ReadMe for Gurukula Test Assignment task

Author: Chandra Sekhar Muttineni

The Automation approach followed, and tactics used, Items not tested etc.,:

i) Problem statement:
Develop Automation framework for the Gurukula Application built using Java.

ii) Feature Scope of Automation:
> Authentication of users on the application.
> Registration of new users.
> Reset User credentials in case of forgot password.
> Create, View, Edit and Delete Branches.
> Pagination of Branches and Staff.
> Create, View, Edit and Delete Staff members.
> Edit Settings of logged in user.
> Edit Sessions available for a user.
> Update Password of logged in user.
> Logout from the application.

iii) Desired Capabilities of the Automation Framework:
> Automation framework under development should be easy to apprehend and use.
> The framework should support extensibility in terms of new requirements like multiple browser testing.
> The framework should support the addition of new input data set. The tests should be data driven, to cover any new or extended data domains.
> The framework should be Modular with regards to the usage.
> It should be as simple as just the addition of few steps to create new automation of features. Should not be tiresome to create or re-write the automation.
> No tweaking should be required to extend support to new test cases. Strict NO to tweaking.

iv) Technology stack used in this Automation Framework:

Java - Coding language used
TestNG - Basic Testing framework
Selenium WebDriver - Web UI Automation Framework
Apache POI HSSF - Test Data Handler utility
Apache Maven - Build System

v) Automation Test Approach:

To achieve the attributes like **maintainability, readability and re-usability in the framework**, I have used the **Page Object Model (POM) design pattern** while constructing every block of this framework.

The approach, according to the POM pattern is simple. Create the pages once and reuse them any number of times. According to this, I have created pages like 'LoginPage.java', 'LoggedInPage.java', 'Branch.java' etc.,. and so on. All these Pages would extend **'BasePage.java'** that will have all the common and required elements, functionalities for any page. This page can be extended as required, providing the extensibility of the framework.

There would be corresponding tests to each page and **DataProviders** for each test. This helps in achieve the data driven nature for our tests.

All the Test Data would reside in '**TestData**' package, in the form of excel. We have the TestDataHandler.java utility class, which will deal with reading of test data from the data files. The **TestDataHandler** would be called by the DataProviders.java, which will have all the data providers initialised, equipped with data for use by TestNG tests.

As part of test data, we also have the environment variables mentioned in **TestConstants.java**.

As the test automation needs to comply with different sets of data and needs to be flexible enough to support the change in input data domain, I have used the DataProvider to realize the Data Driven tests like Login, Registration etc.,.

I have created the **Test utility classes** involving the WebDriver Wrapper utility and TestUtil classes. The WebDriver Wrapper Utility class provides us the re-usability of WebDriver actions like finding an element in given Web Table, Selecting items from dropdown etc.,. and so on. The **TestUtil** will help in supporting the common functionalities like **Logging into the context** for any given test.

vi) How easy or difficult to add new tests to the test:

Steps are as below.

>Create a new page object class
>Create a new context by adding a new enum element
>Add data excel files, if any required
>Create a new test class

vii) How easy or difficult to maintain existing tests:

Steps are as below.

>As I have followed the "**Single Responsibility Principle**", any failures due to flakiness or changes, would be easily identified and isolated for fix.

viii) Items not covered in this assignment:

The following test items are not tested:

1.Globalization or Internationalization (I18N) testing.
2.Performance/Stability/Scalability testing
3.Cross Browser functional Automation testing is not taken care as of now. However, the automation framework developed can be enhanced to achieve this aspect of quality.
4.Security Testing for vulnerabilities like XSS, CSRF, SQL Injection etc.,

ix) How to invoke tests:

Please start the Gurukula application in port 9099 as follows.
java -jar ~/Desktop/gurukula/gurukula-0.0.1-SNAPSHOT.war --server.port=9099

The following 3 ways can be employed to invoke the automation tests.

1)We can run '**mvn test**' from command prompt.
Pre-conditions:
Navigate to the folder where 'pom.xml' is available.
Maven and Java binaries available in $PATH

2)We can run '**java org.testng.TestNG testng.xml**' from command prompt.
Pre-conditions:
Navigate to the folder where 'testng.xml' is available.
Maven, Java and Gurukula test binaries available in $PATH

3)Import the project as 'existing Maven project' to any Java IDE like **Eclipse**. Have TestNG installed in the IDE. Right click 'testng.xml' and select **'Run As' -> 'TestNG Suite'**