**## Project**

This Testing framework is designed as a POC for implementing a Test Automation Solution.

**## Requirement**

The design of this Testing Framework is in keeping with the requirements of the Technical Assignment,( as per the email received from the Dilan Shrimption, i.e. Talent Acquisition Consultant(dated 2nd October 2020), for the role of ‘Test Automation Engineer’.

**## Home Assignment(as quoted in the email)**

## Instructions

Using Selenium with your choice of language and framework, automate the following test cases.

We are looking for your ability to structure your code correctly and how you identify the elements of a page. It is also important to show what you have developed can be executed.

Application URL: <http://jupiter.cloud.planittesting.com>

## Test cases

**Test case 1:**

1. From the home page go to contact page
2. Click submit button
3. Validate errors
4. Populate mandatory fields
5. Validate errors are gone

**Test case 2:**

1. From the home page go to contact page
2. Populate mandatory fields
3. Click submit button
4. Validate successful submission message

**Test case 3:**

1. From the home page go to contact page
2. Populate mandatory fields with invalid data
3. Validate errors

**Test case 4:**

1. From the home page go to shop page
2. Click buy button 2 times on “Funny Cow”
3. Click buy button 1 time on “Fluffy Bunny”
4. Click the cart menu
5. Verify the items are in the cart

**## Downloads**

1. Download Java.

Source: <https://www.java.com/en/download/>

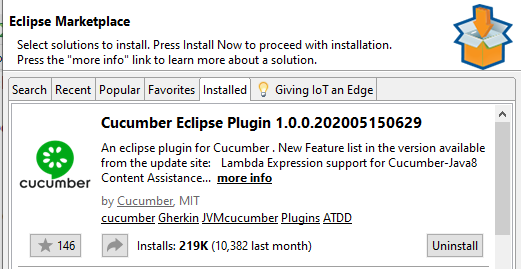
1. Maven download

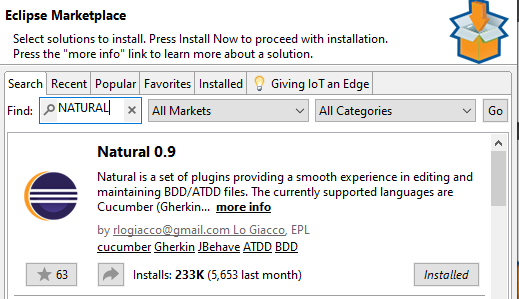
Source: <https://maven.apache.org/download.cgi>

1. Eclipse download

Source: <https://www.eclipse.org/downloads/>

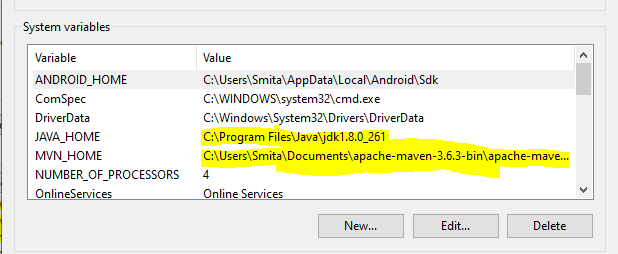
Kindly ensure, the below plugins are installed in the Eclipse IDE for smooth running/editing of the BDD Framework.



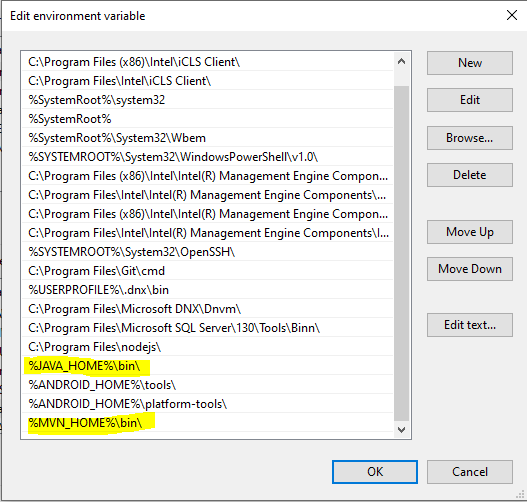


**## Pre-requisites**

The necessary software, as given in the download-section are downloaded and installed accordingly. The environment variables for the JAVA\_HOME, MVN\_HOME are set as shown below:



The Path environment variables are set as shown below:



Note:

The configuration shown above, is considering the fact, that we’ve a Windows 10, 64-bit system in place to run the framework.

Note:

The Framework was developed and built on Eclipse IDE.

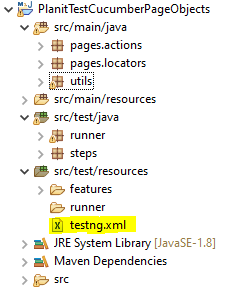
**## Framework**

The framework, which we’ve designed, is a **Cucumber BDD with Page Objects**, implemented as a Maven Project. And the libraries could be configured by quoting the dependencies in the POM.xml.

