|  |
| --- |
| Service Delivery - Test Automation **Selenium Framework (SWITCH Ready)** |
| Version <1.1>  Framework Created by: Sivanaga Raju Grandi & Priyanka V |ITT/QIE |

History of Changes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Author/Reviewer | Date | Version | Relevant Chapters | Changes |
| Sivanaga Raju Grandi - Author | 23.04.2020 | 1.0 | All |  |
| Priyanka V - Reviewer | 23.04.2020 | 1.1 | All | Technical contents review |
|  |  |  |  |  |

Contents Index:

1. [1Test Automation Process 4](#_Toc43983716)
2. [2 Framework features & Used Tools 5](#_Toc43983717)
3. [3 Installation of Tools 6](#_Toc43983718)
4. [4 Framework Design/ architecture 7](#_Toc43983719)
5. [5 Project Structure 9](#_Toc43983720)

[**5.1 Package Overview 9**](#_Toc43983721)

[**5.2 Log4j.properties 9**](#_Toc43983722)

[**5.3 Custom Report Folder 9**](#_Toc43983723)

[**5.4 Doc Folder 9**](#_Toc43983724)

[**5.5 Lib Folder 10**](#_Toc43983725)

[**5.6 Reports Folder 10**](#_Toc43983726)

[**5.7 POM.xml 11**](#_Toc43983727)

[**5.8 TestData.properties 11**](#_Toc43983728)

[**5.9 Jenkinfile 11**](#_Toc43983729)

1. [Package Overview 11](#_Toc43983730)

[**6.1 Custom Report 12**](#_Toc43983731)

[**6.2 Data Loader 12**](#_Toc43983732)

[**6.3 Page Objects 13**](#_Toc43983733)

[**6.4 Module Specific Utilities 14**](#_Toc43983734)

[**6.5 Test Scripts 14**](#_Toc43983735)

[**6.6 Utilities 15**](#_Toc43983736)

[**6.6.1 Base Initialize 15**](#_Toc43983737)

[**6.6.2 Business Utilities 15**](#_Toc43983738)

[**6.6.3 Custom listener’s 16**](#_Toc43983739)

[**6.6.4 Excel Actions 16**](#_Toc43983740)

[**6.6.5 Page Actions 16**](#_Toc43983741)

[**6.6.6 Password Encryptio 16**](#_Toc43983742)

[**6.6.7 Retry Failed Test 17**](#_Toc43983743)

[**6.6.8 Test Annotation Transformer Listener 18**](#_Toc43983744)

[**6.6.9 Image comparison 18**](#_Toc43983745)

[**6.7 Run config 19**](#_Toc43983746)

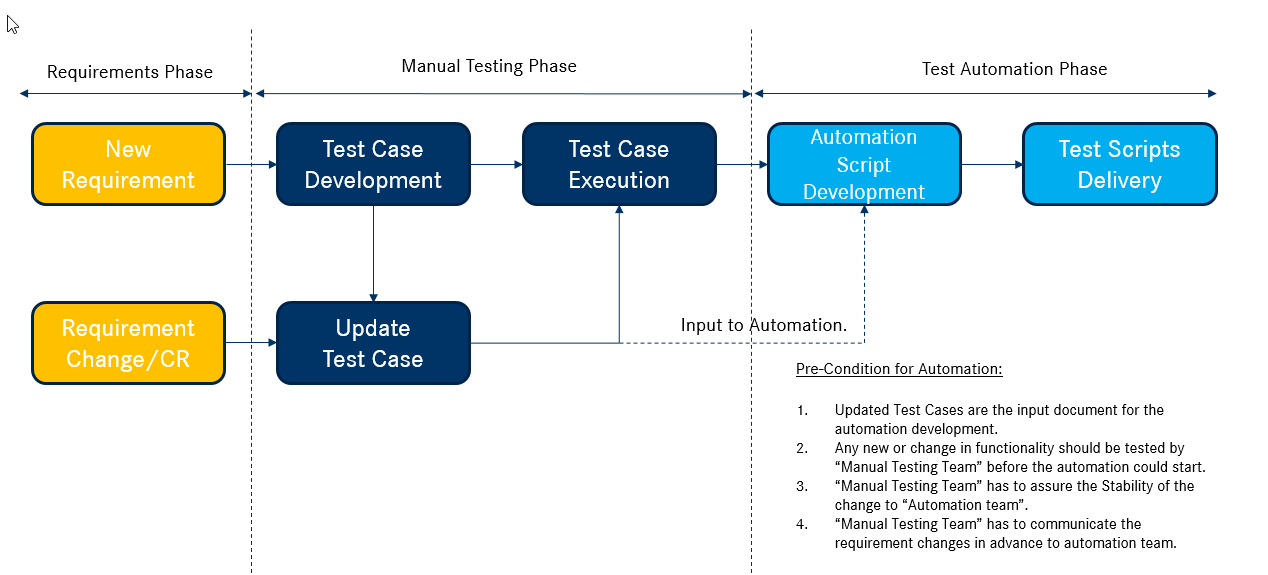
[**6.7.1 Dynamic TestNG 19**](#_Toc43983747)

