Yashwanth Kumar

Humana, Inc.

Selenium Java Framework

# **[Framework Architecture](#FrameworkArchitecutre)**

# **[Features of Framework](#FeaturesofFramework)**

# [**Benefits of Framework**](#BenefitsofFramework)

# [**Overview of Framework**](#_Overview_of_Framework:)

## [**TEST**](#Test)

## **[RESOURCES](#Resources)**

## **[Maven Dependency and JRE System Library](#MavenDependencyandJRESystemLibrary)**

### **[Selenium-standalone-server-jar](#Selenium)**

### [**Selenium web driver Jar Files**](#seleniumwebdriver)

### [**Testing framework Jar Files**](#testingjar)

### [**Apache POI selenium jar Files**](#apache)

### [**POM.xml**](#pom)

# [**Main Components of Framework**](#_Main_Components_of)

## [**Src/test/java**](#srcjava)

### [**Allocator Package**](#allocator)

### [**Business Components Package**](#businesscomponents)

### [**Framework Core Package**](#frameworkcore)

### [**Framework Selenium Package**](#frameworkselenium)

### [**Framework Selenium Core Package**](#frameworkseleniumcore)

### [**Pages Package**](#pages)

### [**Test scripts Package**](#testscripts)

## [**Src/test/resource**](#srcresource)

### [**Data tables Folder**](#datatables)

### [**Property File**](#property)

### [**Run Manager**](#runmanager)

# **[Results folder](#first_heading)**

# [**Test-output folder**](#Testoutputfolder)

# [**How to Create and Run a Sample Test Case**](#Howtocreateandruntestcase) **for Web Application**

## [**Mode of Execution**](#modeofexecution)

# [**How to Create and Run a Sample Test Case for API**](#Howtocreateandruntestcaseapi)

## [**Main Components of API**](#maincomponentofapi)

## **[Mode of Execution of API](#modeofexecutionapi)**

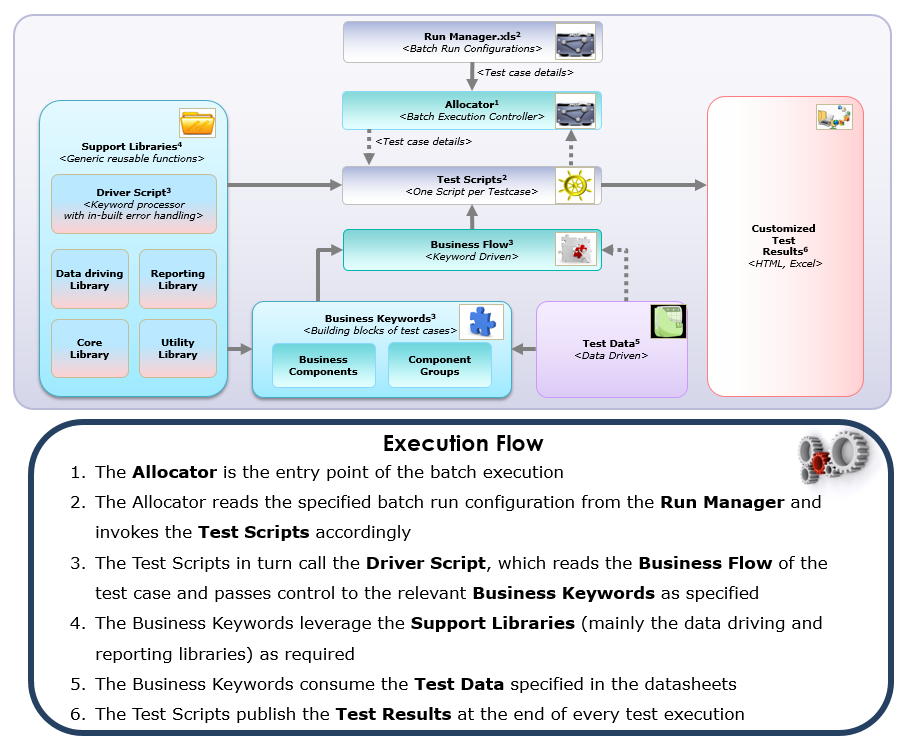
## **[REST and SOAP UI Implementation](#restsoap)**

# [**Summary**](#summary)

# [**Steps to Execute the tests After Completing Design**](#stepsto)

# [**Mode of Execution**](#executionmode)

**[Framework Architecture](#FrameworkArchitecutre)**



**Features of Framework**

* Hybrid framework approach (Keyword + Data driven).
* Test cases are broken down into business keywords.
* Keywords are strung together in an Excel sheet to form automated test cases based on the business flow.
* Centralized test data in Excel, with provision to reuse any data common to multiple test cases.
* Batch execution engine.
* Provides support for a wide range of browsers and platforms.
* Supports script development using Java (apart from a host of other programming languages).
* Facilitates distributed execution of test cases through Selenium Grid.
* Out-of-the-box integration with TestNG and Selenium Grid.

**Benefits of Framework**

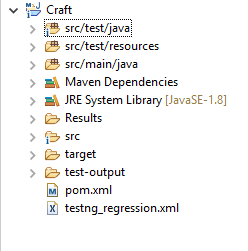
* Simplifies the process of creating end-to-end automated scripts and improves the automation quality.
* Ensures increased reusability and reduced maintenance effort.
* Enables early automation in parallel with the application development.
* Enables multi-platform and multi-browser testing of web applications.
* Enables parallel execution of scripts on multiple machines to reduce the cycle time.

# **[Overview of Framework](#_Overview_of_Framework)**

The framework was implemented using Maven and is configured to test the Web applications, API Web services and Mobile Applications.

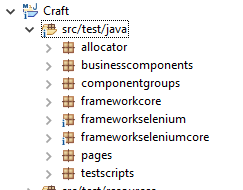
Maven Project structure has two main Folders:

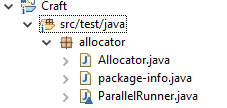
* Test
* Resources



**TEST**:

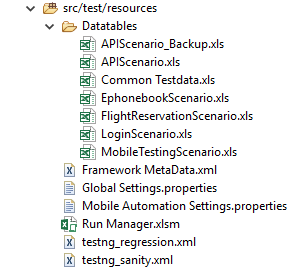
In src/test/java folder we have packages like allocator, business components, frameworkcore, frameworkselenium, frameworkseleniumcore, pages and testscripts.



Each package has a set of java class files and each file perform different functions.

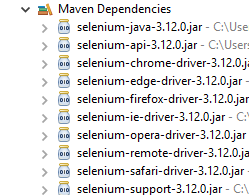
**RESOURCES**:

In src/test/resources we have excel spreadsheet where all the input needed for test execution are placed in this folder. Each excel spreadsheet has separate input values for different type of testing being done like Web Service, Web, mobile etc..,



**Maven Dependency and JRE System Library:**

All the Jar files needed for the project are added in Maven Dependency and JRE System Library.



**Selenium-standalone-server-jar:**

We are using Selenium Standalone Server jar version 2.39.0 in our project. Selenium Standalone Server - is a java jar file which is used for starting selenium server which is as word say server, proxy to selenium grid for browsers you want to automate.

**Selenium web driver Jar Files:**

Below are the web driver jar files for different browsers to initialize.

* Chrome Browser🡪 selenium-chrome-driver-3.12.0.jar
* Internet Explorer Browser🡪 selenium-ie-driver-3.12.0.jar
* Firefox Browser 🡪 selenium-firefox-driver-3.12.0.jar
* Safari Browser 🡪 selenium-safari-driver-3.12.0.jar

**Testing framework Jar files:**

We are using **Junit** and **TestNG** as testing frameworks in our project.

**JUnit** is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit.

**TestNG** is a testing framework for the Java programming language inspired by JUnit and NUnit. The design goal of TestNG is to cover a wider range of test categories: unit, functional, end-to-end, integration, etc., with more powerful and easy-to-use functionalities.

TestNG jar file version **“testng-6.14.3.jar”** and Junit jar file version **“junit-4.11.jar”** is being used in our framework.

**Apache POI selenium jar files:**

Apache POI is the most commonly used API for Selenium data driven tests. POI is a set of library files that gives an API to manipulate Microsoft documents like. xls and .xlsx files through HSSF and XSSF Workbook. Below are the jars added in our framework for Apache POI.

**poi-3.17.jar**

**commons-io-2.6.jar**

Other than the above mentioned jars we have some jars for reporting purpose and provide test results. The reports used in our framework are TestNG and Extent reports.

**JRE System Library:**

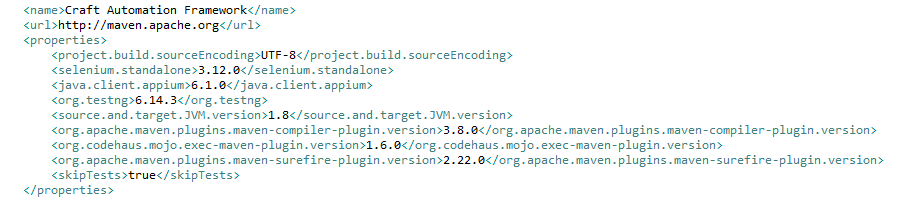
We are using Java 1.8 java development kit in our framework.

JRE System Library is added by Eclipse IDE automatically on creating Java Projects. JRE System Library implements the Java API in Eclipse IDE. So all the predefined functions of Java, can be accessed by the Java Programs written in Eclipse IDE because of this JRE System Library.