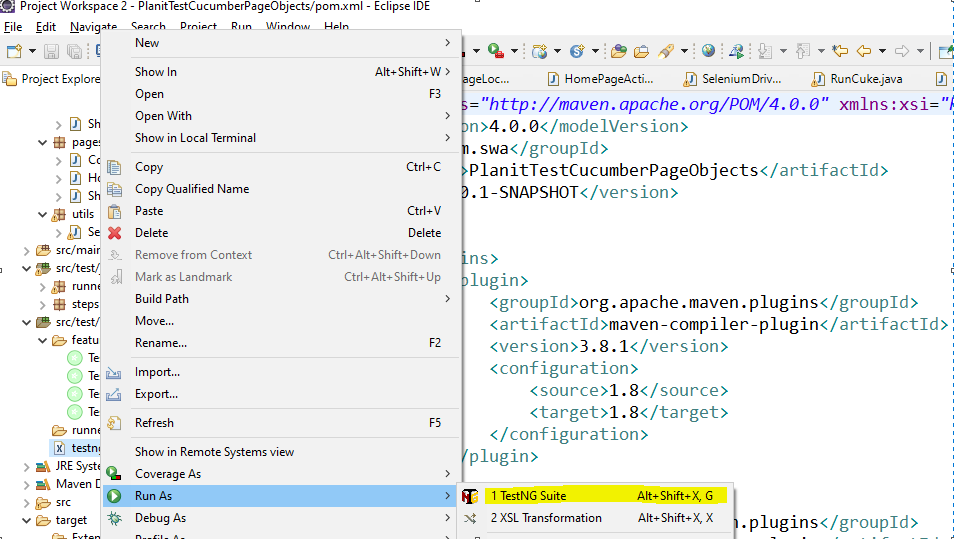


**## Running the Framework**

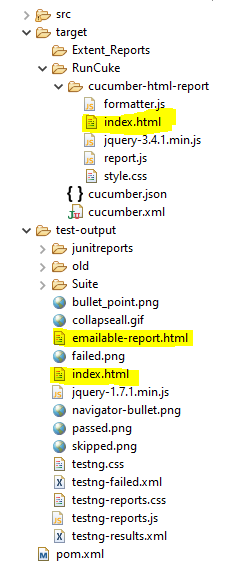
To run the Framework, right click on the ‘testNG.xml, of the ‘PlanitTestCucumberPageObjects’ project as shown in the below pic.



Right click on the ‘Run As’ and TestNG Suite as shown in the below screenshot.



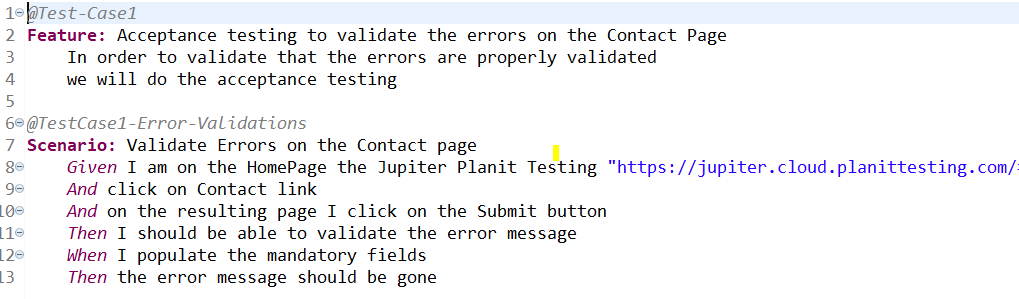
**## Reports Generated**

The **CucumberPageObjects** framework, that we’ve used has automatic reporting enabled, so whenever we run the Framework, the reports generated could be examined by right clicking on the ‘**CucumberPageObjects’** framework, clicking on refresh and clicking on the test-output as shown below.

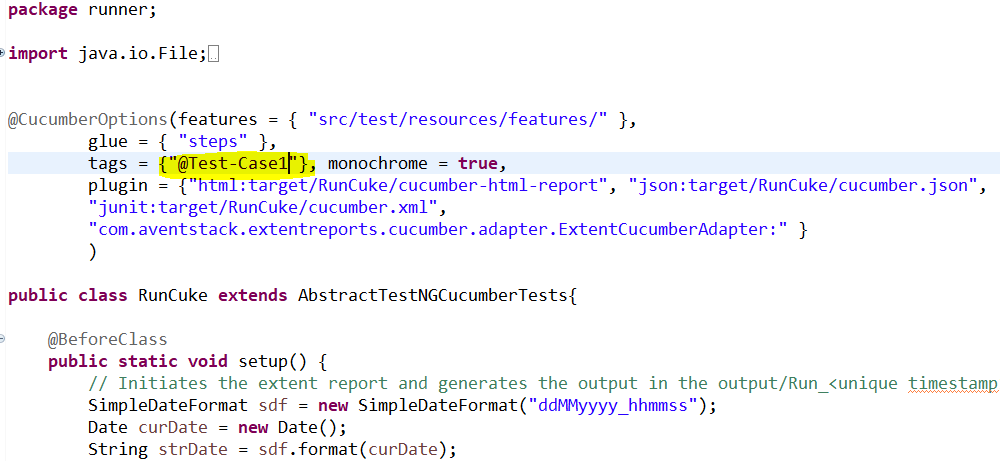
As shown in the above pic, we need to open the index.html/emailable-report, wherein we can see the detailed report of each test case run.

