Selenium Automation Framework:

Selenium WebDriver is a collection of open source APIs which are used to automate the testing of a web application. We can use multiple programming languages like Java, C#, Python etc to create Selenium Test Scripts. I have used Java + Selenium WebDriver to automate the frontend webpage application.

Environment:

- Java version 1.7
- Selenium WebDriver 2.44
- Maven:

Maven is a powerful project / build management tool, based on the concept of a POM (Project Object Model) that includes project information and configuration information for Maven such as construction directory, source directory, dependency, test source directory, Goals, plugins, etc.

Maven in this automation framework is mainly used to manage dependency.

```
<properties>
    <aspectj.version>1.8.3</aspectj.version>
    <allure.version>1.4.6</allure.version>
    <compiler.version>1.7</compiler.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
<dependencies>
    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>6.8.8</version>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>2.44.0</version>
    </dependency>
</dependencies>
<build>
    <resources>
        <resource>
            <directory>src/main/resources</directory>
            <filtering>true</filtering>
```

Selenium webDriver version 2.44.0 is used in this framework. If we would like to use another webdriver version, it is enough to update the version in this pom.xml file. Maven will download the required jar file.

TestNG 6.8.8:

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use.

TestNG.xml file is a configuration file that helps in organizing our tests. It allows testers to create and handle multiple test classes, define test suites and tests.

It makes a tester's job easier by controlling the execution of tests by putting all the test cases together and run it under one XML file.

Please find the testng.xml used in this automation framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="All Tests" verbose="5" data-provider-thread-count="1">

    <test name="QATool">
        <classes>
            <class name="pages.QATool">
            </class>
        </classes>
    </test>
</suite>
```

All testcases are written in QATool.java file.

Automation Framework Design:

Page Object Model (POM) is a design pattern, popularly used in test automation that creates Object Repository for web UI elements. The advantage of the model is that it reduces code duplication and improves test maintenance.

Under this model, for each web page in the application, there should be a corresponding Page Class. This Page class will identify the WebElements of that web page and also contains Page methods which perform operations on those WebElements.

Since Application Under Test (AUT) consists of only one webpage, I have used single java file (QATool.java) to write all testcases.

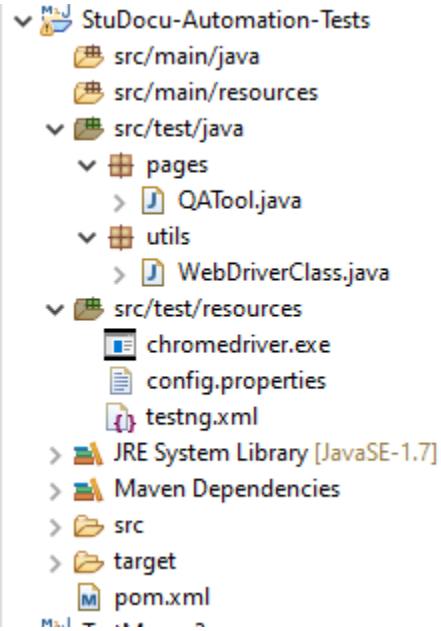
Singleton class

In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time. After the first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created. So whatever modifications we do to any variable inside the class through any instance, affects the variable of the single instance created and is visible if we access that variable through any variable of that class type defined.

I have use WebDriverClass.java as Singleton class to write reusable methods.

Folder Structure:

Folder structure of the Automation Framework is given below,



All the resources and config files such as chromedriver, config.properties and testng.xml are available in src/test/resource folder.

Utility and reusable java files are placed in src/test/java/utils. QATool.java which is a main test java file is created in src/test/java folder.

Testcases:

Three testcases are written to cover all user validations.

Testcase 1: testVerifyUI

This testcase verifies all UI texts in the page. verifyTextIntheWebElement method is reusable method which will perform UI text validation. It accepts two parameters. First parameter is xpath of the element and second parameter is expected UI text.

```
@Test(priority = 1)
public void testVerifyUI() {
    webDriverClass.verifyTextIntheWebElement(awesomeQAToolxpath, awesomeQAToolText);
    webDriverClass.verifyTextIntheWebElement(createdQuestionsXpath, createdQuestionsText);
    webDriverClass.verifyTextIntheWebElement(sortQuestionsBtnXpath, sortQuestionsBtnText);
    webDriverClass.verifyTextIntheWebElement(removeQuestionsBtnXpath, removeQuestionsBtnText);
    webDriverClass.verifyTextIntheWebElement(createnewQuestionXpath, createnewQuestionText);
    webDriverClass.verifyTextIntheWebElement(questionsXpath, questionsText);
    webDriverClass.verifyTextIntheWebElement(answerXpath, answerText);