**POM.xml**

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is target; the source directory, which is src/main/java; the test source directory, which is src/main/test; and so on.

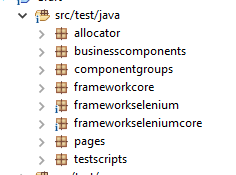
The POM was renamed from project.xml in Maven 1 to pom.xml in Maven 2. Instead of having a maven.xml file that contains the goals that can be executed, the goals or plugins are now configured in the pom.xml. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal.



# **[Main Components of Framework](\\\\Louisilon02s\\userdat01\\YRK2638\\Yashwanth\\New folder\\Craft_Framework.docx)**

**src/test/java**

The test folder of the project consists of packages in which certain java classes are present. Each java class under the package perform different functions.



**Allocator package:**

**Allocator.java**

This java class file is written to manage the batch execution of test scripts within the framework.



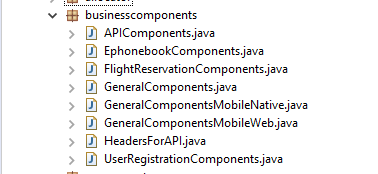
**ParallelRunner.java**

This java Class is written to facilitate parallel execution of test scripts.

**Business Components package:**

In Business Components package we have several java class files in which each class file determine the testing type like API testing, UI and Mobile testing. We design scripts for all testing types according to the Functionality.

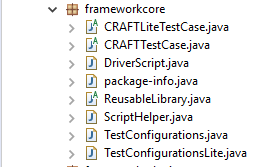
We have “General components” java class file in which we have the common functions written. The common functions include Application login, logout etc., we can reuse these functions according to the application need. Likewise, we have “General components Mobile native” java class file where the common functions for mobile testing are written.



**Framework core package:**

In this package we have several java class files as listed below:

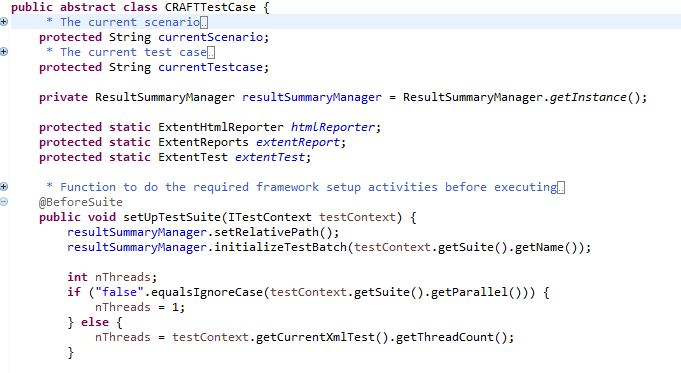
* **CraftTestcase.java**
* **DriveScript.java**
* **ReusableLibrary.java**
* **CommonFunctions.java**
* **ScriptHelper.java**
* **TestConfigurations.java**



**CraftTestcase.java:**

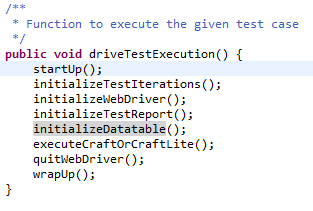
The java class consists of methods like Suite, runner which perform different functions.

A method to set up the test suite, close the same test suite we have written methods in this java class file. Likewise we have methods to set up and close for runner and extent reports.



**DriveScript.java:**

This is the main java class to run a test case. For a test case to run we have a main function **“driveTestExecution”.**When each test case is triggered, it calls this main function under which we have certain sub functions.

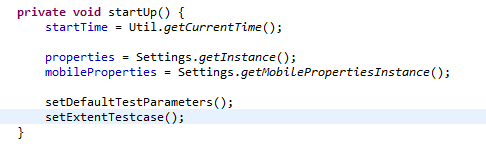


**Sub Functions:**

* + - startUp();
    - initializeTestIterations();
    - initializeWebDriver();
    - initializeTestReport();
    - initializeDatatable();
    - quitWebDriver();
    - wrapUp();

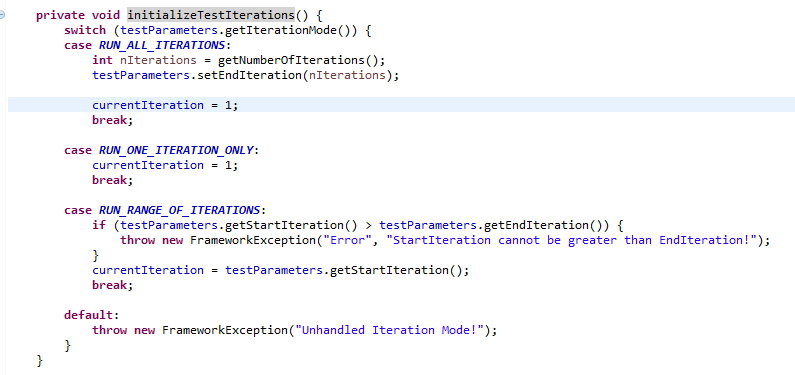
**Startup:**

This startup method has functions to store current time, instance. These two methods refers utility class and settings class respectively to encapsulate functions of the framework. The class file utility and settings are present in the package **“frameworkseleniumcore”.**



**InitializeTestIterations:**

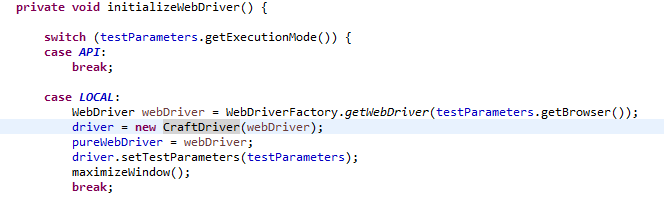
This method is used to run the test cases in iterations. Number of times for the test cases to run is defined in these lines of code. The test run can be once or provided within a particular range or all the test runs at once can be done.



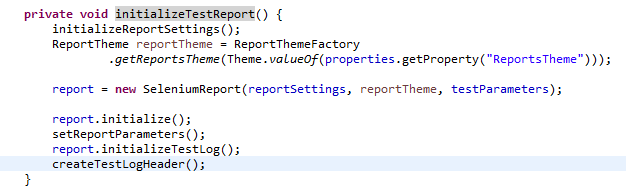
**InitializeWebDriver:**

This method initialize the platform in the tests are to be executed. The different platforms include Local, Grid, Mobile, Perfecto, Saucelabs etc..,

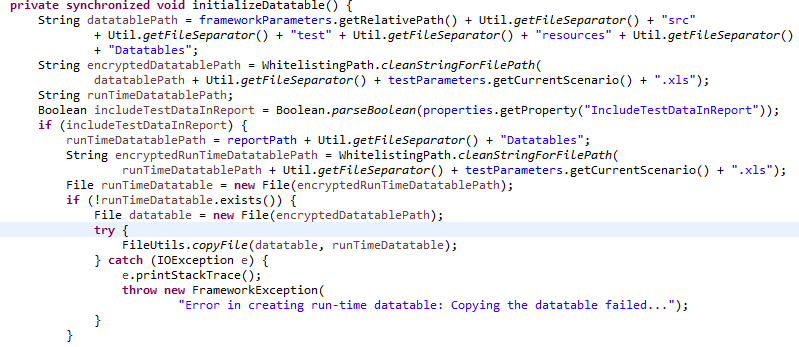
When the platform is set to “Local”, The browser in which the test cases are to executed are provided in “Test configurations” sheet of input test data (Run Manager).Likewise Grid, mobile, perfecto and all others are written in this method.

**InitializeTestReport:**

How to initialize the test report, the theme and format of the report and the test logs are defined in this method.

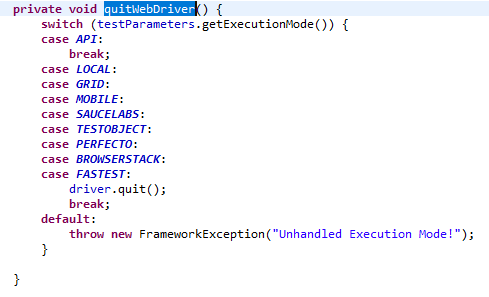
**InitializeDatatable:**

This method consists of the input data sheet that is defined for separate test types. The input required for API is fed in “APIScenario.xls”, the input for mobile testing is fed in “**MobileTestingScenario.xls**”.Also then we can use input sheets for the scenarios used as per our application.



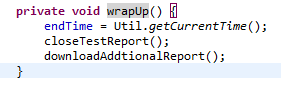
**QuitWebDriver:**

This method is written to close all the open platforms which is opened to execute the test scripts. The platform like grid, perfecto, Saucelabs or any other that is opened automatically closes after the end of execution.



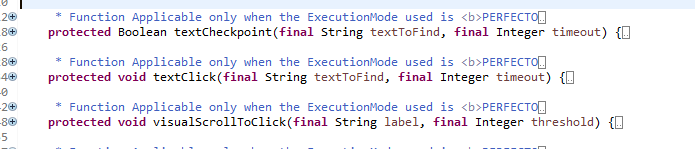
**WrapUp:**

This method is written to close all the reports after the test results are recorded. If any additional reports added like extent reports, the reports will be downloaded.



