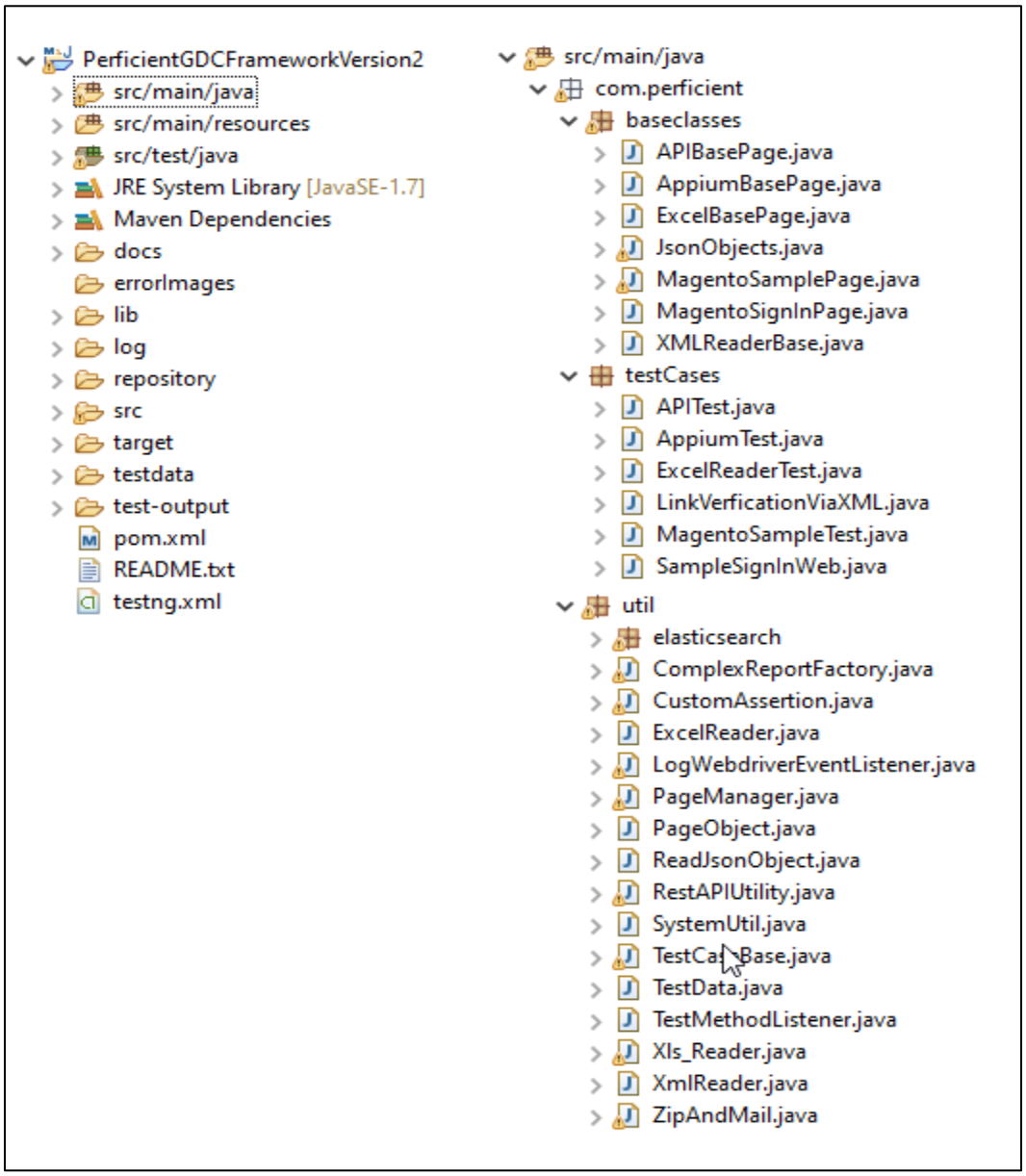
PERFICIENT
vision. execution. value.

GDC Selenium Automation Framework Usage

PERFICIENT
vision. execution. value.

Table of Contents

1. The GDC Selenium Automation Framework folder structure



1) **Folder – src/main/resource**:

This folder contains log4j.properties file. The log4j. properties file is a log4j configuration file which stores properties in key-value pairs. The log4j properties file contains the entire runtime configuration used by log4j. This file will contain log4j appenders information, log level information and output file names for file appenders.

2) **docs**:

This folder contains all the documents related to the Framework.