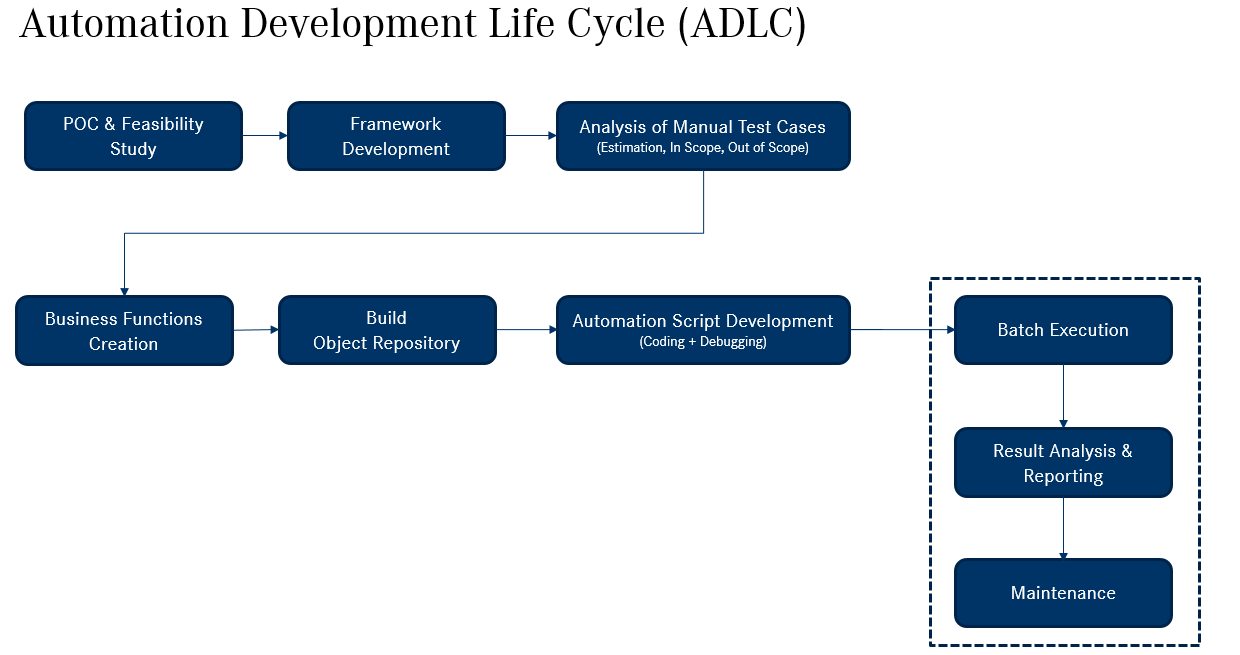
1. [Execution Flow 21](#_Toc43983748)
2. [Jenkin Job Configuration: 22](#_Toc43983749)

# **Test Automation Process**

* Test Automation process provides complete solution from requirements to reporting.
* The purpose is to define the scope of automation and pre-condition for the automation.
* Under the Manual testing phase should get the clear understanding of requirements and the queries should get resolved prior to implementation.
* Testcase development should covered the all requiremetns.
* In Automation phase should get the updated test cases.
* Team should aware of changes in requirements/new features.

Below pictures describe the Test automation process in-detailed.





# **Framework features & Used Tools**

* Implemented Framework has many features.
* To implement the framework different tools are used.
* Below table gives you the summary of features and used tools.

|  |  |  |
| --- | --- | --- |
| S. No | Feature | Implemented In |
| 1 | Source code management | Git |
| 2 | CI/CD | Jenkins |
| 3 | Multiple Browser support | Chrome, Edge, Firefox, IE |
| 4 | Generate multiple reports | Extent report and Custom report |
| 5 | Log Captured | Log4j |
| 6 | Screenshot | Selenium |
| 7 | Group Testing | TestNG |
| 8 | Object Management | Page object Model |
| 9 | Control Execution flow | TestNG Listeners |
| 10 | Re-run failed test | TestNG Listeners |
| 11 | Screenshot controlling | TestNG Listeners |
| 12 | Password Encryption | Java Implementation |
| 13 | Image Comparison | Java Implementation |
| 14 | Email Report | Java Implementation |
| 15 | Multi Linguistics support (Localization) | Java Implementation |
| 16 | Groups execution controlling | Java Implementation |
| 17 | Maintain History of Reports  In public shared drive | Java Implementation |
| 18 | Generate java documentation in Eclipse IDE | Javadoc Generation wizard |
| 19 | Building the project | Maven |

# **Installation of Tools**

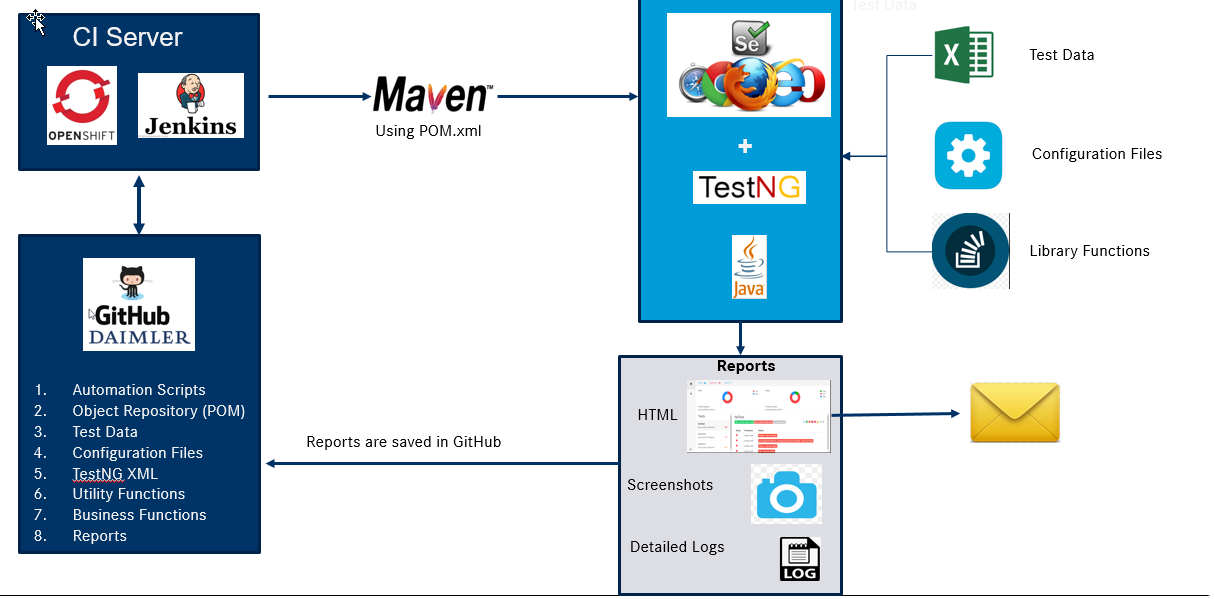
* Installation of tools describes the systematic installation process for the list of software’s.
* The attached document gives systematic installation of below items.
* Java Installation
* Eclipse IDE installation
* Selenium installation
* Maven installation
* Git installation
* TestNG Installation



