Test Automation – A Guide to Test Development & Execution

|  |  |
| --- | --- |
| Author: | Veera |
| Version: | 0.1 |
| Date: | 09/26/2017 |

**Contents**

**1.** **Introduction** 3

**2.** **Framework Overview** 3

**3.** **Directories and Packages Hierarchy** 3

**4.** **Test Script Development** 5

**5.** **Test Execution** 6

**6.** **Post Execution - Analysis** 7

**7.** **Test Script - Maintenance** 7

**8.** **UML – Class Flow Diagram** 8

1. **Introduction**

This document provides the information and guidance for scripting, execution and maintenance of the automated test scripts using automation framework developed for Digital Automation. It elaborates not only the steps involved in development of test script, but also the best practices to be followed along the process

1. **Framework Overview**

The automation framework developed to test Digital Applications is hybrid-driven where TestNG @Test annotation tends to control the flow of the tests and a relevant test data used for the tests can be read from MS Excel primarily. Java dependencies are being managed by Maven which is also used for building automation project. Design pattern such as PageObject and PageFactory are followed throughout the framework for test scripts development. The below table summarises the tool stack used in the framework.

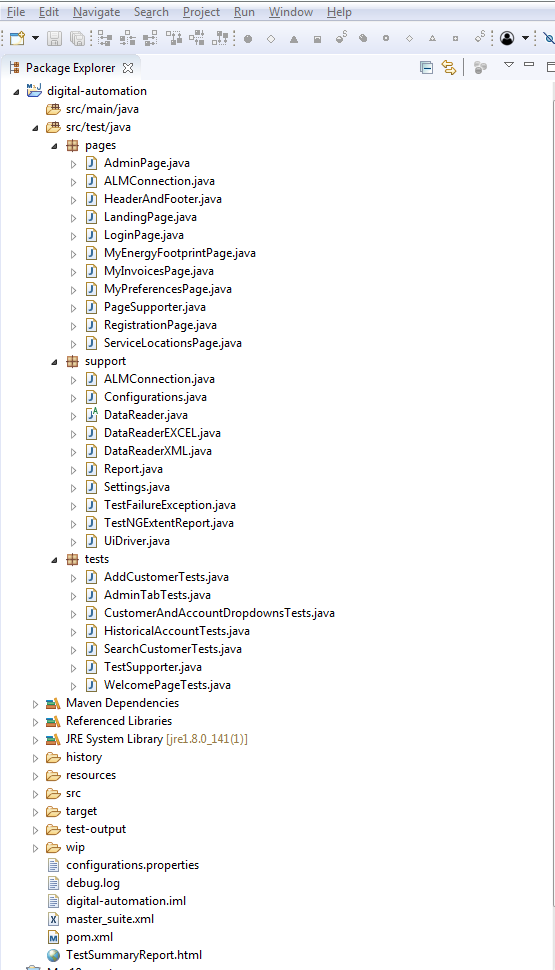
|  |  |  |
| --- | --- | --- |
| **Framework/Library** | **Usage Description** | **Version** |
| Selenium WebDriver | Java API which controls Web browser and emulate user interactions | 3.8.1 |
| TestNG Framework | It provides features for organising and running tests in a flexible manner | 6.8 |
| Maven | Dependencies and Build Management | 4.0.0 |
| Extent Reports | Open source Java library which provides API for reporting selenium-based tests | 3.0.6 |
| POI | Open source Java library which gives API for reading MS EXCEL files | 3.16 |
| Eclipse | IDE for developing and executing test scripts using Java | 4.7.1 |

1. **Directories and Packages Hierarchy**

The source directory mainly contains packages to distinguish test methods, page objects and common utilities. All other resources such as driver exe files, test data source will be placed inside the resource director. Maven POM XML, TestNG XML and configuration properties are placed at the root of the project directory.

|  |  |
| --- | --- |
| **Directory/File** | **Description** |
| *src/test/java/pages* | Java package that holds page objects and page methods |
| *src/test/java/tests* | Java package that holds Java class files grouped under functionality, which has TestNG test methods annotated with @Test |
| *src/test/java/support* | A collection of Java class files which provide features for reading data from external files, generating test reports etc. |
| *master\_suite.xml* | TestNG suite file that controls the execution flow |
| *pom.xml* | Maven POM file that maintains the dependencies and helps in build process |
| *TestSummaryReport.html* | HTML report that has details on recent automated test execution |
| *history* | The directory where history of previous execution report will be maintained |
| *confirgurations.properties* | Any test parameters can be configured if it needs to be. This is required only if we have to override any existing configurations |

The snapshot of sample project developed using this framework has been shown in the next page



