **Parsers**

These classes will help parsing or reading an external file sources namely property files, json files and excel files. These external files may contain test data required by out tests, credentials or domain details etc.

Section 2.1.2: **Page Objects**

Here in our framework we follow page object approach, which defines each application page or section that is displayed on more than one pages containing respective locators and helper methods. Page objects are a classic example of encapsulation - they hide the details of the UI structure and widgetry from other components (the tests).

Section 2.2: **Test Directory**

Core template is one of the modules in our framework which will contains all the base classes, helper methods, reusable methods that we will be using in our automation project. Since core template is module, all the dependencies from parent pom will be inherited to child modules as well.

Section 2.2.1: **Test Base**

This file will implement TestMaster interface from base template, which will mainly contain how the base class will be structured. TestBase class will have setup and teardown methods, which instantiates the kind of driver that is required for project and teardown will kill this instance. Also, base file will initialize different property readers, whose values will be read from respective test or page classes.

Section 2.2.2: **Test Classes**

This directory will contain all test script that are written for a particular project. All test classes will inherit behaviours from TestBase class, which provides required driver instance and test data. It should be noted that test class should not contains any business logic, it should be used for assertion purpose only. Each test class should accompany required listeners and prerequisite data.

Section 2.2.3: **Listners**

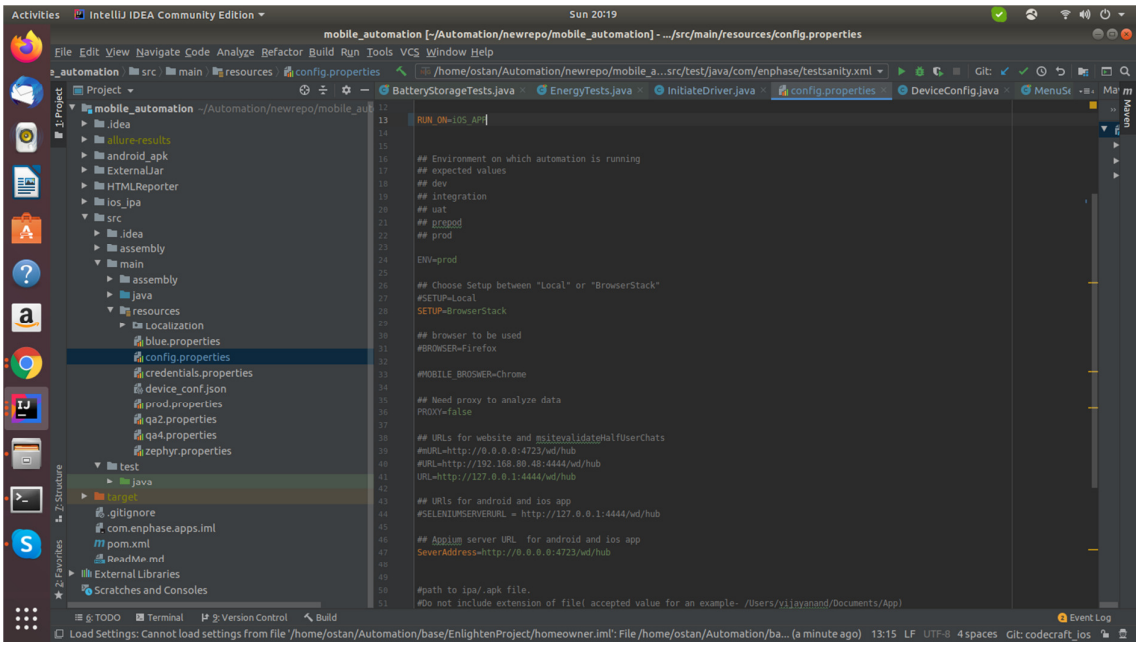
This package will contain all the common testing listeners that are used across the project. TestNG provides the @Listeners annotation which listens to every event that occurs in a selenium code. Listeners are activated either before the test or after the test case. It is an interface that modifies the TestNG behaviour. Listeners "listen" to the event defined in the selenium script and behave accordingly. It is used in selenium by implementing Listeners Interface.

Section 2.2.3: **EmailableTestNGReport**

This package will contain all the common testing listeners that are used across the project. TestNG provides the @Listeners annotation which listens to every event that occurs in a selenium code. Listeners are activated either before the test or after the test case. It is an interface that modifies the TestNG behaviour. Listeners "listen" to the event defined in the selenium script and behave accordingly. It is used in selenium by implementing Listeners Interface.

Section 2.2: **Resources**

This package will mainly contain all property files and test data in form of json or excel files. Property file will include config.properties which helps in identifying the platform or other configurable items, credentials.properties will contains credentials for login etc.



Chapter 3: **How to run this?**

Before running automation project, make sure that folder structure is followed as explained above. In short both the project must reside in same folder i.e your sample project and also base template project.

mvn clean test

or

mvn clean test -Dsurefire.suiteXmlFiles=<path to testing xml file>