# **Framework Design/ architecture**

* Framework built on Java+ selenium with help of TestNG assertion framework.
* Test data maintained in Excel, for reading the excel data used Apache POI libraries.
* Implemented the common utility functions and business utility functions.
* Object repository maintenance using Page Object Model, which support multi linguistics objects.
* Implemented the listeners for listens to certain events and keep track of test execution while performing some action at every stage of test execution.
* Framework generate multiple reports i.e extent report and customized report.
* Building of project using maven.
* Source code maintained using Daimler Git.
* For the CI/CD activities used the Openshift Jenkins.

Framework architecture picture has given below for clear understanding.



# **Project Structure**

* Project Folder structure given below.
  1. src/main/java  
     [Package Overview](#_Package_Overview)
  2. src/main/resources  
     log4j.properties
  3. Custom Report
  4. Doc
  5. Lib
  6. Reports
  7. Jenkinsfile
  8. POM.xml
  9. TestData.properties
  10. Jenkinfile

## [**Package Overview**](#_Package_Overview)

* [Click here](#_Package_Overview)for detailed overview of packages**.**

## **Log4j.properties**

* “Log4j” used to generate the customized logs during test script development.
* Log4j.properties is a configuration file.
* Different levels are available while logging, like DEBUG, INFO, WARN, ERROR, FATAL and much more.
* Generated logs will be stored in “ExecutionLogs.log” file under the project

Refer the sample logs in below attached file.

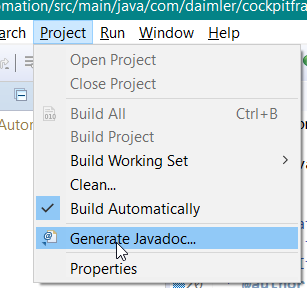


## **Custom Report Folder**

* Custom report contains template and customized reports.
* Generate Customized reports after the execution.

Ex: [Click here](#_Custom_Report) for sample customized report.

## **Doc Folder**

* Documentation folder has multiple HTML files for each package.
* Doc folder has major file for entire project documentation i.e**. Index.html**.
* **Index.html** file gives information of all packages and class’s.
* Doc folder get created when generate the documentation using “Generate java doc” option.
* Step by step process of generate the java documentation.
* Click on project in Eclipse Menu. Refer the sample screenshot for the project menu.  
  
* Select “Generate Javadoc.” Option.
* Select check box of project.
* Select destination path (by default it takes project path)
* Click on Next
* Again Click on Next
* Select 1.8 option in “JRE source compatibility “
* Click on Finish
* Create the “Doc” folder In the Destination.
* Entire project documentation is available in the “index.html”

## **Lib Folder**

* Library folder consisting of several items:
* Browser drivers such as Chrome, Firefox, IE and Edge.
* Excel Configuration file for test data and test script names
* Test data maintained in separate sheet for English and German.
* Separated Test scripts based on Suite (Smoke and Regression) and run configuration option to skip
* To skip tests we have to specify in “Run mode” column value as “No”.

Sample excel attached below for reference.

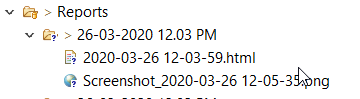


## **Reports Folder**

* Reports folder has sub-folders with Date and time.
* Date and time folder get created before start the test execution dynamically.
* Screenshots and extent report get stored in the date and time folder.
* Extent report name get created with date & time.
* When suite run in “remotely”, generated extent reports copied to specified public shared folder.

Refer the below sample extent report and tree structure of “Reports” folder.





## **POM.xml**

* POM.xml has information of project and configuration for the build the project.
* POM.xml contains dependencies, build directory, source directory, test source directory, plugin, and goals.
* Benefit of adding dependencies the Maven automatically fetches the jars at the given version from Maven Central.
* POM.xml has different libraries sample given below.
  + Selenium, TestNG, Log4j.

## **TestData.properties**

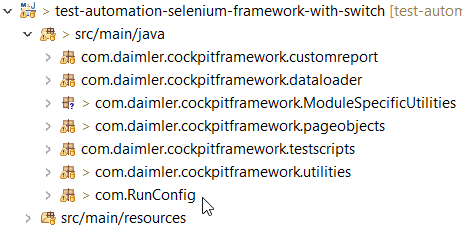
* TestData.properties has list of configuration details.
* Environment URL of TI, INT and PROD.
* User credentials those are User Name and encrypted Password.
* Count for re-run failed scripts i.e max count for re-run.
* Shared driver path for store the reports after the execution.

## **Jenkinfile**

* Jenkinfile is having pipeline script for building the project.
* Having the node configuration & build command.

# **Package Overview**

* Package overview describes the specifications of each package under the project.
* Each package has one or multiple classes.
* Refer the below package structure of Automation Project.



## **Custom Report**

* Custom report package consisting of “customized reports” and “template”.
* The customized report get generated by using the “customreportTemplate.html“.
* Customized report will not provide screenshots and failure reason.
* For detailed report implemented the [Extent Report](#_Reports_Folder) .
* Customized report contains below information.
  + Execution Summary
  + OS version
  + Host Name
  + Browser
  + Environment
  + Duration
  + Total # of executed tests and status of those tests Passed, Failed and Skipped.

Ex: Sample report attached below.



## **Data Loader**

* Data loader class implemented to get the “test data” from the excel file.
* The “test data” will be stored in “Data provider” (it is feature of testNG)
* To use the test data in “test script”, we have to specify “data provider name” and data-loader class path,

Sample test script code given below.