**Pages where Page Elements and Page methods will be developed and maintained**

**Java files to provide re-usable methods which can be accessed across projects and tests**

**The packages to place all the TestNG annotated TEST methods**

**The directory where all driver exe files, screenshots used in the report, data files**

**TestNG suite file to control the tests execution**

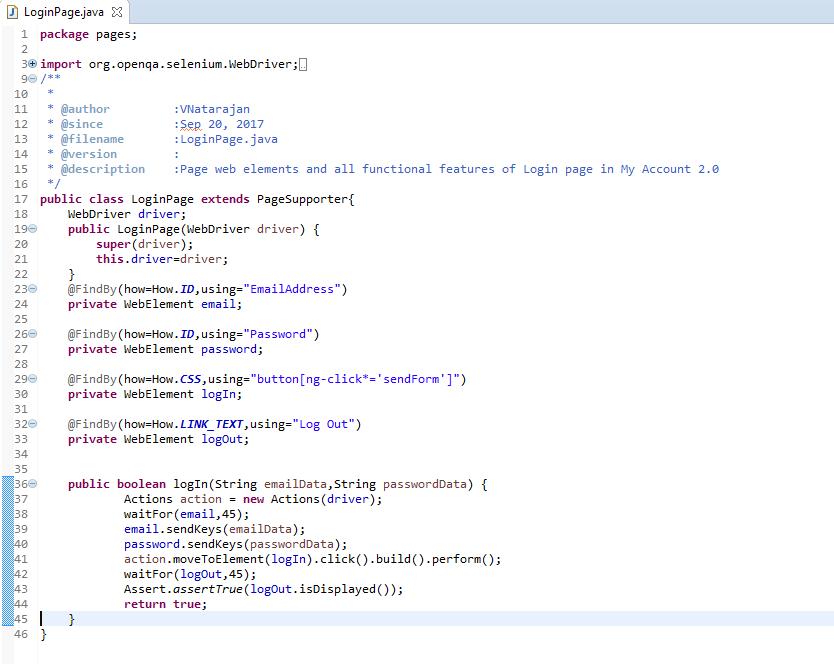
**Test Execution Summary Report generated after every test**

1. **Test Script Development**

As this framework has been based out of best open source practices, it’s important to follow the same guiding principles we have applied developing it. The steps involved in test script development is illustrated as following.

**Step 1 :** A Java class has to be created to capture all web elements in the application page. And it also has methods that perform a specific task in the page.

**Step 2 :** The newly created class needs to extend the class - *PageSupporter* that comes with framework to re-use the methods to handle UI interactions.

The below code snippets show the page class created for a login page of an application 

**Step 3 :** After creating the page,the class has to be created under *tests* package to develop the test methods annotated with TestNG @Test

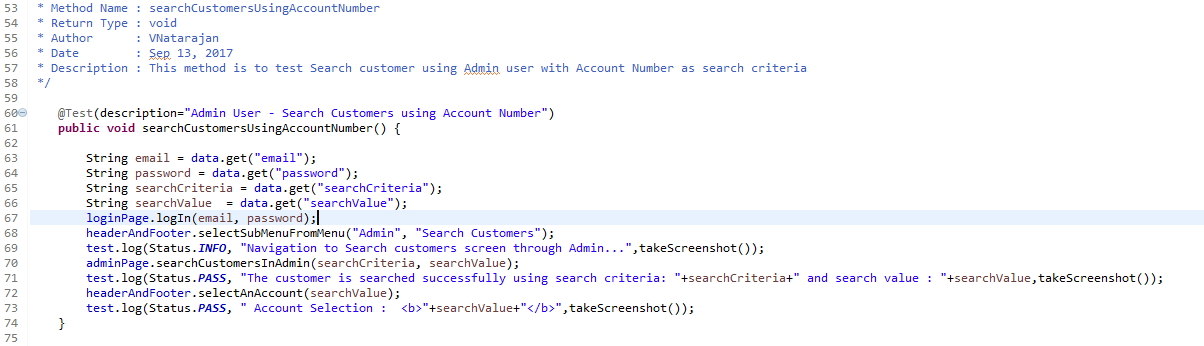
**Step 4 :** The classes inside the *tests* package has to extend the class *TestSupporter* that provides methods for logging test results and for reading data.

**Step 5 :** The test annotated methods has to be given with short *description*,stating functional flow or the test case name that the automated test covers.

**Step 6 :** The test data can be accessed within the test methods by calling the method -*get* using reference- *data.*For instance,the data in the column name “Example” can be read with the statement: *data.get(“Example”);*

**Step 7 :** The test result can be logged within the test methods by calling the method -*log* using reference- *test.*For instance,the test result “The successful login” can be printed in the HTML report with the statement: *test.log(“The successful login”);*

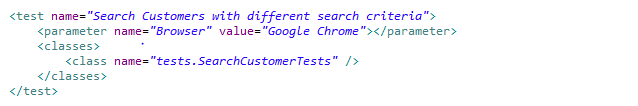
The below code snippets show the page class created for a login page of an application



**Step 8 :** The instance variable for the page class has to be intialized and the page object has to be instantiated inside the method *initializeBrowser* annotated with @BeforeMethod inside *TestSupporter*

**Step 9 :** As test annotated methods are executed in the class level,the class name where methods reside has to be updated within the tag *test* of the *TestNG* suite xml.

The below code snippet shows a part of TestNG suite which has class name and its associated package within *test* tag



**Step 10 :** Test data required for the automated tests are being supplied by MS Excel where each row of data is for a test.Test description given in the test method must be same as the one in the “Test Description” column of the file.

1. **Test Execution**

* The data which will be used during the execution need to be updated in the Excel file in the relevant sheet
* It is configurable in a way that all tests can share same sheet or every test can have a separate sheet. This can be achieved by updating TestNG parameters in the TestNG suite file.



If above statements are placed on the top of the suite file, all tests share the same workbook(*Datasource*) and workbook (*SheetName*) and on the other hand, when it is placed inside the *test* tag of the suite file, only that test receive data from the *Datasource* and *SheetName.*

* Browser name and number of thread to be used for the execution can be updated in the TestNG suite file.
* Optionally all class names of the test methods can be grouped within the test tag or any such logical grouping can be achieved through suite file.

1. **Post Execution - Analysis**

* Once the execution is completed, the report in HTML format that captures all steps during the execution, which can be used to trace the execution flow.
* Every step has status such as INFO,PASS,WARNING and FAIL and it also captures the screenshot relevant to it.
* Test Summary Report in the project directory has only latest execution logs, and all the previous execution reports can be found at *history* folder.

1. **Test Script - Maintenance**

* Any changes in the application functionalities can be remediated by modifying or creating methods under all impacted page classes, keeping all other pages unchanged.
* Test failures due to changes in the web element properties can be handled by updating properties of the them in Page classes.

1. **UML** – Class Flow Diagram