**Reusable.java:**

The common functions like function to click an element, mouse hover on an element, select a value from dropdown, check the element visible and all other functions used as commands are written in this method.

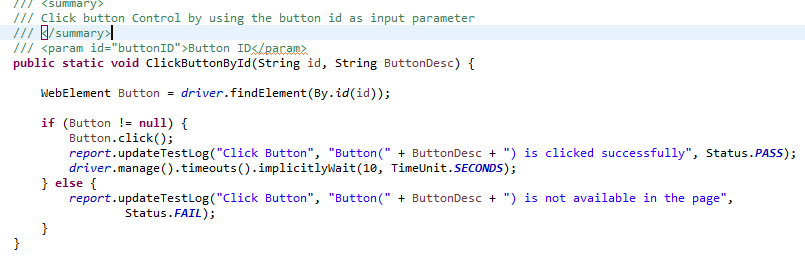


**CommonFunctions.java**

The common functions java file consists of the reusable methods to perform the basic operations like select a value from dropdown, enter value in text box, click an element by locator, scroll the page up and down, validate if an element is present ,page header is displayed etc.,

We can use these methods for the reusability purpose. Call these methods when designing the test scripts.

For example, some methods are explained below.



By calling ClickButtonById, we have to provide the inputs of the parameters passed. The value of the locator id should be sent as first parameter and any description for the element can be given in second parameter. The method should look like the below



Here**,** btnSearchAssociates is the value of id from html and SearchAssoicates is the description of the element.

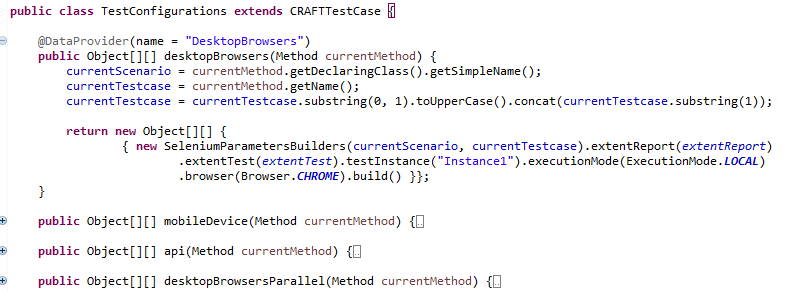
Method Name with return type of this Framework are listed in the below spreadsheet document.

****

**Testconfigurations.java:**

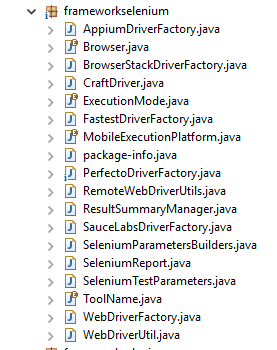
 To write **data-driven tests** which essentially means that same test method can be run multiple times with different data-sets, we use Data Provider annotation. The Data Provider returns a double Object class array with two sets of data i.e. “data one” and “data two”. This is defined in testconfigurations.java such that the data provider used in tests are called from here.

**@DataProvider** is the second way of passing parameters to test methods except passing parameters from testng.xml.



**Framework Selenium package:**

All the configurations required for selenium browser settings, webdriver properties , saucelabs setting, execution mode are all written in separate java classes and are present under “**framework selenium**” package.



**Browser.java**

Enumeration to represent the browser to be used for execution. Each browser to be used is mentioned in this class file.

**SeleniumReport.java**

Whenever each test cases runs, the application screenshot is taken against the report generated. You need to type cast WebDriver instance to TakesScreenshot. The code to take screenshot is written in this java class file.

**WebDriverFactory.java**

The Brower system setting is defined in the java class. The path of each browsers, proxy required for every browser configurations is defined in this java class.

The desired capability is a series of key/value pairs that stores the browser properties like browser name, browser version, the path of the browser driver in the system, etc. to determine the behavior of the browser at run time. The desired capabilities for our framework is set in this java class file.

**SauceLabsDriverFactory.java**

The same browser name, version, platform, proxy required for the tests to be run in saucelabs are defined in the java class. The System setting and desired capabilities for saucelab are set in this java class file.

**PerfectoDriverfactory.java**

The same browser name, version, platform, proxy required for the tests to be run in perfecto are defined in the java class. The System setting and desired capabilities for perfecto are set in this java class file.

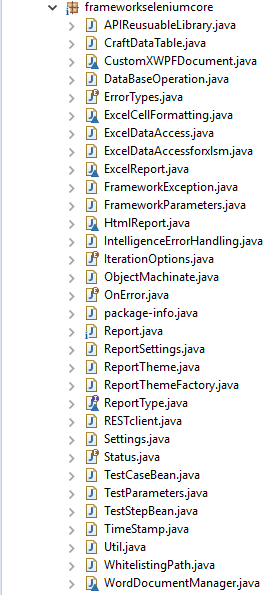
**ResultSummaryManager.java**

To set the path of the report, test batch ID, test summary, report timestamp we have written code in the java class. Also, the format in which the report should be like the header of the report is defined. And the method to override the test executed results when it is run for more than one time is present in this java class file.

Other java class like “Executionmode.java” to define the mode of execution, tool name, package info are present in this “**FrameworkSelenium**” package.

**Framework Selenium Core package:**

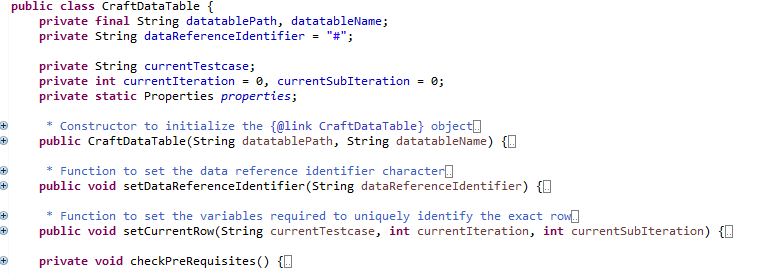
The detailed description of each java class file in **FrameworkSelenium** package is explained in **FrameworkSeleniumCore** package. In **FrameworkSeleniumCore** package we have certain main java class files as briefed below:



**CraftDataTable.java:**

Class to encapsulate the data table related functions of the framework.

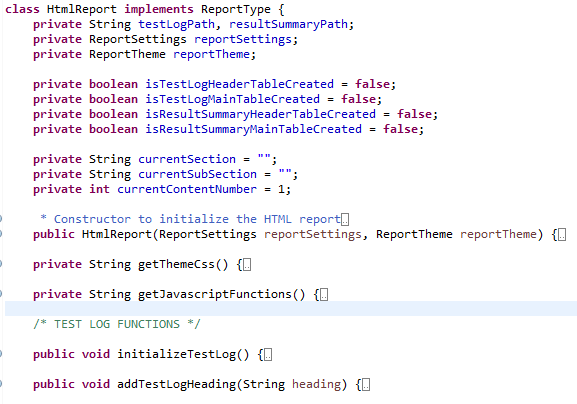
The main function of this class file is to read and write excel data. The excel data is fetched using a method called “**getData**” and the code is written in this class. Also if we want to write something to excel “**putData**” method is written in this class file.



**HtmlReport.java**

Class to encapsulate the HTML report generation functions of the framework.

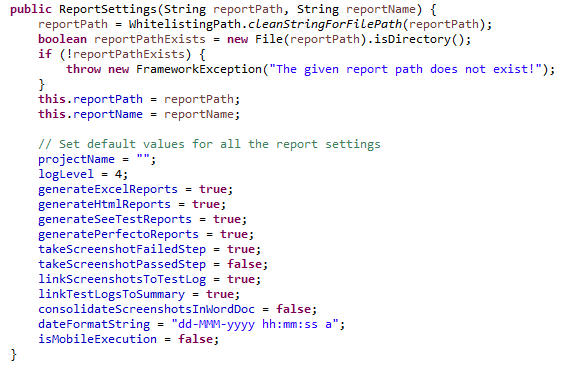
This class file is used to customize the HTML reports. Here the function is written to set the theme, font, heading, sub heading, main heading, color etc., of the report.



**ReportSettings.java**

Class to encapsulate the reporting settings in the framework.

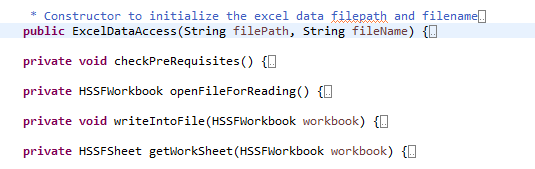
The Project Name, logLevel, date Format String, takeScreenshotFailedStep, takeScreenshotPassedStep, linkScreenshotsToTestLog, linkTestLogsToSummary of the result report are defined in this class. The defined methods work according to the true/false values set. For example, if the value is set as “True” for takeScreenshotFailedStep, the method takes screenshot of the test case and if it is set to “false”, the screenshot will not be taken.



**ExcelDataAccess.java**

Class to encapsulate the excel data access layer of the framework.

