**Solution:**

1. What other tests would you suggest could be written? You do **not** need to write them out in detail or code them.

* Validate for a set of test data and check how much borrowing capacity amount it is returning for each different value set.
* Validate the alert messages if we do not enter each field (mandatory/non mandatory) and click calculate
* Verify the scenarios in different browsers as well in different platforms(like mobile and ipad devices).
* Verify invalid data test scenarios for each of the field.
* To test the responsiveness of the page.

1. If this test was part of a much larger test set, how would you speed it up?

* Can do parallel executions like group a set of test cases which are to be done in a sequence or related test cases and execute these groups parallel.

1. Sometimes UI tests can fail unpredictably. For example, the page may not have fully loaded before test automation attempts to click a button on a webpage. How would you improve reliability of these tests?

* This is a common problem when we are doing automation. For this we have a feature called implicit wait in selenium, which can be used. The wait time is not hard coded. It is calculated on realtime. It just waits for an element to load in the UI and once completely loaded, then it proceeds to the next step.

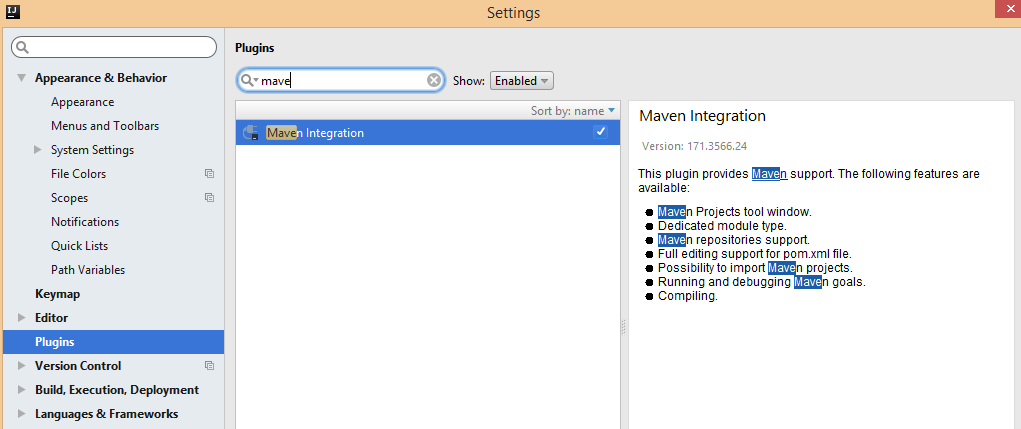
I have implemented the solution in testng- cucumber framework. The project is of type maven and the programming language used in java.

**Steps to install the Project:**

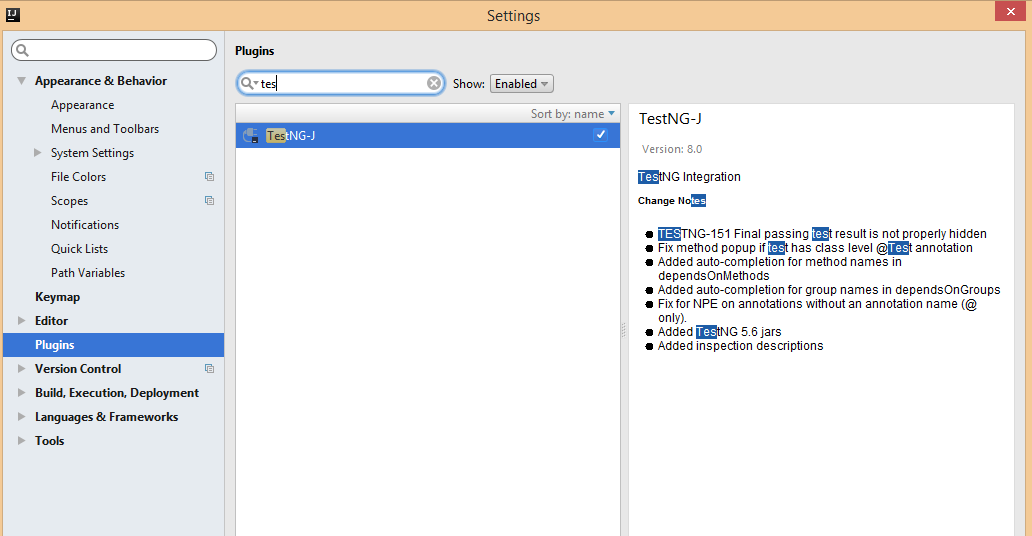
* Step-1: Copy the repository (anz-test-exercise) from GITHUB to you local machine
* Step-2: Browser & version required : Chrome - 73.0.3683(as this test will run in chrome browser)
* Step-3: Import the project to eclipse/Intellij. The plugins that are required are maven, testng and cucumber for java. The essential maven dependencies are mentioned in the pom.xml file.