3) **repository:**

This folder has the jars which are used in the GDC Framework. The jars are Xls_Reader and extetntReport jars.

4) **testdata**:

This folder has all the properties files, excel and text files. All the required testdata is stored at this location.

5) **test-output**:

This folder contains the output files like error images and extent reports which are generated after the execution of testcases.

6) **Pom.xml:**

POM is an acronym for Project Object Model. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

7) **Readme:**

It is a text file that contains the information for the user about the software, project, code, or it might contain instructions, help, or details about the patches or updates. In GDC framework the readme file has updates that are done in Framework.

8) **testNg.xml:**

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.

9) **baseclasses:**

This package has java class. The java classes will extend the Pageobject class. This class perform all the operation on testpage class by using page object model with page factory. And it contains PageManger and Excelreader class object and CustomAssertion class object, with the help of this you can use any methods directly that are present in respective class.

The classes which are in baseclasses package contains locators (Xpaths, CSS and others) and the declarations of methods that will be used in the testclass.

10) testCases:

This package also contains java classes. The java classes under this package will extends the "TestCaseBase" class.

The classed under this package should contain @Test annotation and its methods, which will be executed at runtime. Also, assertions should be added in these classes only for the verification of testcase.

11) util:

This package contains all the Utility classes that will be used in the framework. The classes and its working are listed below.

a. ***ComplexReportFactory***:

This class contains ExtentReport generation and its configuration code.
ExtentReport is an HTML reporting library for Java, which can be used with Selenium WebDriver. ExtentReport gives simple and elaborated execution reports.

b. ***CustomAssertion***:

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

c. ***ExcelReader***:

This class contains the methods which deals with Excel sheets. It has methods of loading an Excel sheet, read and write methods of Excel related operations.

d. ***LogWebdriverEventListner***:

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.

e. ***PageManager***:

In this class, all the page handling methods are written which will be called in Baseclass to perform operation on respective web page. Such as clickByJavaScriptExecutor, sendkeys, waitForPageload, etc.

f. ***PageObject***:

This class has the collection of constructors which are used in baseclasses to call all the objects of concerned Utility class through which it forms connection with PageManager and PageFactory methods. 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

g. ***SystemUtil***:

This class has methods to load the test data in properties file and driver killer method.

h. ***TestCasebase***:

This class contains all the TestNG annotations except *@Test*. It has all the methods which are used for basic setup of framework like setupBrowser, teardown & setUpChromeWin32 is used to perform all the operation after and before test execution.

i. ***TestData***:

This class has all the methods to handle Testdata files which are located at testData folder like load, get and set test data methods.

j. ***ReadJsonObject***:

This class is used to load and read Json related data.

k. ***XmlReader***:

This class is used to load and read XML (Extensible Markup Language) related data.

l. ***ZipAndMail***:

This class contains the methods which deals with mail operation. It has the method which will convert all the reports in zip file and send the mail to the recipients.

m. ***ElasticListener***:

This class is a listener class for Elastic Search Utility. It listens to the testcase execution and sends the recorded data to the Elastic Dashboard IP.

n. ***ElatsicModel***:

This class has the structured code setup for Elastic Search and converting it to string.

o. ***ElasticSender***:

This class has the code to convert the data to Json and sends all the data to the Elastic Search.

p. ***RestAPIUtility***:

This class has methods which deals with the response codes, such as get, post, put, delete, and update.

2. Working of GDC Framework

1) Creating Base class

A Base class will contain all the locators and definition of the methods. Every Base class extends Page Object class. Because Page object class has instances of Page manager and Page factory model.

To create a Base class, right click on Baseclasses package and click on New and the select Class option. Give any relevant name as per your requirement. Ex: MagentoSignInPage

MagentoSignInPage.java

```
package com.perficient.baseclasses;

import org.openqa.selenium.WebElement;

public class MagentoSignInPage extends PageObject {
    public static String SIGNINTITLE = "Customer Login";

    @FindBy(xpath = "//div[@class='panel header']//a[contains(text(),'Sign In')]")
    public WebElement clickSignIn;


    public void open(String url) {
        pageManager.navigate(url);
    }

    public void clickOnSignIn() {
        pageManager.click(clickSignIn);
    }
```

2) Creating Test class

A Test class will contain @Test annotation and the object of the Base class, with the help of this object we can call the methods of utility classes. The Test class will extend the TestCaseBase class.

@Test - The test method annotated with @Test annotation tells TestNG that this method is a “test”, and it should be executed when the user runs the class.