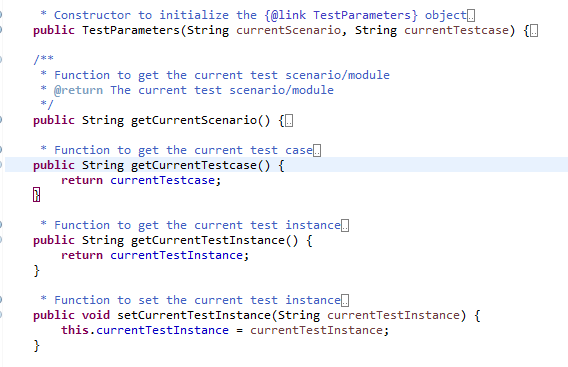
This class is used to create a excel workbook (like HSSF Workbook), to feed input into it, set sheet name etc..,



**TestParameters.java**

Class to encapsulate various input parameters required for each test script.

This method consists of the parameters that is passed in input excel sheet. The column header values like current Scenario, current Test case, current Test Instance, current Test Description, additional Details, iteration Mode, start Iteration, end Iteration are defined in this class file.



**DataBaseOperation.java**

Similarly, all the configuration setting for database to set the port, host, store data, read and pass the data and all methods are written in this class file. All the database operations performed are defined.

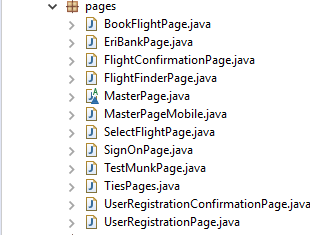


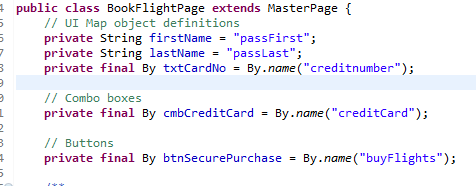
**Pages package:**

Page Object Model, also known as POM, is a design pattern in Selenium which has gained more popularity in the market for test automation development for the maintenance of code, such as reusability, extensibility, and avoiding code duplication etc. Page Object Model is used for creating Control properties or Object Repository for controls on a webpage.

For each webpage which we want to automate there should be a separate class such as if we are performing the automation for the login page, we need to maintain all the login page control properties in the separated class file. If we consider a login page, there are controls available in a login page such as Username, Password, Login, Forget Password etc. Each control will have unique control properties such as Name, ID, Tag Name, XPath, Class Name, css Selector etc.

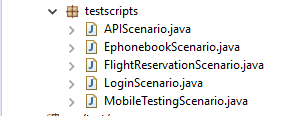
In our Framework we create each java class file for each module like API, Mobile and Web. According to the module we store the objects in the java class file.

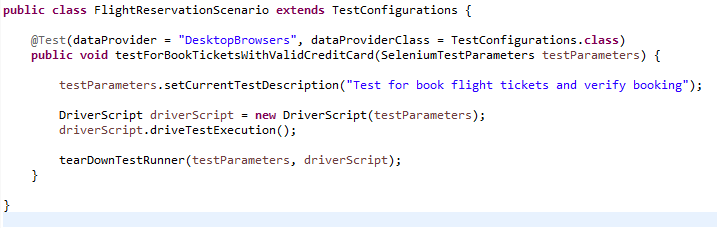




**Test scripts package:**

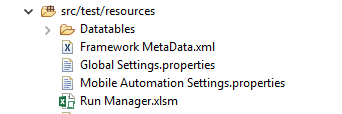
In this package we have java class files for each module like API, Mobile and Web. In each java class file we have the test case written (@test) with respect to the module/scenario.





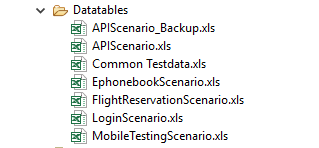
**src/test/resource:**

This package has a “**Datatables**” folder, two **Property** **file** and “**Run Manager**” excel file.



**­Datatables Folder:**

This folder has all the mandatory input data sheets needed for test execution. Here the excel data is maintained according to the module/scenario. The input values required for the test cases designed in “business components” package is stored in the folder.



**Property file:**

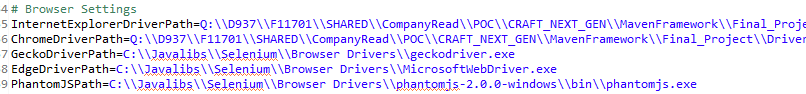
* + **Global settings.properties**
  + **Mobile automation settings.properties**

**Global settings.properties:**

Read data from Properties file using Java Selenium. '.properties' files are mainly used in Java programs to maintain project configuration data, database configuration or project settings etc. Each parameter in properties file are stored as a pair of strings, in key and value format, where each key is on one line.

The project name, environment, driver path and run configuration should be set according to the project. The project name here is sample project, it should be changed according your project and also the environment. The driver server path should point the path where the browser exe file is stored.



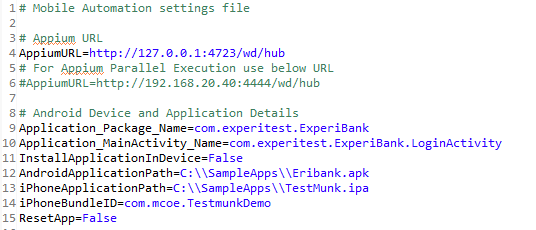
The run configuration is set as “Sanity” here so that the sanity test cases in Run Manager will be executed. So the run configuration can be set to “Sanity”, “Regression” or “API” according to the requirement.



**Mobile automation settings.properties:**

The data configuration, database configuration and all other project setting needed for mobile automation is maintained in this property file. The general configurations are present in “Global settings” file whereas the configuration for Sauce labs, perfecto and mobile are stored as a pair of strings, in key and value format in this property file.

* Mention the HP Mobile Center Host, User & Password
* **Application\_Package\_Name:** Provide the Package Name of Android Application
* **Application\_MainActivity\_Name:** Provide the Main Activity Name of Android Application.
* **iPhoneBundleID:** Provide the iOS application bundle ID.

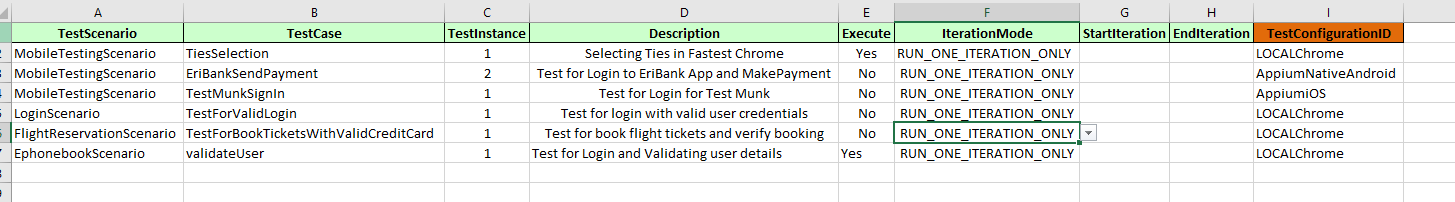


**Run Manager:**

The mode of execution for Desktop, Mobile, Perfecto and Saucelabs are defined in the “**TestConfigurations**” sheet of Run Manager excel spreadsheet. The detailed explanation of each type is explained below as follows:

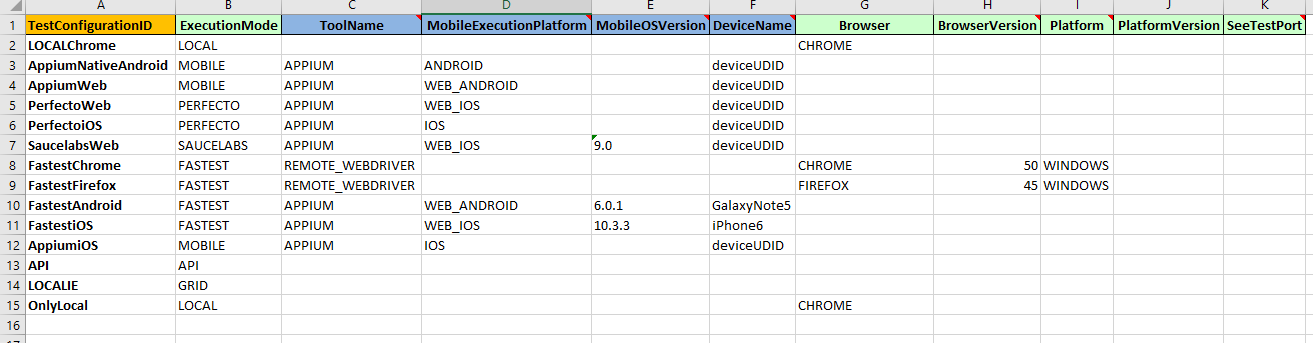
**Working with Desktop Automation:**

* Configure the Run manager with all required details such as Test Scenario, Test Case, Test Instance, Iteration, and TestConfigurationID.



* The TestConfigurationID’s are prepopulated from TestConfigurations Sheet. All Configuration Details are abstracted in separate sheet.

* TestConfigurations Sheet contains all execution details mapped to a TestConfigurationID.
* Any New Test Configuration can be added.
* The TestConfigurationID should be Unique Name. The TestConfigurationID can be referred within your Suite Sheets like Regression, Sanity.

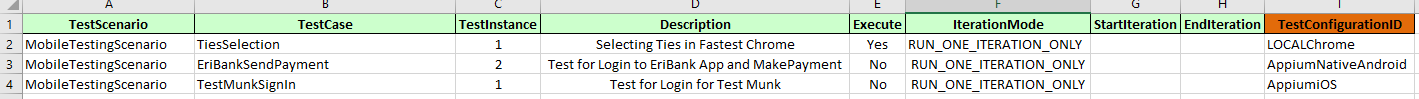


* For **Desktop** automation, set the following as,
  + **Execution Mode** – LOCAL
  + **Browser** – CHROME/ FIREFOX/ SAFARI/ GHOST\_DRIVER/ INTERNET\_EXPLORER/ OPERA
  + **Browser Version** – This option is applicable only when Execution Mode=GRID.