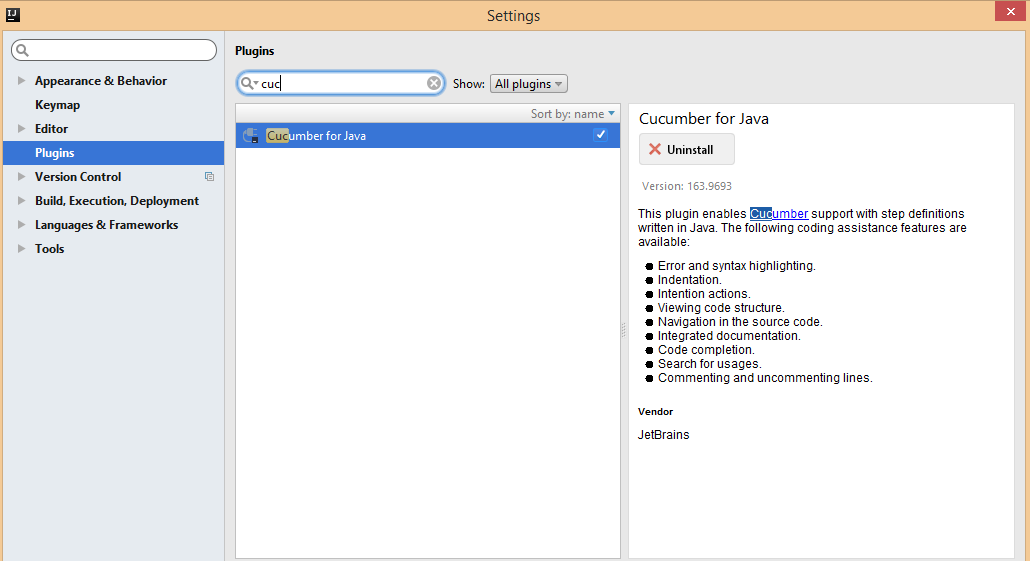
Screenshots of Plugins used:

1. Maven



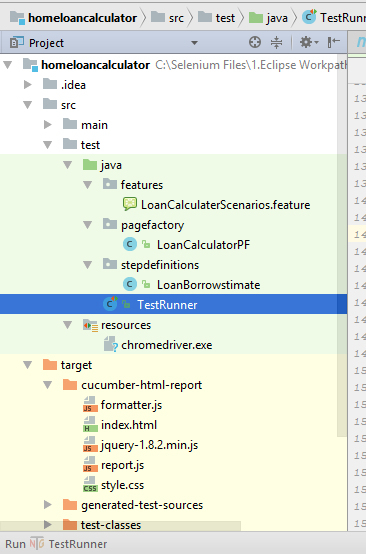
1. TestNG



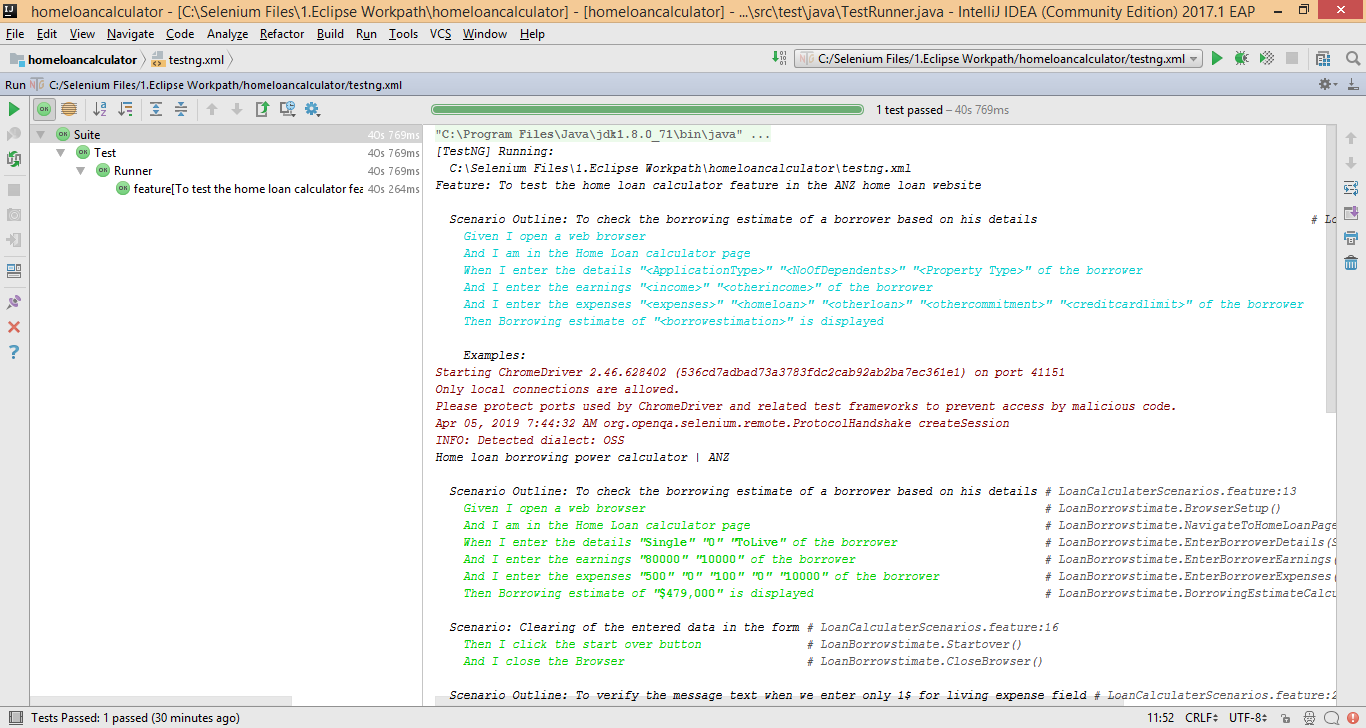
1. Cucumber

**Steps to run the Project:**

* Under the features folder we have the feature file stored which contains the test scenarios.
* The page factory folder contains all the Web elements in a structured format which we are using in our test.
* Step definitions containing the test steps matching to the scenarios from the feature file.
* The test runner is heart of the project. We can either run the test by running the TestRunner or running the testing.xml file

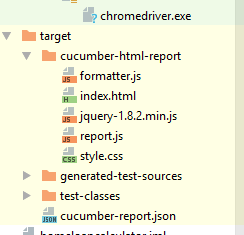


Once the test is run, the status of the run should be displayed like this below. For our test cases, all of the scenarios should pass.

Screenshot of the test run:

To view the reports, In your project structure , there is a folder called *target*. Inside *target* folderour cucumber html report will will be generated.

Screenshot of the target folder structure:



On opening the index.html in a browser, the test results will shown by scenarios wise, how it has fared.

Screenshot of the test results:

