Overview of

Perficient GDC Selenium Framework

**TABLE OF CONTENTS**

1. INTRODUCTION
   1. *Purpose of this document*
   2. *Scope*
   3. *Structure of framework*
2. STEPS TO IMPORT GIVEN PROJECT
   1. *Import project in Eclipse*
   2. *Maven configuration*
   3. *TestNG configuration*
3. TECHNOLOGIES USED IN FRAMEWORK DESIGN
   1. *TESTNG*
   2. *MAVEN*
   3. *Page object model with page factory*
   4. *Properties files*
   5. *Log4j*
4. WORKING OF FRAMEWORK
   1. *Sample project*

5. REPORT GENERATION

5.1 *HTML report using ExtentReports Jar*

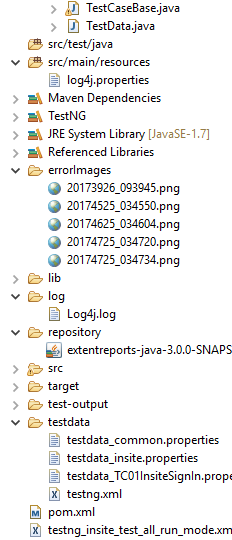
**1. INTRODUCTION**

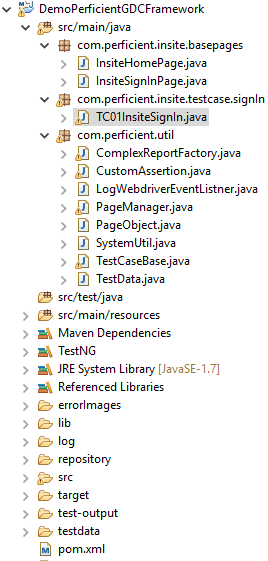
* 1. Purpose of this document

This document gives a working idea about the ‘Perficient GDC Selenium framework’. It provides guidance and template material which is intended to assist the relevant technical staff.

* 1. **Scope**

1. Simple and effective automation using the page object model.
2. Annotation based test execution management support.
3. Test reports and coverage report generation.
4. Selenium webdriver integration and ability to run UI tests to cross browser environments.
5. Comprises of maven as a build automation tool and TestNG.
6. Logs generation for every test case execution.
   1. **Structure of framework**





**2. STEPS TO IMPORT GIVEN PROJECT**

2.1 **Import project in Eclipse**

a. First we need to import the project from GitHub repository.

b. Follow the below link to import the project:

<http://agile.csc.ncsu.edu/SEMaterials/tutorials/import_export/>

2.2 **Maven configuration**

a. Make sure your system configures with Mavenenvironment setup.

b. To check go to CMD (Command Prompt) and type *mvn -version.*

c. If this command gives error, then configure your system for maven as explained in below link:

<https://www.tutorialspoint.com/maven/maven_environment_setup.htm>

d. Install Maven plugin for Eclipse from Eclipse marketplace as given below:

1. Open Eclipse
2. Go to Help -> Eclipse Marketplace
3. Search by Maven
4. Click "Install" button at "Maven Integration for Eclipse" section
5. Follow the instruction step by step

After successful installation do the followings in Eclipse:

1. Go to Window --> Preferences
2. Observe, Maven is enlisted at left panel

Finally,

1. Click on an existing project
2. Select Configure -> Convert to Maven Project

**2.3 TestNG configuration**

a. Install TestNG plugin for Eclipse from Eclipse marketplace as given below:

|  |  |
| --- | --- |
|  | 1. Open eclipse 2. Go to Help -> Eclipse Marketplace... 3. Do search for TestNG (Type the text TestNG in Find text box > Click Go button) 4. After searching: Click Install button at TestNG for Eclipse area 5. Follow the further instructions by eclipse   After successful installation: Go to **Window -> Preferences**. TestNG is enlisted at the left panel in pop-up. |

1. **TECHNOLOGIES USED IN FRAMEWORK DESIGN**

**3.1 TESTNG**

a. TestNG is a testing framework designed to simplify a broad range of testing needs.

b. Here, we are using TestNG to run test cases on different browser parallelly.

c. To configure ‘testng\_insite\_test\_all\_run\_mode.xml’ with pom.xml.

**3.2 MAVEN**

a. We need maven to get all our dependencies automatically, which also allows users to reuse same jars across multiple projects.

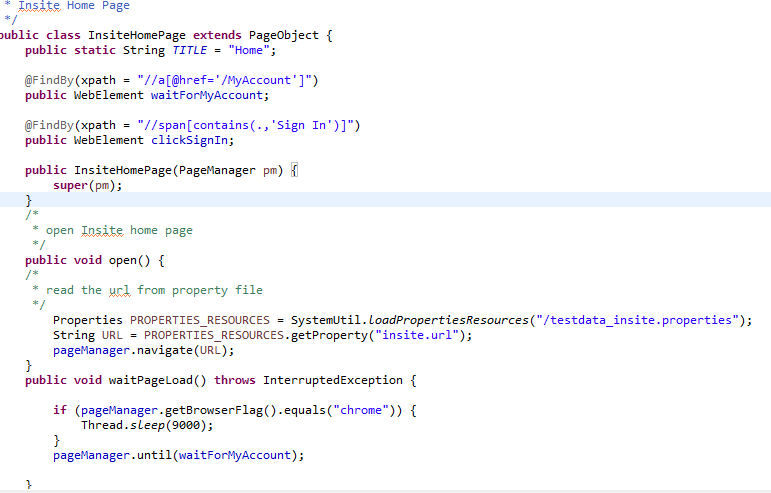
b. Every engineer in a project use the same jar dependencies due to the centralized POM.

c. It provides the complete structure with naming conventions which is easy to locate and execute tests.

d. We can automate the complete build procedure.

**3.3 Page object model with page factory**

* 1. A better approach to script maintenance is to create a separate class file which would find web elements, fill them or verify them. This class can be reused in all the scripts using that element. In future, if there is a change in the web element, we need to make the change in just 1 class file and not 10 different scripts.
  2. This approach is called **Page Object Model(POM)**. It helps make the code **more readable, maintainable**, and **reusable**
  3. **Page Factory** is an inbuilt Page Object Model concept for Selenium WebDriver but it is very optimized.
  4. Here as well, we follow the concept of separation of Page Object Repository and Test Methods. Additionally, with the help of PageFactory class, we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements
  5. **@FindBy** can accept **tagName, partialLinkText, name, linkText, id, css, className, xpath**as attributes.
  6. Let's look at the same example as above using Page Factory



**3.4 Properties files**

a. In this project, we are using .**properties** files to pass test data in test cases.

b. **.properties** is a file extension for files mainly used in Java related technologies to store the configurable parameters of an application. They can also be used for storing strings for Internationalization and localization; these are known as Property Resource Bundles.

c. Each parameter is stored as a pair of strings, one storing the name of the parameter (called the *key/map*), and the other storing the value.

**3.5 Log4j**

a. log4j is a reliable, fast and flexible logging framework (APIs) written in Java.

b. In this project, we are using log4j for logs generation with respect to the test case name.

**4. WORKING OF FRAMEWORK**

**4.1. Sample project (DemoPerficientGDCFramework)**

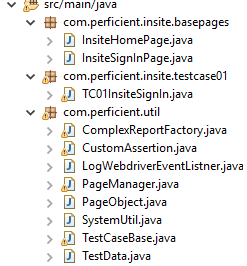
**[ Note:** *We need to specify properties file name into the* ***testdata*** *folder as ‘****testdata\_Test Case name’*** *to insert data into the current test case.*

*e.g.: Test case name:* ***TC01InsiteSignIn*** *then file name will be ‘testdata\_TC01InsiteSignIn.properties’*  **]**

In this project, we have two source folders as given below:

* 1. src/main/java
  2. src/main/resources

1. **src/main/java**



**A**. package: ***com.perficient.insite.basepages***

This package having base pages of the test case: ***TC01InsiteSignIn.java*** as:

* 1. ***InsiteHomePage.java***

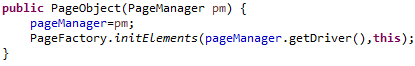
This class perform all the operation on Insite home page by using page object model with page factory.

Step wise working of this class as follows:

* + 1. *InsiteHomePage.java* classextends by *PageObject.java* class

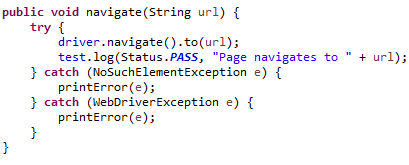
**public** **class** InsiteHomePage **extends** PageObject

* + 1. PageObject.java class manage connection with PageManager.java class.



* + 1. *PageManager.java class* having all the useful methods which calls from test case base pages.

e.g.:



4. In *InsiteHomePage.java* class, we are using page factory annotations to pass the value of locator for WebElement.

e.g.:

*C:\Users\niklesh.bahad\AppData\Local\Microsoft\Windows\INetCacheContent.Word\2017-04-27 13_56_41-workspace - Java - DemoPerficientGDCFramework_src_main_java_com_perficient_insit.png*

5. And we are also building new methods to perform operation on same page (*InsiteSignInPage*) but this method also connected with the *PageManager.java* class to called defined method.

e.g.:

C:\Users\niklesh.bahad\AppData\Local\Microsoft\Windows\INetCacheContent.Word\2017-04-27 13_57_53-workspace - Java - DemoPerficientGDCFramework_src_main_java_com_perficient_insit.png

* 1. ***InsiteSignInPage.java***

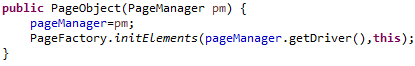
This class perform all the operation on Insite Sign In page by using page object model with page factory.

Step wise working of this class as follows:

* + 1. *InsiteSignInPage.java* classextends by *PageObject.java* class

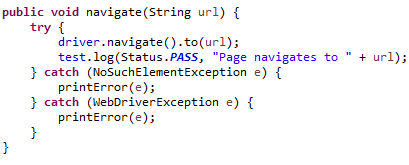
**public** **class** InsiteSignInPage **extends** PageObject {

2. PageObject.java class manage connection with PageManager.java class.



*3. PageManager.java class* having all the useful methods which calls from test case base pages.

e.g.:



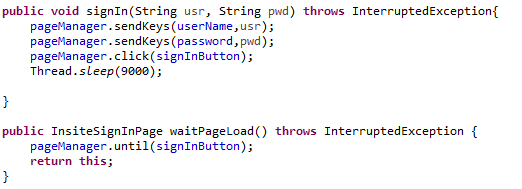
4. In *InsiteSignInPage.java* class, we are using page factory annotations to pass the value of locator for WebElement.

e.g.:

C:\Users\niklesh.bahad\AppData\Local\Microsoft\Windows\INetCacheContent.Word\2017-04-27 12_56_15-workspace - Java - DemoPerficientGDCFramework_src_main_java_com_perficient_insit.png

5. And we are also building new methods to perform operation on same page (*InsiteSignInPage*) but this method also connected with the *PageManager.java* class to called defined method.

e.g.:



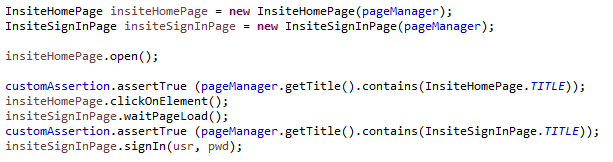
**B**. package: ***com.perficient.insite.testcase.signIn***

1. This package having the test case as: ***TC01InsiteSignInPage.java*** class.

2. TC01InsiteSignInPage.java class extends by TestCaseBase.java as:

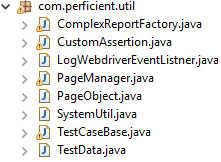
**public** **class** TC01InsiteSignIn **extends** TestCaseBase{

1. By using TestNG annotation ***@Test*** we are running this test case.
2. And calling every class from ‘*com.perficient.insite.basepages*’ package to perform relative operation.
3. Also, calling relative method belong to classes.



C. package: ***com.perficient.util***

1. This package having all the utilities related classes used in this project as:



* 1. ***ComplexReportFactory.java*** class:

This class having the *.html* report (extent report) management related code.

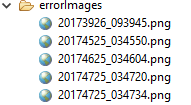
ExtentReport is an HTML reporting library for Java, which can be used with Selenium WebDriver.

We are using this tool within our TestNG automation framework.

ExtentReport is a simple to use tool to make excellent execution reports.

* 1. ***CustomAssertion.java*** class

This class contains all the assert condition related code to capture screenshot of failed test case and store it into the folder *errorImages* with respect to their date of execution as:



* 1. ***LogWebdriverEventListner.java*** class

In this class, we get to know all the events triggered by webdriver.

It also plays an important role in analyzing results and helps us in debugging issues if we encounter any.

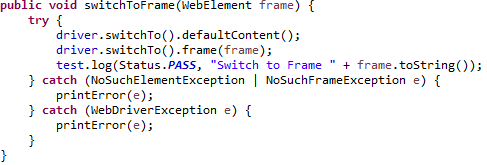
Selenium webdriver has ability to track different events such as '*beforeNavigateTo*' , '*afterNavigateTo*' ,

'*beforeClickOn*' , '*afterClickOn*', '*onException'* and so on. Whenever we develop test scripts we can write our own implementation for handling events during the execution.

* 1. ***PageManager.java*** class

In this class, all the page handling related common methods present to perform operation on respective web page.

e.g.:



* 1. ***PageObject.java*** class

This class manage connection with PageManager.java class and with PageFactory method. Here as well, we follow the concept of separation of Page Object Repository and Test Methods.

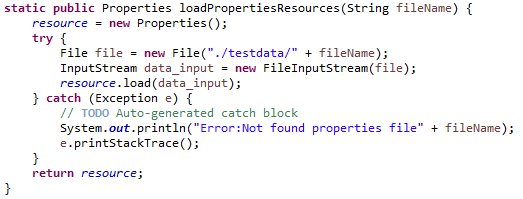
Additionally, with the help of PageFactory class, we use annotations @FindBy to find WebElement. We use initElements method to initialize web elements

e.g.:



* 1. ***SystemUtil.java*** class

This class used to load the test data in properties file as:

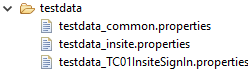
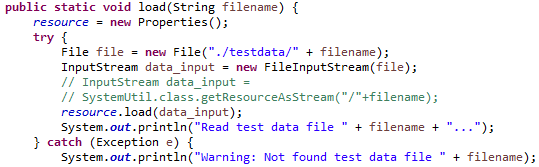


* 1. ***TestCasebase.java*** class

This class contains all the TestNG annotations except *@Test*. It is used to perform all the operation after and before test execution and passing TestNG parameters with respect to the *TestNG.xml* file.

* 1. ***TestData.java*** class

This class is used to load the test data file which is located into the testData folder as:

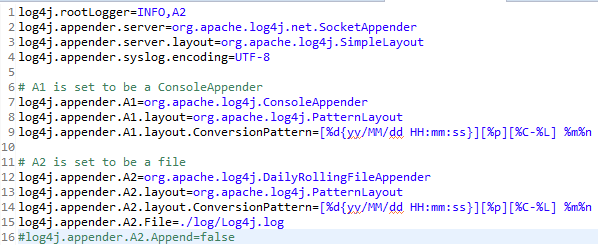


* 1. **src/main/resources**



* + 1. log4j.properties

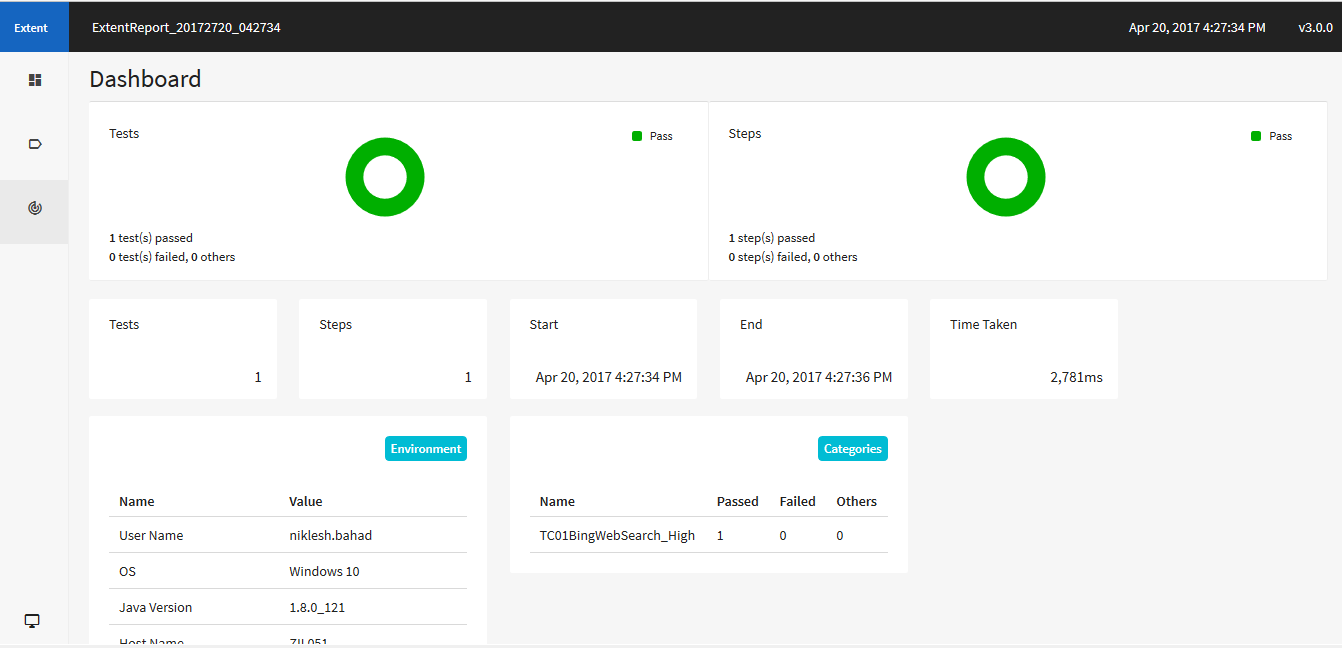
In this properties file, we defined all the log4j related configuration with respect to the project specification as:



**5.REPORT GENERATION**

**5.1** *HTML report using ExtentReports Jar*

This report generated by using *Extent report java* jar and *ComplexReportFactory.java* class.



*Thank You*

.