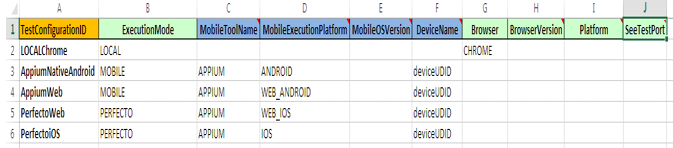
|  |  |  |
| --- | --- | --- |
| **Column Name** | **Possible values** | **Additional Details** |
| TestConfigurationID | Any Valid Unique Name – Should Explain the Respective Configuration |  |
| ExecutionMode | |  | | --- | | LOCAL | | REMOTE | | LOCAL\_EMULATED\_DEVICE | | REMOTE\_EMULATED\_DEVICE | | GRID | | MOBILE | | PERFECTO | | SEETEST | | MOBILELABS | | SAUCELABS | |  |
| MobileToolName | |  | | --- | | DEFAULT | | APPIUM | | REMOTE\_WEBDRIVER | | These Values Depend on the ExecutionMode |
| MobileExecutionPlatform | |  | | --- | | ANDROID | | IOS | | WEB\_ANDROID | | WEB\_IOS | |  |
| MobileOSVersion | Version of the Device | Optional value |
| DeviceName | Device Name/UDID |  |
| Browser | |  | | --- | | CHROME | | FIREFOX | | GHOST\_DRIVER | | HTML\_UNIT | | INTERNET\_EXPLORER | | OPERA | | SAFARI | | For Mobile it will take default CHROME for Android & SAFARI for iOS. |
| BrowserVersion | Version of the browser | Applicable only if ExecutionMode is GRID, also wanted to test for different versions of Browser |
| Platform | Platform of OS | Applicable only if ExecutionMode is GRID |
| SeeTestPort | SeeTestPort like 8889 etc., | Applicable only if ExecutionMode is SEETEST |

**Working with Appium Automation:**

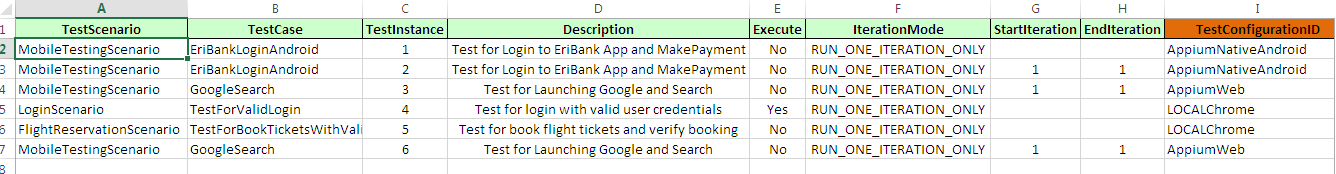
* Configure the Run manager with all required details such as Test Scenario, Test Case, Test Instance, Iteration, and TestConfigurationID.

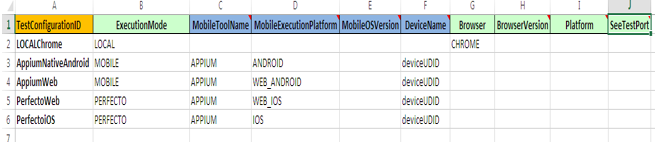


* For **Appium** automation, please change configure as following within TestConfigurations Sheet.
  + **Execution Mode** – MOBILE
  + **Mobile Tool Name** – APPIUM
  + **Mobile OS Version** - Optional
  + **Mobile Execution Platform** - ANDROID/IOS/WEB\_ANDROID/WEB\_IOS
  + **Device Name** – Provide the respective device serial name or UDID



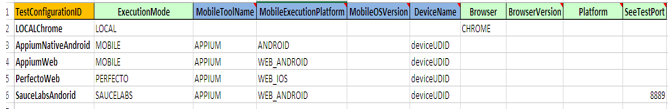
**Working with Perfecto Automation:**

* Configure the Run manager with all required details such as Test Scenario, Test Case, Test Instance, Iteration, and TestConfigurationID.
* For **Appium** automation, do the following
  + **Execution Mode** – PERFECTO
  + **Mobile Tool Name** – APPIUM/REMOTE\_WEBDRIVER
  + **Mobile Execution Platform** - ANDROID/IOS/WEB\_ANDROID/WEB\_IOS
  + **Device Name** – Provide the respective device serial name or UDID



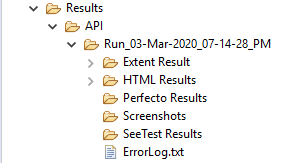
**Working with Sauce Lab Automation:**

* For **Sauce Lab** automation, do the following in Test Configurations Sheet
  + **Execution Mode** - SAUCELABS
  + **Mobile Tool Name** -APPIUM
  + **Mobile Execution Platform** - ANDROID/IOS/WEB\_ANDROID/WEB\_IOS
  + **Device Name** - Provide the respective device serial name or UDID

****

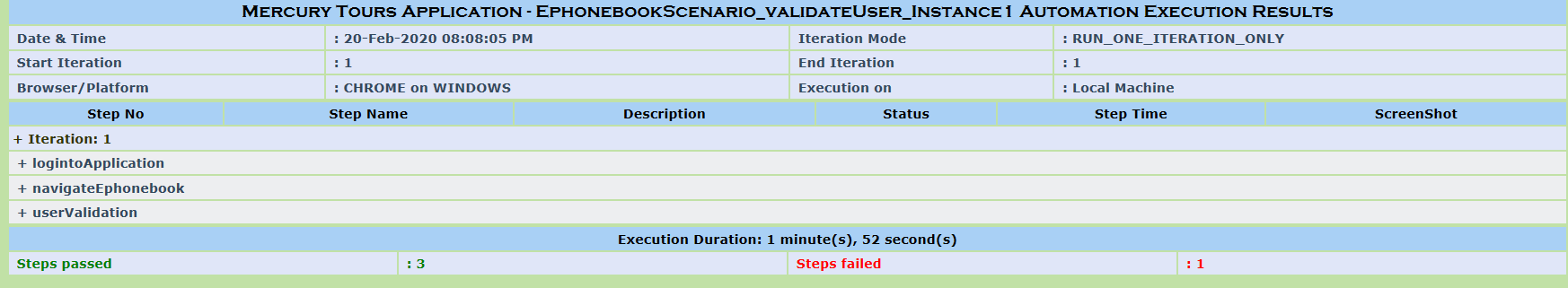
**[Results folder](#_Results_folder)**

The results folder consists of the module-wise test execution results. A batch run ID will be created for each module under which the test results fall. Every batch runner generated has “Extent results” and “Html results”. We can also customize the reports according to our requirement.

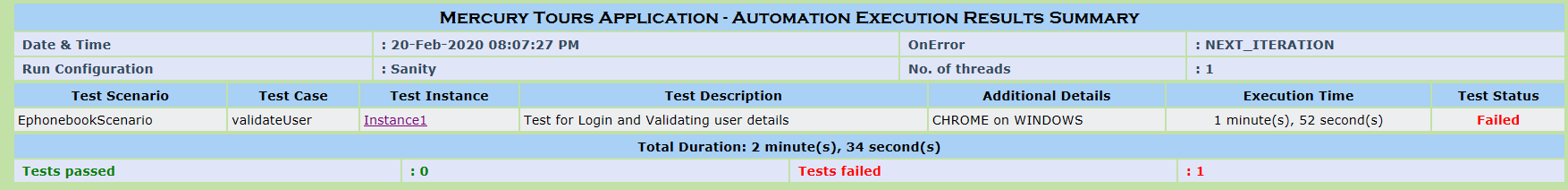


Extent Reports is a reporting library for automation testing for. NET and Java. Extent Reports shows test and step summary along with dashboards, system and environment details for quick analysis of your tests. It is very easily integrated with our frameworks (TestNG and JUnit).

“HTML report” combines detailed information like the errors, test groups, execution time, and step-by-step logs and generates report in html format.



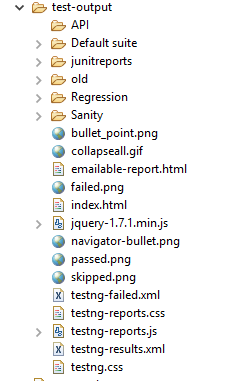
Here we can have the results and view it test case-wise as well as summary of the test execution.



**Test-output folder**

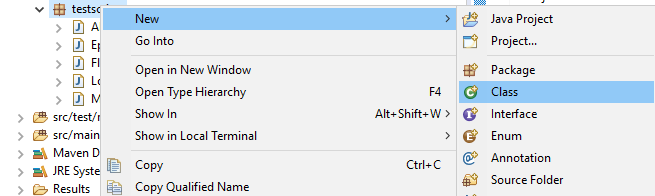
By default the report files (HTML & XML) are written to a folder named test-output under your workspace.

When you execute testng.xml file, and refresh the project, you will get test-output folder in that folder. Right click on the emailable-report.html and select the option so that you can open with the web browser and view the results.