To create a Test Class, right click on TestCases package and click on New and then select Class option. Give any relevant name as per your requirement. Ex: SampleSignInWeb

SampleSignInWeb.java

```
package com.perficient.testCases;

import com.perficient.baseclasses.MagentoSignInPage;

public class SampleSignInWeb extends TestCaseBase {

    @Test(groups = { "firefox", "IEWin32", "ChromeWin32", "browserPercentageSpecified" })
    public void signIn() throws Exception {

        String url = TestData.get("url");

        MagentoSignInPage magentoSignInPage = new MagentoSignInPage(pageManager, excelReader);
        magentoSignInPage.open(url);
        customAssertion.assertTrue(pageManager.getTitle().contains(MagentoSignInPage.HOMEPAGETITLE),null);

        magentoSignInPage.clickOnSignIn();
        customAssertion.assertTrue(pageManager.getTitle().contains(MagentoSignInPage.SIGNINTITLE),null);


    }
}
```

3) Creating Properties file

We need to specify properties file name into the testdata folder as, 'testdata_TestCasename.properties' to insert data into the current test case.

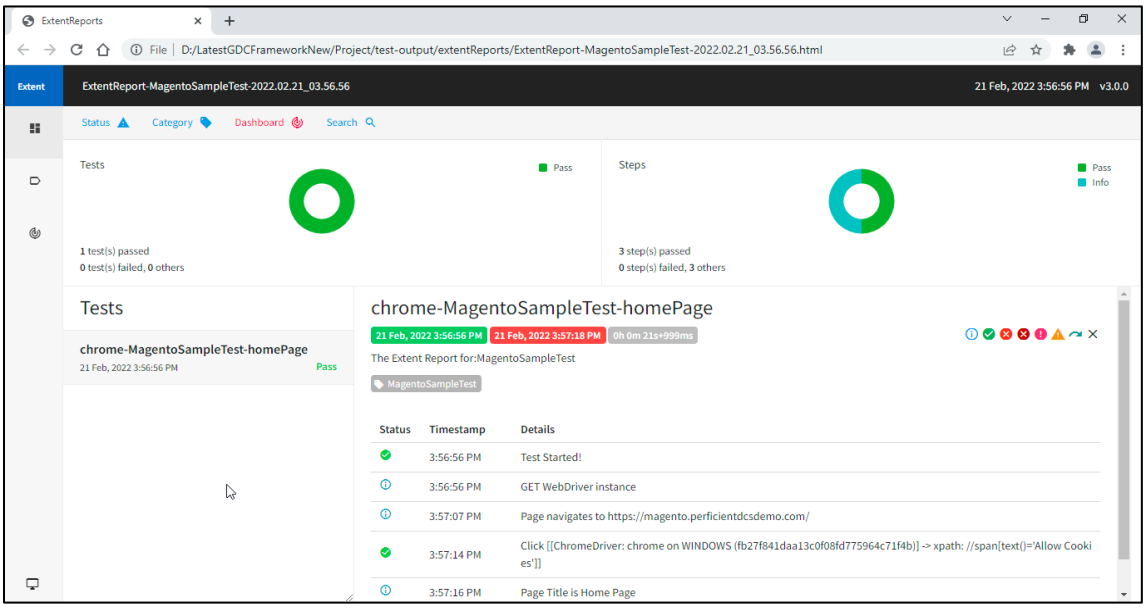
Ex: Testcase name: SampleSignInWeb then file name will be testdata_SampleSignInWeb.properties

testdata_SampleSignInWeb.properties

```
url=https://magento.perficientdcsdemo.com/
```

4) Report Generation

After execution of the testcase, Extent report gets generated in Testoutput folder.
(NOTE: If in case the report is not visible Refresh the project)



If you want the execution report should be available to all the team members. Then you need to uncomment the method i.e Zip and Method in Testcasebase class. also add the recipient email ids in mailreport.properties file.

TestCaseBase.java

```
@AfterSuite(alwaysRun = true)
public void afterSuite() throws ZipException {
    TestData.load("mailreport.properties");
    clientEmail = TestData.get("recepientEmailIds");
    mailAddress = TestData.get("recepientEmailIds");
    ComplexReportFactory.closeReport();
    // zipAndMail(); --- Uncomment this method to receive the email of the extent reports
}
```

mailreport.properties

```
fromEmailId=gdcperficientnagpur@gmail.com
Password=PerftNagpur#$1234
recepientEmailIds=xyz.abc@perficient.com,pqr.fgg@perficient.com
```

3. Diagrammatical representation of the Automation Flow