## **Page Objects**

* Maintained the Page objects using the Page Object Model.
* Representation of web pages as classes for each page.
* Various elements on the page defined as variables on the class.
* To support the Page Object Model we use Page Factory, it is an extension of selenium.
* By using page factory initialized the Web-elements with available locators, it done through @FindBy annotation.
* Page object classes implemented the “Multi language objects and parameterized objects”
* Defined the naming conventions for objects for easy understanding.

|  |  |  |
| --- | --- | --- |
| **S.no** | **WebElement Type** | **Representation** |
| 1 | Dropdown | Dpdw |
| 2 | Link | Li |
| 3 | Text Box | TBox |
| 4 | Button | Btn |
| 5 | image | Img |
| 6 | Table | Tbl |
| 7 | Checkbox | ChkBox |
| 8 | Filter | Fltr |
| 9 | Text | Txt |
| 10 | Radio Button | RBtn |
| 11 | Combo box | CmbBox |
| 12 | Sector | Str |
| 13 | Icon | Icn |
| 14 | Tooltip | Ttip |
| 15 | Column Header | Colhdr |
| 16 | Table rows | Tbltr |
| 17 | Multi select | Multslt |
| 18 | Option | Optn |

* Sample structure of variable name “Screen Name \_object Type \_Object name”
* Sample code has given below for Page object model with Multi language & Parameterized object creation.



## **Module Specific Utilities**

* Module Specific Utilities used to implement the functions, which are specific to feature.
* Module Specific Utility class extends “Business Utilities” class Whereas common Business utility functions may require to access in feature specific utility
* Example: “Navigate back to Home page” is common function, which is available in “Business Utility”, to access that function in Module specific utility we have to extend it rather creating object.

## **Test Scripts**

* Test scripts package implemented to write test scripts for the “test cases”.
* Each class contains multiple test scripts.
* Step by step procedure to write test script.

**Step 1**: Call the listeners

**Step 2:** Create test script with unique name as per test case title.

**Step 3:** To read the test data we have to pass the mapper parameter in test script.  
 “Map<String, String> mapper”.

**Step 4:** Store the test data in variables.

Ex: String testData=mapper.get(“UserName”)

**Step 5**: access the web-elements using the page objects

**Step 6:** call the functions & do the validations as per the test steps



## **Utilities**

* Utilities package implemented to develop different Utilities classes.
* Each class under utilities perform the different actions from beginning of test suite execution to end of suite execution.

### **Base Initializer**

* Base Initializer is the main driver class, which will perform the different actions before and after each execution
* Few major actions listed below
  + Object initialization for [page objects](#_Page_Objects)
  + Launch browser and login to application.
  + Language selection EN/DE.
  + Close browser.
  + Take screenshots.
  + Create report with unique name.
  + Send report.
  + Place extent report in shared drive (when run as remotely).

### **Business Utilities**

* + Utility class implemented for re-usable functions specific to Application.
  + Implemented functions related to switch framework (reactJS)
  + To develop the business functions we have been used cockpit application object properties.
  + To use Switch framework functions you have to modify the object properties as per your application.

### **Custom listener’s**

Custom Listeners implemented to do certain action like:

* + **onTestStart** :
    - It will add test script to extent report
  + **onTestSuccess :** 
    - Test status will be update as passed in report.
    - Takes screenshot based on run configuration
  + **onTestFailure :** 
    - Test status update as failed in report
    - Automatically takes screenshot.

### **Excel Actions**

Excel Actions class implemented for access the excel file. It has implemented functions related to excel listed below:

* + - Get row
    - Get column
    - Read cell
    - Write cell
    - Save Excel.

### **Page Actions**

Page action class implemented to perform the webpage related actions using selenium, few functions listed below for understanding the purpose of page actions class.

* + - ClickOnElement
    - Enter value
    - Clear
    - Close window
    - Switch to new window
    - Wait until progress complete

### **Password Encryption**

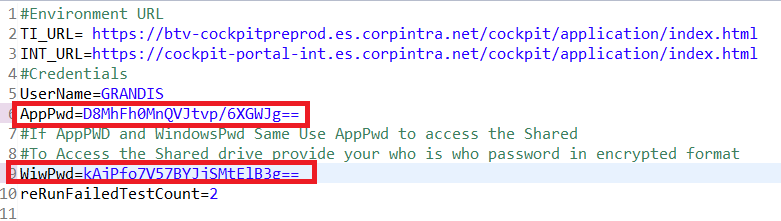
Password Encryption class implemented to Encrypt and Decrypt the password

* Password Encryption class has to run individually (run as java application )
* After executed the class will generate the encrypted password and display encrypted password in console output

**Note**: Execute same class whenever password changes.

* Once the password encrypted, Copy the encrypted password and paste it in the property File. Screenshot given below for the reference.
* Generate encrypted password for windows credentials. It required while placing the reports in the shared drive.

**Note**: if application password and windows password both are same, provide the same encrypted password in both places

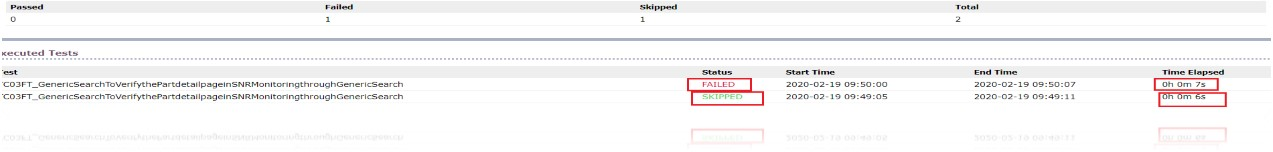
Refer the property file screenshot for easy understanding.

### **Retry Failed Test**

Retry Failed Test Class executes the failed test cases with the number of times the user requests. i,e the Retry happens based on the max retry count specified for each of the test cases.