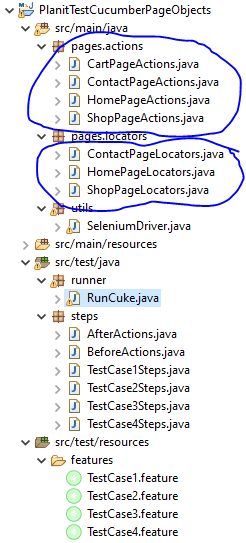
**## The Test Cases and other file locations**

The Test Cases are defined in the feature files and the related step definition files as shown below in the pics. The comments are included in the files, where ever necessary to understand the workflow of the Framework. Here, it should be noted that we’ve the related Locators and Page actions defined separately for each given ‘user story’. For e.g. To validate the errors on the Contact page, we’ve defined the story in the ‘TestCase1.feature’, for which we’ve a separate ‘TestCase1Steps.java’ file. This step definition file refers to the relevant Page action and Page locators file, which could be found out in the Step file itself. Here every Page has a separate Class(i.e as per the Page Object model framework).

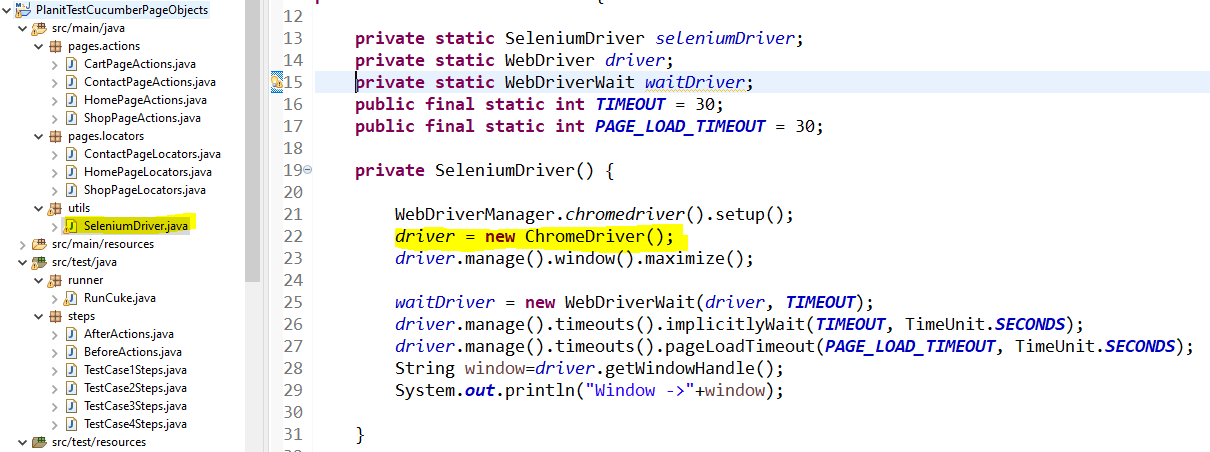


Note: We need to make changes in the RunCuke.java in order to run each feature- file/Test-Case separately or eliminate the tags defined in the RunCuke, to run all the feature-files/Test-Cases. Please refer the pic below





Note: As shown in the above pic, we can see the Feature files, wherein each Test-Case is defined separately. Also could be seen the relevant step definition file for each feature file. The files circled in blue shows the relevant Page Locator and the Page actions file. Here every page is treated as a separate class, which refers to a particular actions class and a locator class. So maintaining the locators becomes easy.



The Driver is maintained in a separate class as shown in the pic above. It could be customized further to do the testing with various browsers of ‘Internet Explorer’, ‘Mozilla firefox’, Safari etc. Here for the sake of testing, we’ve used the most stable ChromeDriver.

**## Further improvements/enhancements to the Framework.**

While, every care and effort is taken to ensure, that the Framework is robust and runs smoothly, yet there is always a room for improvement in any Testing Framework.

Additional reporting features could be incorporated in the Framework, such as ReportNG. Logging could be enabled using Log4j.

The Framework was prominently tested on the Chrome browser, as it is the most stable for our testing needs. However further modifications could be made to the Framework, so as to accommodate the need of Firefox and InternetExplorer browsers as well.

And any more suggestions to improve on the Framework, are most welcome.

**## Thanks giving note**

Here, I’d like to sincerely thank you, for taking time to go thru my hard work done.