}
```

Testcase 2: testCreateQuestions

This testcase creates 3 questions with answers. While creating each question, it verifies whether question is displayed in the webpage.

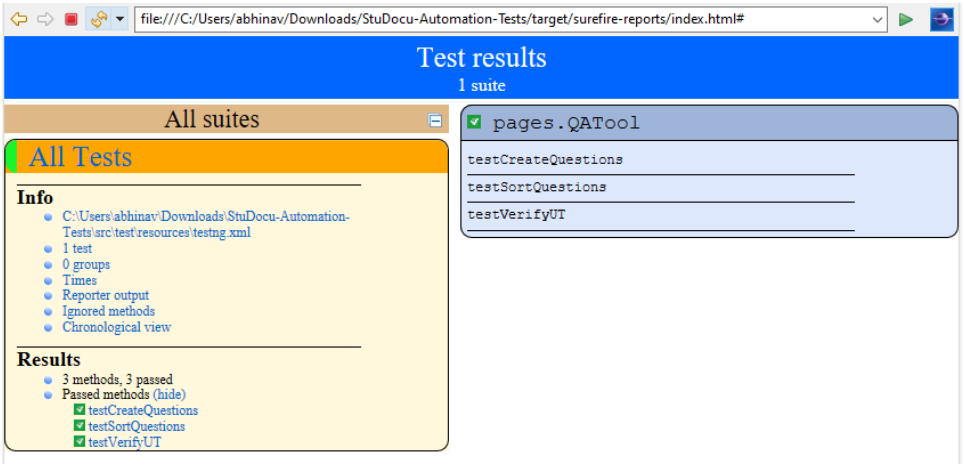
```
@Test(priority = 2)
public void testCreateQuestions() {
    String q1 = "C-Is this my first question?";
    String a1 = "Yes. This is my first question";
    String q2 = "B-Is this your second question?";
    String a2 = "Yes. This is my second question";
    String q3 = "A-Is this your third question?";
    String a3 = "Yes. This is my third question";
    //Click on "Remove Questions" button
    webDriverClass.clickOnWebElement(removeQuestionsBtnXpath);
    //Verify "No questions yet" message is displayed"
    webDriverClass.verifyTextIntheWebElement(noquestionsYetXpath, noquestionsYetText);
    //Creates first question (q1) with answer (a1)
    createQuestionsAndAnswers(q1, a1);
    //Creates second question (q2) with answer (a2)
    createQuestionsAndAnswers(q2, a2);
    //Creates third question (q3) with answer (a3)
    createQuestionsAndAnswers(q3, a3);
    //Finds the total no of questions
    int noOfQuestionsint = driver.findElements(By.xpath(listOfQuestions)).size();
    //Construct the UI text with total number of questions information
    String expectedText = "Here you can find " + noOfQuestionsint
            + " questions. Feel free to create your own questions!";
    //Gets the actual UI text
    String actualText = webDriverClass.getWebElement(noofQuestionsXpath).getText();
    //Verify whether expected text is displayed
    Assert.assertEquals(actualText, expectedText);
}
```

Testcase 3: testSortQuestions

This testcase verifies the sorting functionality.

Test report:

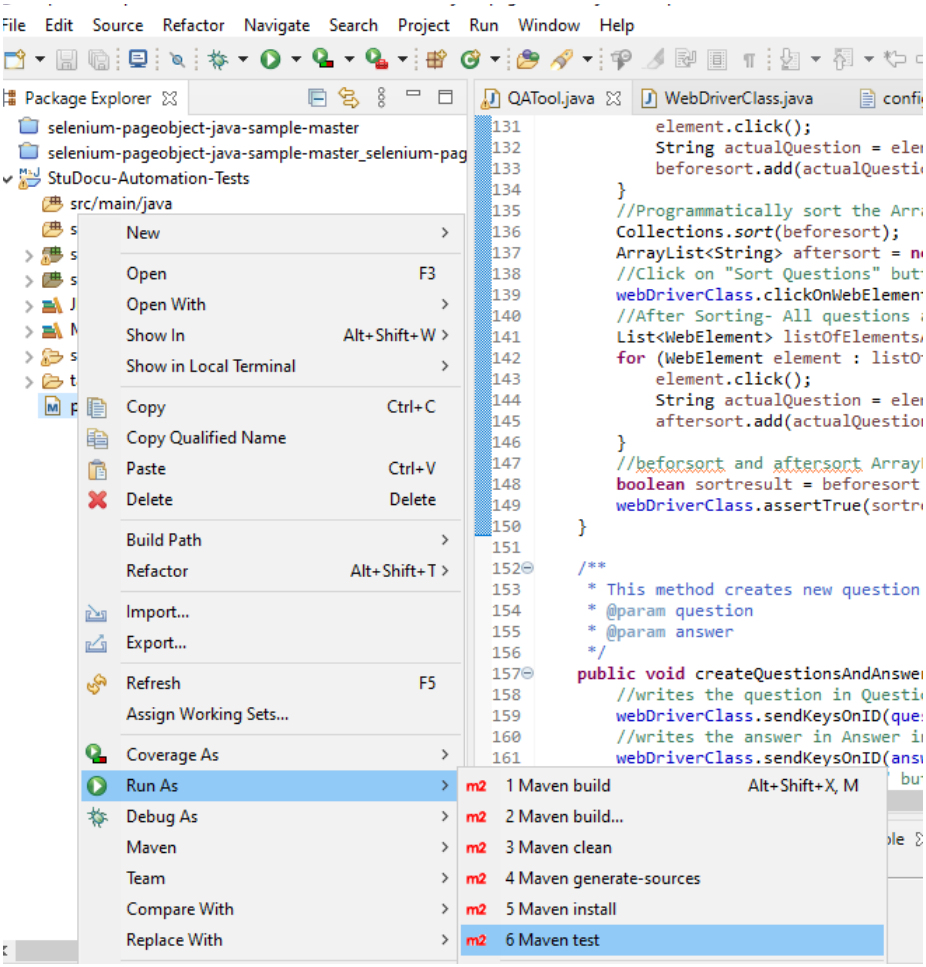
I have used the default report generated by TestNG.



How to execute the testcase:

There are several ways to execute the testcases.

If Eclipse is configured, we can execute pom.xml as below.



We can use mvn test command to execute the pom.ml file

We can run the java file as below,