* When Test suite is run, Retry happens on individual test cases, once the test case fails. Immediately the retry happens before moving on to new Test Case Execution.
* In the Report view, we would be seeing the Results of the previous retry runs of the test cases shown as SKIPPED, and the last run test result would give the exact test result as PASS/FAIL.
* Tests running multiple times accounts to the increase in the Total test count than the actual number when Tests run for one time.
* For the Failed Rerun test-case results adds to the SKIPPED test result. There will also be increase in the overall execution time.

Refer the sample screenshot for representation in report when re-run failed test multiple times.



### **Test Annotation Transformer Listener**

* This Class reads the Excel sheet that comprises of all the list of test cases with the “RunMode” column details provided as YES/NO.
* YES indicates the tests needs to be as part of the test execution under specified group.
* NO indicates the tests should not Disabled and not be run during the execution.
* Tests that configured as “No” indicates that tests get skipped, skipped test will not counted in reports.
* This Class implements IAnnotationTransformer one of the testng listener which is capable of setting the enable attribute of the @Test Annotation to false.(By default this attribute will always hold the value as true) where true being – the test-case is enabled and will be executed , false being the testcase is disabled and will not be executed.
* We have to specify the “Test Annotation Transformer listener” class path for each class under "TestScripts" package.
* If test not specified in excel by default it consider to run.

Refer the attached excel for sample configuration.



### **Image comparison**

Image Comparison class is used for verifying the existence of the image at a particular place and comparing the expected Image with the actual image using different mechanism.

* + Simple Validation

Using isDisplayed() Function of selenium.

* + IMAGE PRESENT OR NOT

Verify the image is present or not using JavaScript, which checks for the arguments like Width, certain other dimensions of the Image

* + IMAGE COMPARISON TESTING BETWEEN TWO IMAGES

Using Yandel QA tool, which supports to take a screenshot of particular Web Element and compares the expected Image with Actual Image.

* + PIXEL COMPARISON TESTING

It performs image comparison testing pixel by pixel. (A large number of pixels together combined to make an image) User needs to provide both the expected and the actual Image Files as the input

* + Image comparison approach supports all the formats are JPEG, PNG, GIF, BMP and WBMP. For the image type of svg, it first needs to be converted to Buffered Image jpeg/png and then can be used.

## **Run config**

Run Configuration package implemented for configuration of run parameters.

### **Dynamic TestNG**

* The main purpose of implementing Dynamic TestNG class is for make the "Group of Tests" as run configuration parameter.
* Dynamic TestNG class is to create a testNG.xml at run time.
* We have to Run-as Java Application
* Defined the run configuration parameters listed below:
* Steps to set the Run configuration   
  DynamicTestNG🡪Right click and select Run as🡪 Run Configuration 🡪 Java Application click on Environment 🡪 Click on New then add below parameters

|  |  |
| --- | --- |
| **Parameter** | **Parameter value** |
| Browser | chrome/edge/IE/Firefox |
| Environment | TI/INT/PROD |
| ExecutingIn | local/remote |
| ScreenShot | OnFail/Both |
| EmailFrom | Any email id. |
| EmailTo | Any recipient email id |
| SuiteName | SmokeTest/RegressionTest |
| Language | EN/DE |

**Execution configuration:**

* Here we can execute a entire “test scripts” package (or) specific classes
* List of Run configuration parameters
* Run specific class: specific test class to run

Sample code:



**Note**: Provided class name should have test scripts by providing @Test annotation

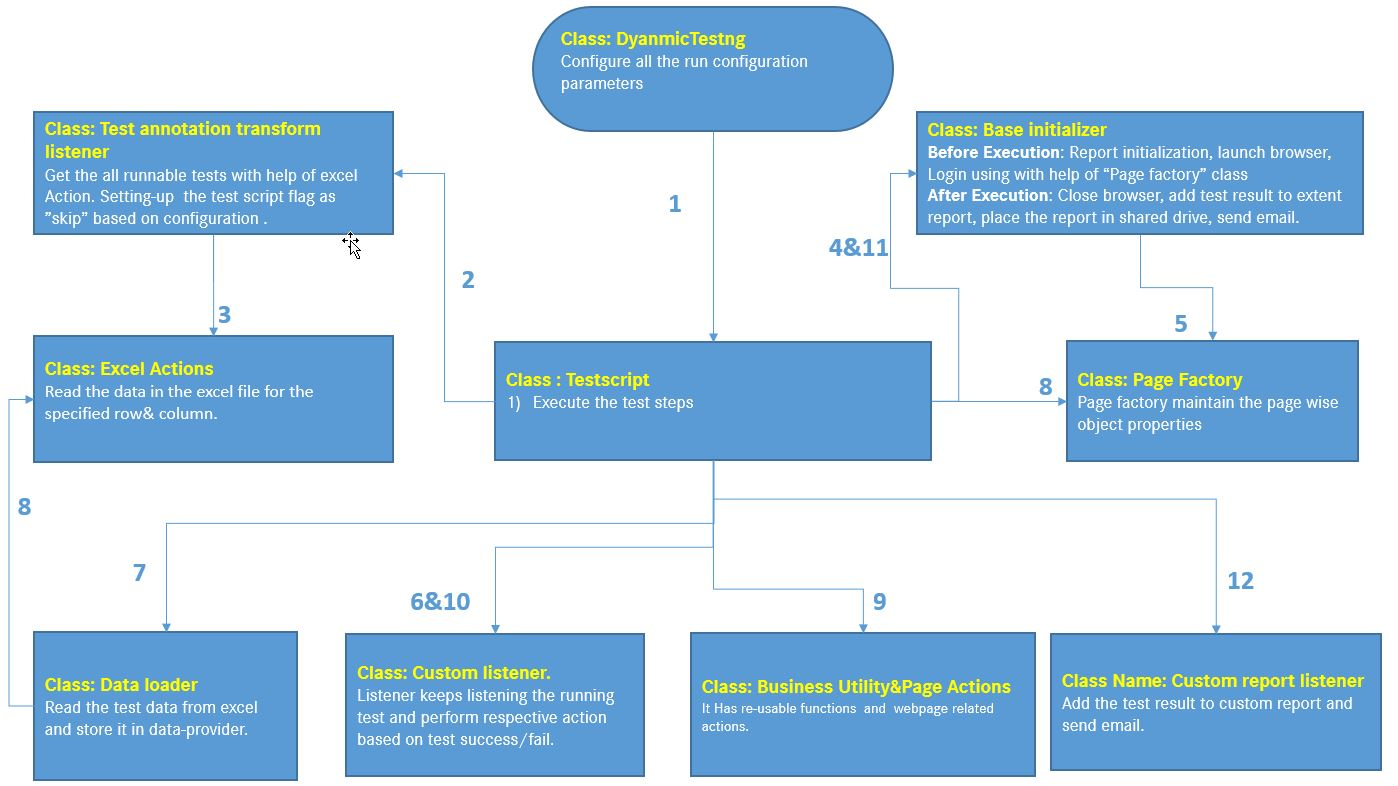
* Run as suite:

Sample code:



# **Execution Flow**

* Below flow chart describes the execution flow from starting point to end.
* To start the execution we have run to run-as java application in “Dynamic TestNg” class.

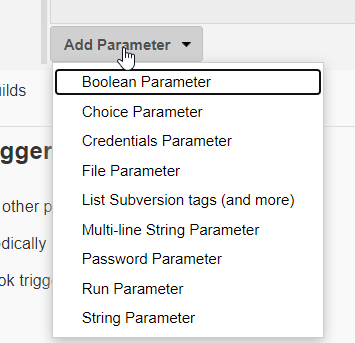
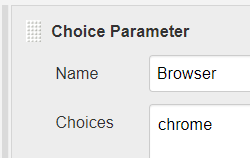
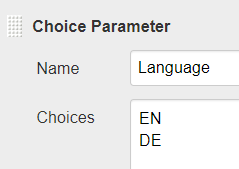


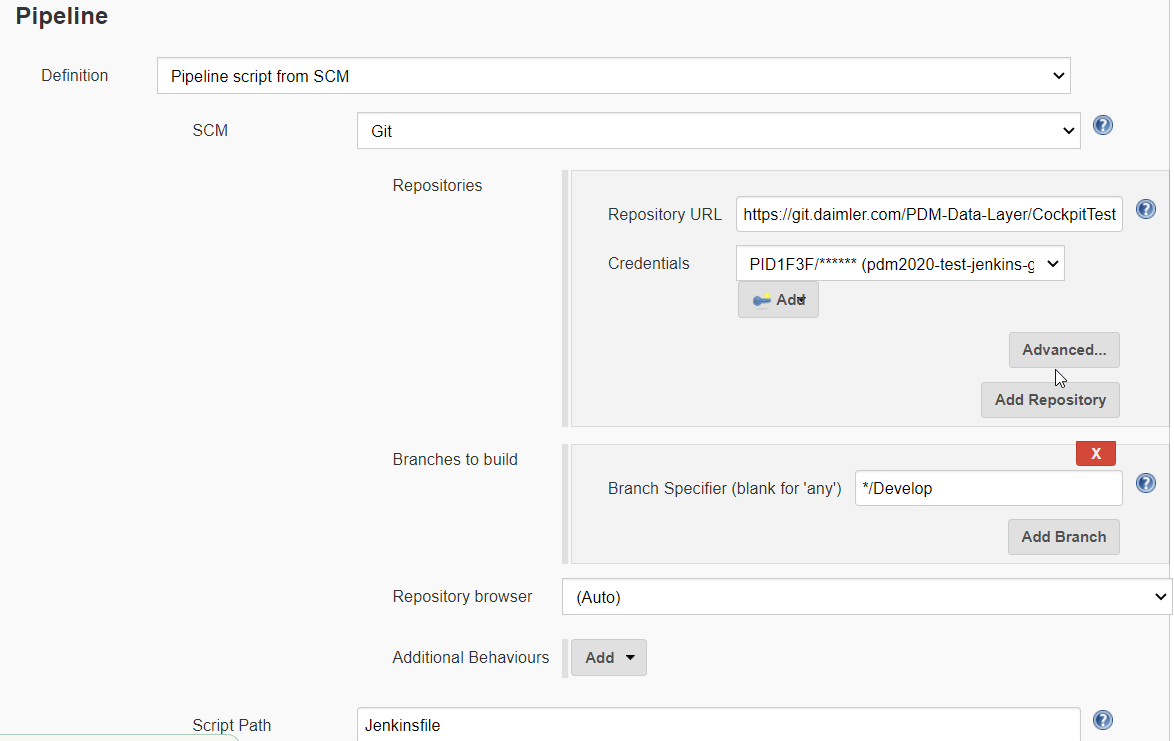
# **Jenkin Job Configuration:**

Jenkins Job configuration details given below.

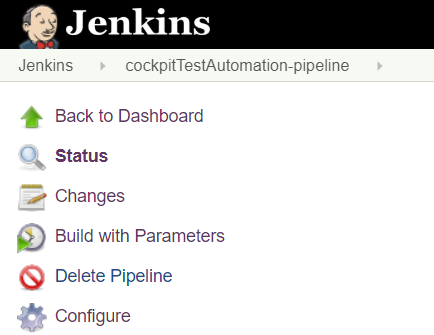
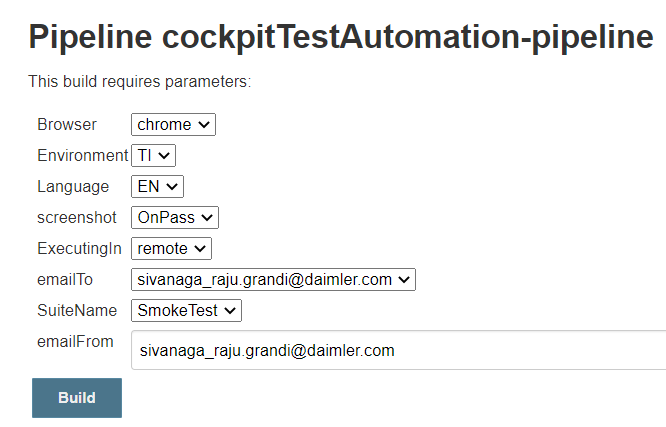
1. Click on “New Item” in Jenkin dashboard