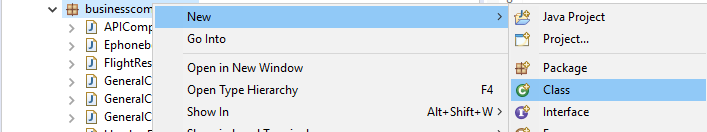


**How to Create and Run a Sample Test Case for Web Application**

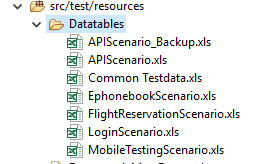
Step 1:We have to create a java class under “testscripts” package where we will write the test cases.



Step 2: We have to create a java class under “businesscomponents” package. It has all the functions/method designed according to the test cases defined in testscripts package.

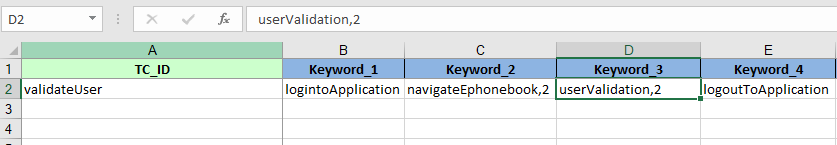


Step 3: The input data required to run the test cases is provided in the excel spreadsheet. And it is present under “Datatables” folder of “src/test/resources” folder. We can create a new excel spreadsheet under this folder and feed the inputs according to the scenario/module.



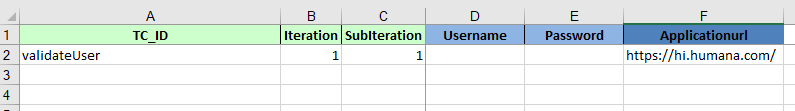
**Business\_flow sheet:**

In the below example, “validateUser” is the Test case present in “EphonebookScenario“ of testscript package. When the execution is triggered, the test “validateUser” is called and the methods (in column B,C,D,E) is performed according to the code designed in “EphonebookComponents” java class of businesscomponents package.

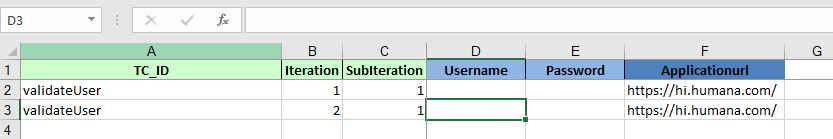


**General\_Data sheet:**

The login credentials and application URL required for a test case to run is mentioned in this sheet.

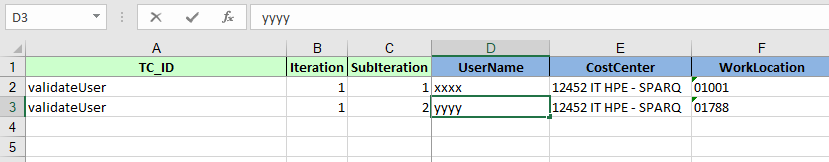


When the “Iteration” is set to 1, the test case will execute once with the login details and URL provided. If we want to run the same test case with some other login credential we should set the “Iteration” to 2 as mentioned below and provide the user details.

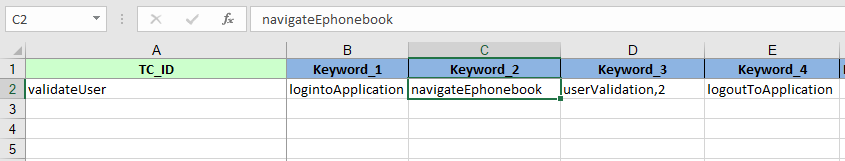


**Passenger\_Data:**

If a particular function needs one more run (or) if it requires different data’s to be passed, we use “Passenger\_Data” sheet.



This sheet is linked to the “Business\_flow sheet” (below). For example validateUser is the test case and userValidation is the method written in it. If we want to pass another input data to this userValidation function,mention “userValidation,2” in column D.(say for this scenario) and the input values for this method(userValidation) should be given in “Passenger\_Data” sheet.



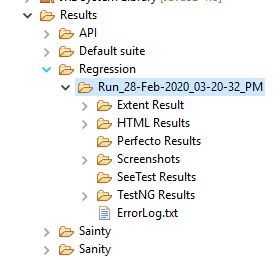
**Mode of Execution:**

We can execute the test cases in two different ways.

* One way of execution is using allocator.java class.
* Right click on allocator.java class file🡪Run as🡪Java Application
* We can create separate profile for running Allocator.java.
* We would need mojo plugin as mentioned below to trigger the java file.
* We can run below as **mvn clean test –P runAllocator** from command prompt.

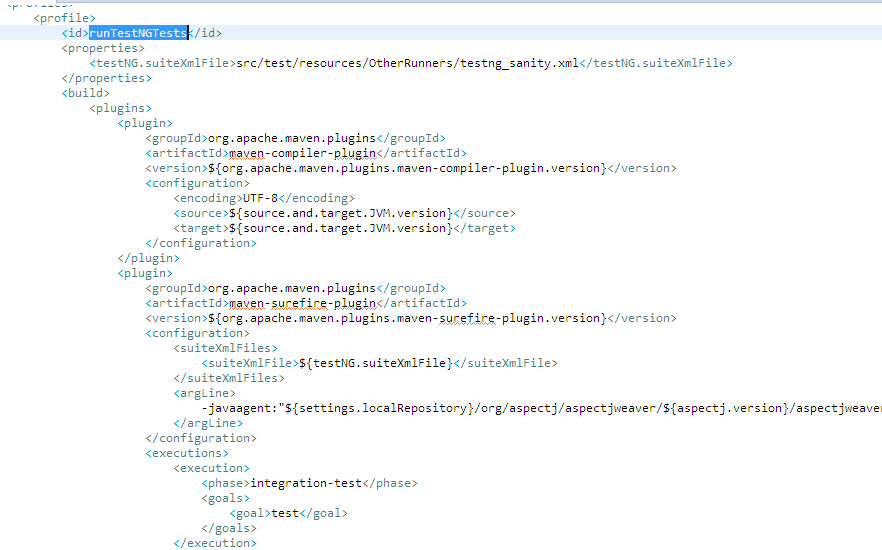


On executing the test cases using “Allocator”, the results lie under “Results” folder. If it is regression (or) sanity (or) API, the test results lies under the respective folders accordingly. A batch run ID will be created under the subfolder and the results will be stored in it. (For example, this is the batch run ID created Run\_28-Feb-2020\_03-20-32\_PM as mentioned below).

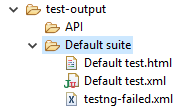


And other way of executing the test cases is as follows:

* Right click on the particular test case which you want to run.
* Run as🡪TetsNG Test.
* We can create separate profile for running TestNG Suite files.
* We would need surefire plugin as mentioned below to trigger the TestNG files.
* We can run below as **mvn clean test –P runAllocator** command prompt.

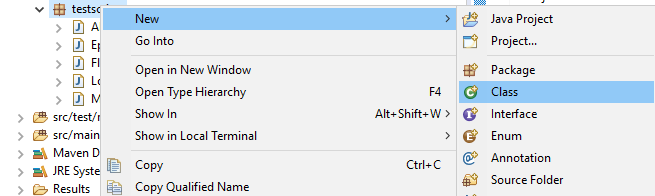


On executing the test cases using “TestNG Test”, the results lie under “test-output” folder. Here the results are stored in the “Default suite” folder.

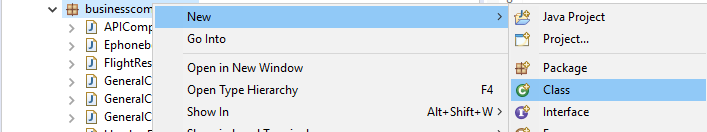


**How to Create and Run a Sample Test Case for API**

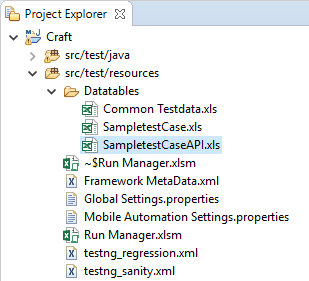
Step 1:We have to create a java class under “testscripts” package where we will write the test cases.



Step 2: We have to create a java class under “businesscomponents” package. It has all the functions/method designed according to the test cases defined in testscripts package.

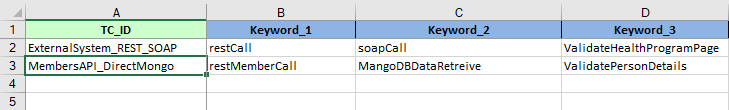


Step 3: The input data required to run the test cases is provided in the excel spreadsheet. And it is present under “Datatables” folder of “src/test/resources” folder. We can create a new excel spreadsheet under this folder and feed the inputs according to the scenario/module.



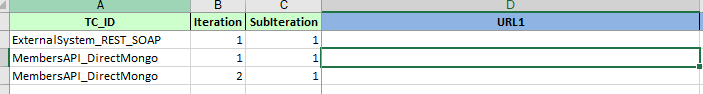
**Business\_flow sheet:**

In the below example, “MembersAPI\_DirectMongo” is the Test case present in testscript package. When the execution is triggered, the test “MembersAPI\_DirectMongo” is called and the methods (in column B,C,D,E) is performed according to the code designed in “SampletestCaseAPIComponents” java class of businesscomponents package.