1. Enter the job name
2. Select “pipepline” option
3. Click on “OK”
4. Click on “This project is parameterized “ option  
   
5. Click on Add Parameter and select required “choice parameter” or “String parameter” option  
   
6. Add Run configuration parameters  
   Example:  
    
7. Pipeline: Select Definition as “Pipeline script from SCM” option
8. SCM : Git
9. Provide Repository and credentials (using token)
10. Branch Specifier : provider your git branch name   
    example : master
11. Script path: Jenkinfile
12. Click on Save



Running the job:

1. Click on job name in Jenkin dashboard
2. Click on build with parameter option  
   
3. Click on Build Button  
   

## **Pre requisites to run simple script on your Application:**

**Note**: Install all necessary software’s, refer the installation guide [click here](#_Installation_of_Tools)

Step 1: Download the framework from the Repository

[https://git.daimler.com/IT-QIE-FOSS/test-automation-selenium-framework-with-switch](file:///C:\Users\grandis.APAC\AppData\Roaming\Microsoft\Word\Reports)

Step 2: Copy project to your workspace. Open Eclipse and import this project as existing maven project.

Step 3: Open TestData.properties file and update following value.

a. Provide the Application URL TI/INT/PROD.

b. UserName and AppPwd.

Use com.daimler.cockpitframework.utilities/PasswordEncryptDcrypt.java class to encrypt your password before copying to property file.

c. reRunFailedTestCount

Step 4: Update Maven project. Right click on pom.xml->Maven-> Update Project... To download all dependency files.

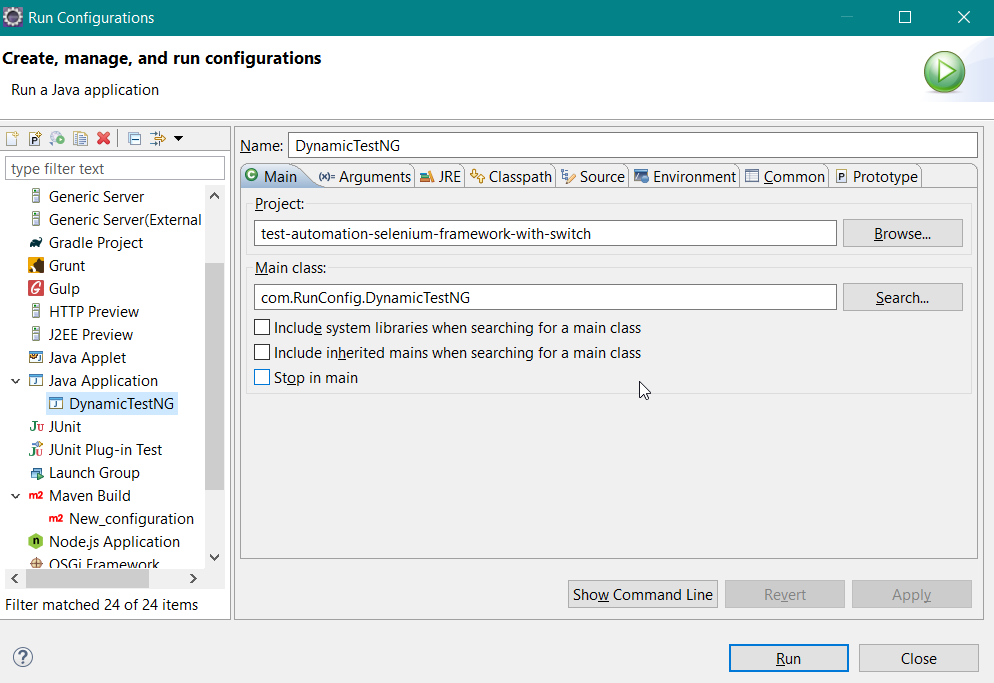
Step 5: Write a sample test method in com.daimler.cockpitframework.testscripts/SampleTestScripts.java.

Step 6: If Test script required Test Data, fill the test data in “ConfigurationFile.xlsx” in respective sheet.

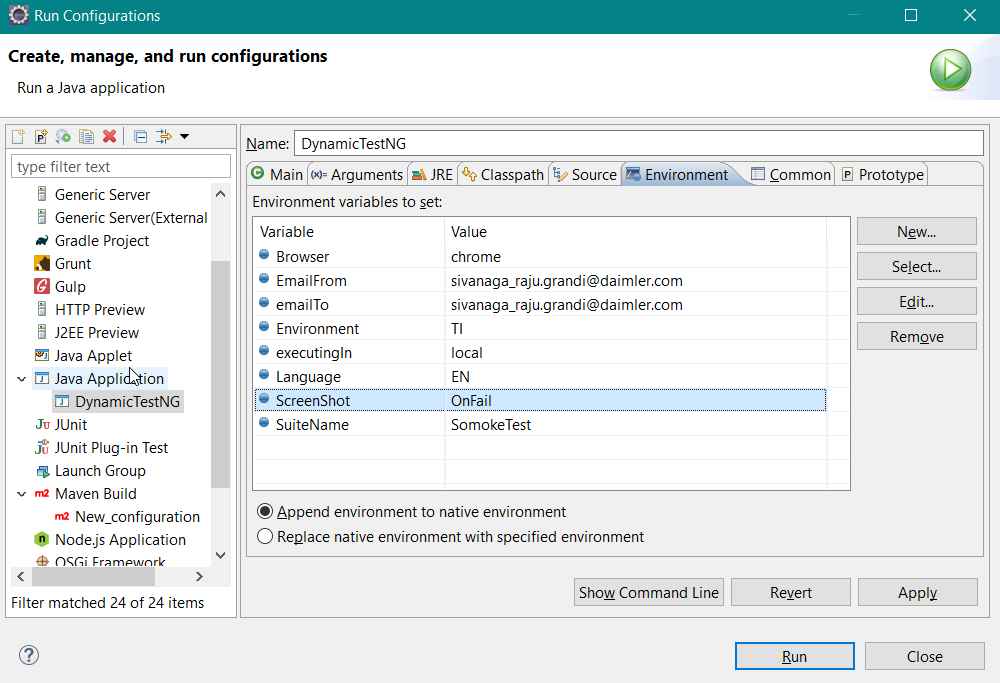
Step 7:

1. **Run as Java Application :**

Right click in com.RunConfig/DynamicTestNG.java and run as java application to run test



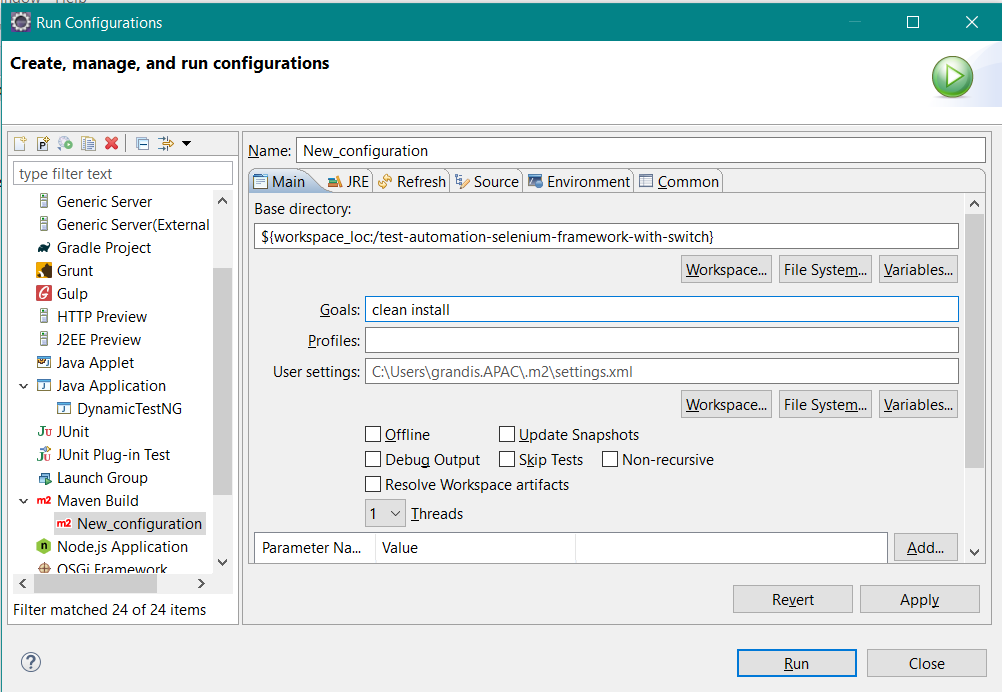
Click on Environment Tab 🡪 set the Environment Variables 🡪 Click on Run



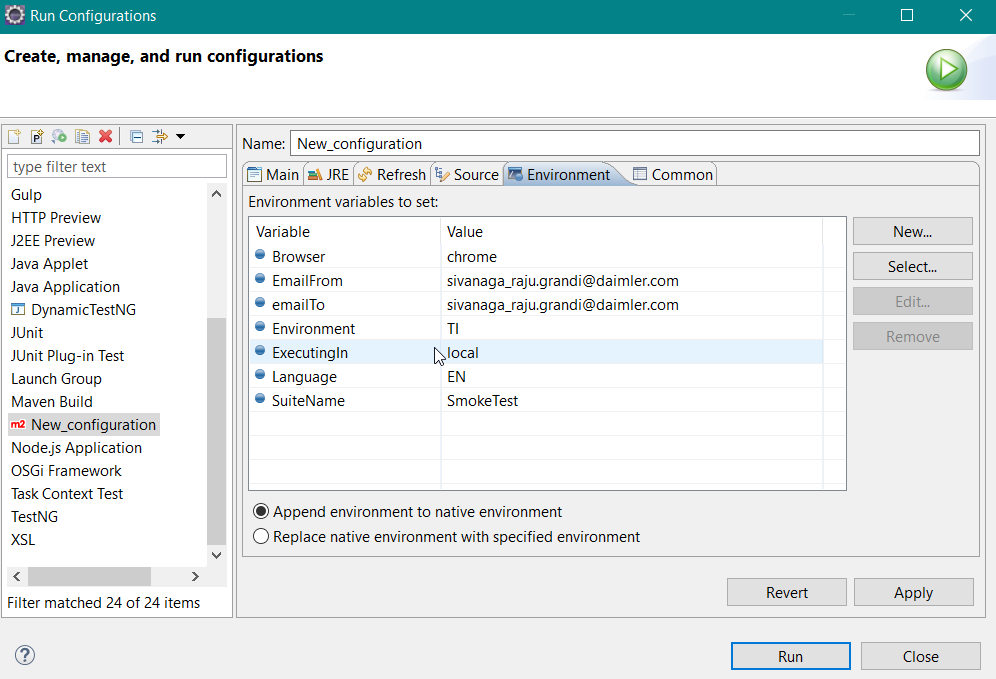
1. **Run as Maven :**

Right click in pom.xml 🡪 run as maven 🡪 in run configuration

(Or)  
Right click on project 🡪 run configuration.



Click on Environment Tab 🡪 set the Environment Variables 🡪 Click on Run



Thank You ☺