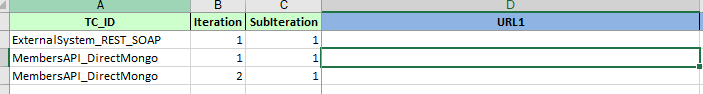


**General\_Data sheet:**

The login credentials and application URL required for a test case to run is mentioned in this sheet.

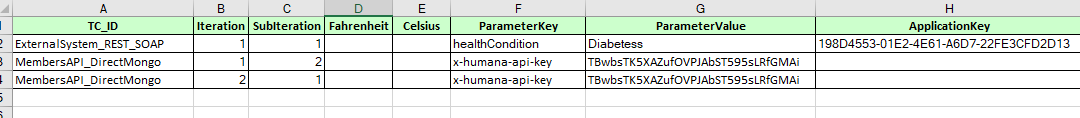


When the “Iteration” is set to 1, the test case will execute once with the login details and URL provided. If we want to run the same test case with some other login credential we should set the “Iteration” to 2 as mentioned below and provide the user details.

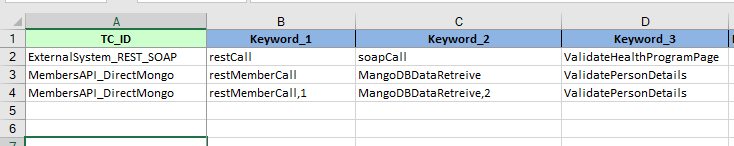


**INT\_DataSheet:**

If a particular function needs one more run (or) if it requires different data’s to be passed, we use “INT\_DataSheet” sheet.



This sheet is linked to the “Business\_flow sheet” (below). For example “MembersAPI\_DirectMongo” is the test case and restMemberCall is the method written in it. If we want to pass another input data to this restMemberCall function, mention “restMemberCall,2” in column B. (say for this scenario) and the input values for this method (restMemberCall) should be given in “INT\_DataSheet” sheet.



**Main Components of API:**

There are two class files considered as the main components of API.

1. APIReusuableLibrary.java
2. HeadersForAPI.java

**APIReusuableLibrary.java**

All the common and generic functions are written in this class file. API test cases are designed with the help of these common functions. This java file is present in framework selenium core package.

The detailed description of the common functions are explained in the excel file attached.

****

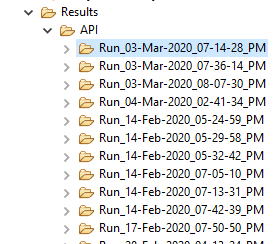
**HeadersForAPI.java**

This java class file is present in the business components package. Here the headers of the API are defined and each header definition has the content type, header key, header value etc., we can select the header according to our business requirement.

**[Mode of Execution of API:](#_Mode_of_Execution)**

* To execute API test cases, the Run configuration should be set to API in Global setting.properities file.
* Using allocator.java class file we are executing our scripts.
* Right click on allocator.java class file🡪Run as🡪Java Application

On executing the test cases using “Allocator”, the results lie under “Results” folder. If it is regression (or) sanity (or) API, the test results lies under the respective folders accordingly. A batch run ID will be created under the subfolder and the results will be stored in it. (For example, this is the batch run ID created Run\_03-Mar-2020\_07-14-28\_PM as mentioned below).

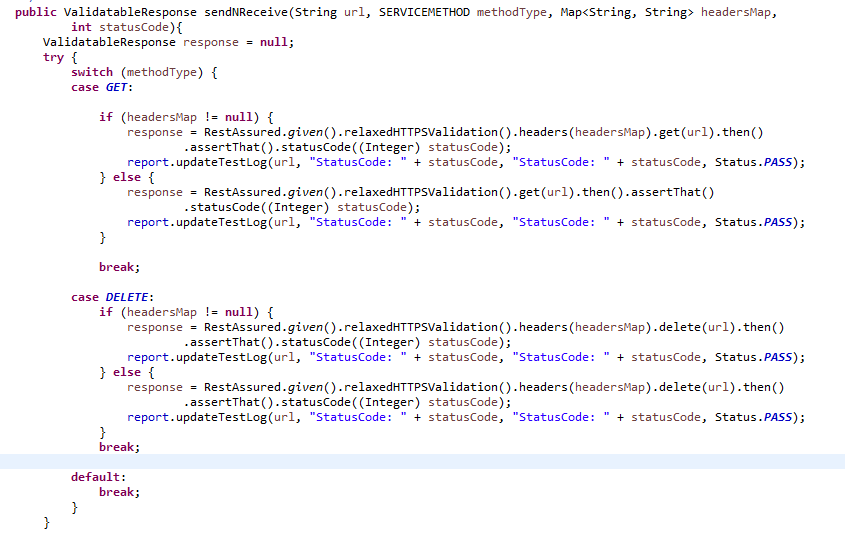


## **REST and SOAP UI Implementation:**

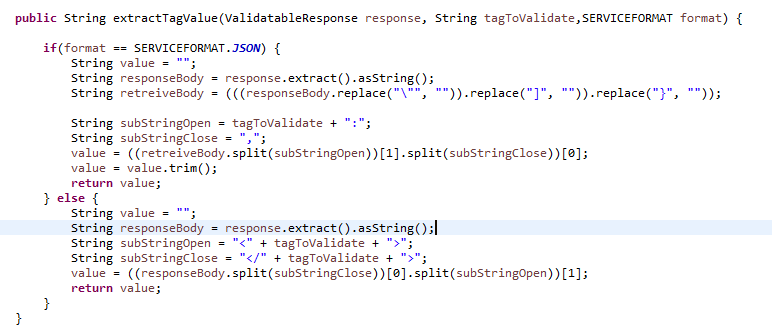
**RestCall** is the main function for rest services defined in our framework and it is present in SampleAPIComponents.java class file.



In APIReusuablejLibrary.java class file, we have “sendNReceive**”** function validates the response received from Endpoint URL sent. The Endpoint is parameterized in General data sheet of Sample test caseAPI.xlsx. After the URL hits, we have some defined the method like GET, POST, PUT, PATH, DELETE, UPDATE, HEAD and OPTIONS. After the method we get status code. The status code validates the valid and invalid response of the response received. For example, using Get method, we get a status code and if the code is 200, that is considered a valid response and if it is other than 200, that is an invalid response.

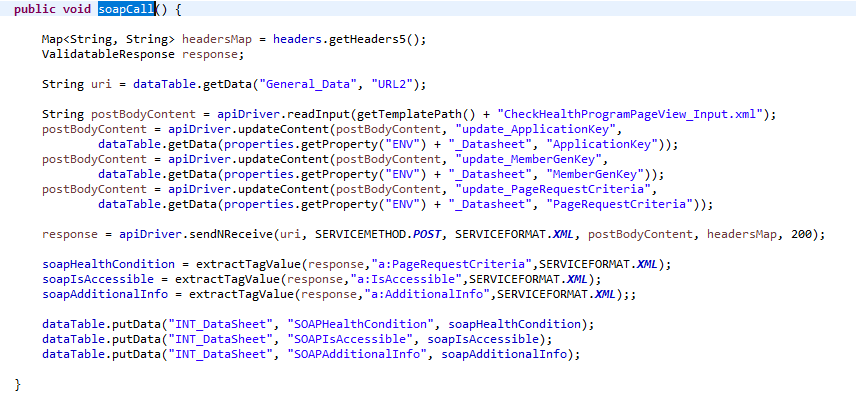


In case of the valid response, the response body is validated by the method “**extractTagValue**”.One of the Headers called Content-Type which tells how to interpret the data present in the Body of the Response. If the Body contains data in the form of JSON, then the value of Content-Type header will be application/json. Similarly, if the data in the body is XML the Content-Type header will be application/xml.

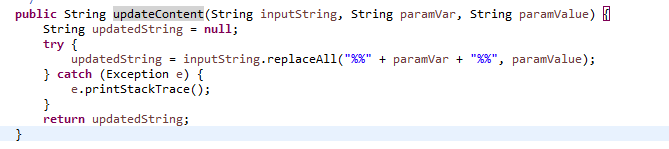


SOAP is a remote procedure call protocol that works over HTTP. SOAP requests are HTTP POST requests made to the web service endpoint URL. The client and server exchange data in the XML format in the body of HTTP requests and responses.

**soapCall** the main function for rest services defined in our framework and it is present in SampleAPIComponents.java class file.



The method “**getTemplatePath**” picks the xml file provided in the miscellaneous maven path and “**readInput**” method reads the xml file and stores the post body content. The value of the body content is updated according to the “**application** **key**” as per business requirements by the “**updatecontent”** method. The application key is parameterized in the “**INT\_DataSheet**” of Sample test caseAPI.xlsx.



This method “**sendNreceive”** hits the URL, we have defined method POST, and using the method we post the URL. The “**getPostBodyContent”** function is invoked to get the body content provided by the content type like XML, JSON, File etc., One of the Headers called Content-Type which tells how to interpret the data present in the Body of the Response. If the Body contains data in the form of JSON, then the value of Content-Type header will be application/json. Similarly, if the data in the body is XML the Content-Type header will be application/xml. using the status code validates the valid and invalid response of the response received. We get a status code and if the code is 200, that is considered a valid response and if it is other than 200, that is an invalid response.

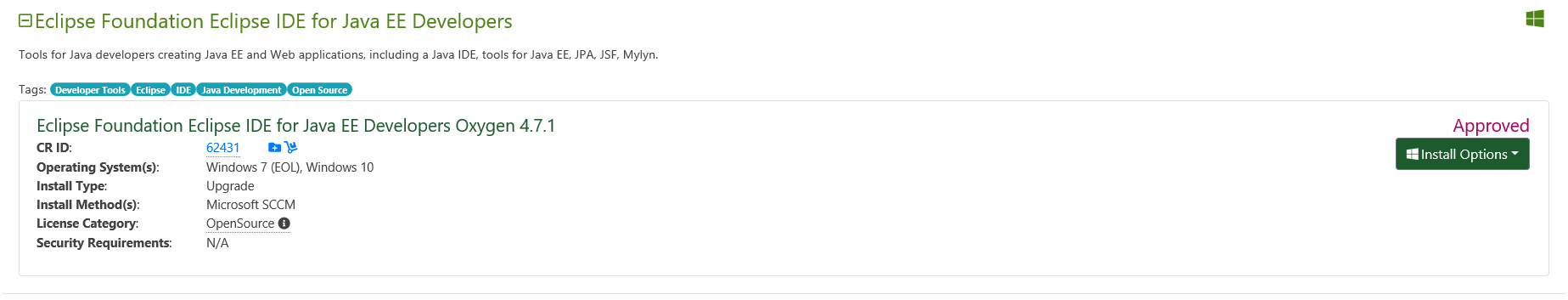


In case of the valid response, the response body is validated by the method “**extractTagValue**”.An XML is extracted, and several tag names are retrieved. The retrieved value is validated with the updated values provided as inputs.

# **Summary**

**Get Eclipse Oxygen installed in your Local Machine through the following steps:**

1. Enter URL “go/software/” in the browser.
2. Search for “Oxygen” in the search box provided and select Eclipse Oxygen to install.



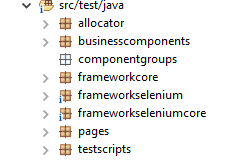
1. After that Eclipse Oxygen will be installed in your local machine.

**How to implement the existing framework**

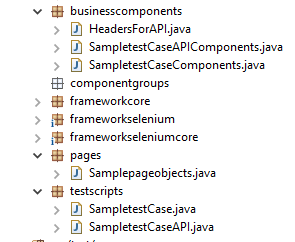
1. Open the Eclipse Oxygen installed in your machine.
2. Click File 🡪 Import 🡪 Maven 🡪 Existing Maven Project 🡪 Click next 🡪 Browse the Directory where the framework is located 🡪 Click Finish.
3. Now the “Craft Framework” is imported to the eclipse.

**Create a sample project in Craft Framework:**

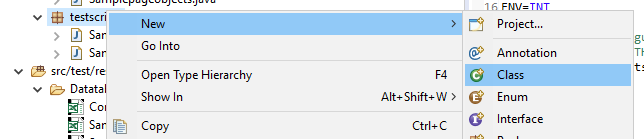
1. In here we have three base packages “**businesscomponents**”, ”**pages**”, ”**testscripts**”.



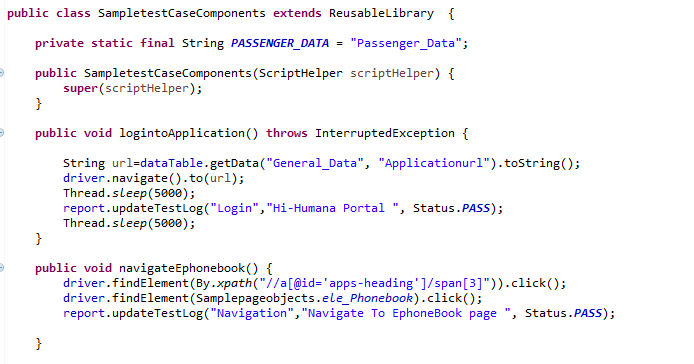
1. A java class should be created under each Packages mentioned.



Create a java class by right clicking on the respective package 🡪 new 🡪 Class



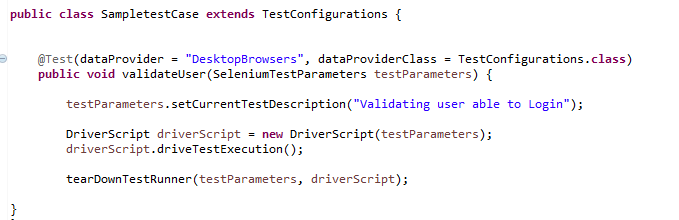
1. For example, **businesscomponents** package consists of a java class in which the scripts are designed according to the scenario. For Example, here we create SampletestCaseComponents.java class file. In the class file we design scripts according to our test case scenario.



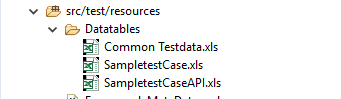
1. Likewise **pageobjects** package consists of the java class in which web elements are stored according to the web page. For Example, here we create Samplepageobjects.java class file. In the class file we stored our locators/elements according to our test case scenario.



1. **testscripts** consists of java files in which test cases are written. For Example, here we create SampletestCase.java class file. In the class file we have test cases according to our application.



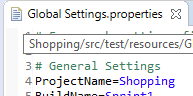
1. The data inputs required for the test case should be given in the excel spreadsheet under **datatables** folder of **“src/test/resources”**. Here we have sample test data as shown below. The naming convention of the test data should be of the java class name mentioned in test scripts package. For example, SampletestCase.java is the java class file in test scripts package, so the test data being used should be “SampletestCase.xls”.



# **Steps to Execute the tests After Completing Design**

The first step to do when executing a test case is to open “Global Settings.properities” file and make the below changes.

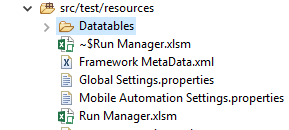
1. Modify the Project name.



1. Run Configuration should be set to “Sanity” or “Regression” or “Crossplatform” or “API” according to the mode of test that we are planning to execute. The options are present as sheets in the “Run Manager.xlsm” spreadsheet.

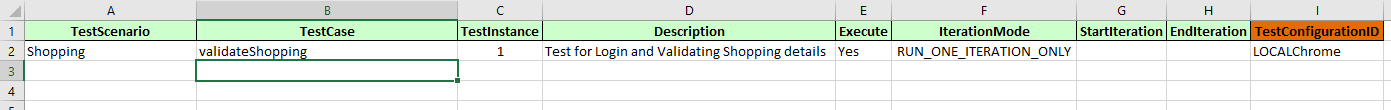


Provide the inputs of the selected test cases in Run Manager.xlsm present in “src/test/resources” folder

.

Please find the below example where the input required to run a test cases is given

1. The Test Scenario name to be executed in the TestScenario column.
2. The Test Case name of the scenario selected should be given in the TestCase column.
3. The TestInstance should be selected as “1”.
4. The description of the Test Case is given in the Description column.
5. If the “Execute” column is set to “Yes” the selected test runs, if it is set to “No” the test will not run.
6. IterationMode is set “RUN\_ONE\_ITERATION\_ONLY”,so that the test cases executes for one test data provided. It can be set to “RUN\_ALL\_ITERATIONS”, so that the test case executes for n number of test data provided. We can customize according to the requirement.
7. TestConfigurationID column selects the browser in which the tests are to be executed. It can be set to Chrome, Internet Explorer etc.,

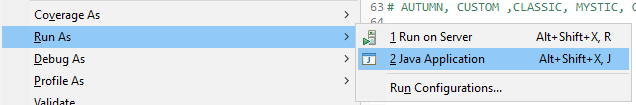


# **Mode of Execution**

## **Execution using Allocator**

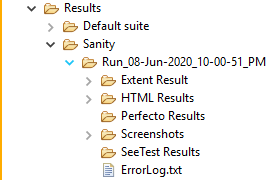
Select allocator package in src/test/java 🡪 Right click on Allocator.java present in allocator package 🡪 Click Run As 🡪 Java Application.





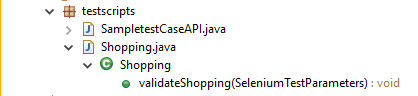
All the test cases that is set to “Yes” in Execute column of Run Manager.xlsm spreadsheet will be executed.

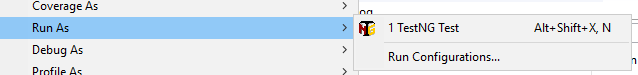
The results of the Allocator execution lies in the folder “Results”. A folder will be created in the name that is set in Run Configuration of Global Settings.properities file. In this example, the Run configuration is set as “Sanity”, so we can find the folder “Sanity” under “results” folder. A batch run ID will be created during every run with the date, timestamp where html results, screenshots etc., of the execution are created.



## **Execution using testscripts**

Select testscripts package in src/test/java 🡪 we can find many Test Scenarios 🡪 Expand the test Scenario (For Example:shopping.java as shown below) 🡪 Select the test case to be executed (validateshopping as shown below) 🡪 Right click on the selected test case 🡪 Click Run As 🡪 TestNG Test.





The results of the TestNG test are created in the “test-output” folder. A default suite folder is present under test-output where we can find html, xml and failed results of TestNG execution.

