NITECO – TEST AUTOMATION FRAMEWORK

GUIDELINE

JUN 2016

Record of change

\*A - Added M - Modified D - Deleted

**Contents**

[1. Niteco Automation Test Framework 4](#_Toc456014838)

[1.1. Architecture 4](#_Toc456014839)

[1.2. Structure 6](#_Toc456014840)

[1.3 Test case flow 9](#_Toc456014841)

[1.4 Process to design test 10](#_Toc456014842)

[1.5 Steps to create project structure and implement scripts 10](#_Toc456014843)

[1.6. Notes when running scripts 11](#_Toc456014844)

[2. Basic git and guide to clone code, create branch and push code to git 12](#_Toc456014845)

[2.1 Download and install gitbash: 12](#_Toc456014846)

[2.2 Create repository, clone source code from bitbucket, create a new branch to code and push code to bitbucket 12](#_Toc456014847)

[3. Example script by using Niteco – Automation Framework 17](#_Toc456014848)

[3.1 Use data driven with excel and @Dataprovider 18](#_Toc456014849)

[3.2 Use Data driven with excel 21](#_Toc456014850)

# Niteco Test Automation Framework

This framework is hybrid framework type. It is combination of Common Library and Data Driven framework.

## 1.1. Architecture

Fra

**Automation Framework Core**

System.properties

Configuration

LocalReportListener

Resource

PropertiesHelper

Testdata.xlsx

DriverUtil

ExcelUtil

**REPORT**

**TEST**

**SUITES**

ReadExcel

Constant

LogUtil

TakeScreenShot

Page’s Action

CommonOperation

Page1’s Action

Page2’s Action

Listener

LocalTestListener

LocalSuiteListener

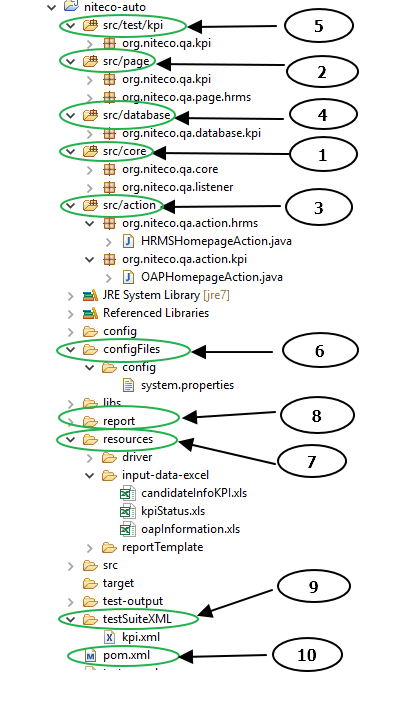
Object Repository

Page1’s web element

Page2’s web element

LocalSuiteExcelListeners

## 1.2. Structure



After clone code and import to project, you will see packages:

1. **src/core:**

* *Introduction:* This package is core of our selenium project, this package will include all core functions that will be able to use in every automation projects (different webs, apps, …).
* *Important note:* You do not need to update anything in this package. In case while writing script, If you need to write new core functions, please let me know so that I can review and decide those functions will be added in core or not.
* *The main classes in Core package:*

+ APIUtils (not used yet)

+ AppUtil (not used yet)

+ BootstrapReporter: contains the functions that will use for report

+ ExcelUtils: contains the functions that read and implement excel files

+ DriverUtil: contains the initiates browser functions

+ Constant: contains the constant variables that will use in the project

+ LogUtil: contains functions that generate log file when running script

+ TakeScreenShot: contains take screen shot functions

+ CommonOperation: contains common functions that can be used in every projects (click, select, check, drag and drop, …)

+ PropertiesHelper: contains functions that read from [system.properties] file or other configuration files in folder [config]

+ LocalReportListener: contains fucntions for report

+ LocalSuiteExcelListener: contains fucntions for report

+ LocalSuiteListener: contains fucntions for report

+ LocalTestListener: contains fucntions for report

1. **src/page:**

* *Introduction:* This package contains the locator for each page following Page Object Model. Currently, each class will include locators of 1 separate page

1. **src/action:**

* *Introduction:* This package contains the actions that implement in each page following Page Object Model. Currently, each class will include actions of 1 separate page

1. **src/database:**

* *Introduction*: This package contains functions that read a Specific excel file and 1 class contains excel file paths
* *Important Notes:*

Currently we support 2 ways of data driven

* One for using excel and @DataProvider
* Another for using only excel

1. **Src/test:**

* *Introduction:* This package contains scripts for test suites

1. **Config folder:**

* ***I****ntroduction****:*** This folder contains config files using for each project

+ system.properties: to configure all parameters related to system such as path to project folder

+ Config.properties: to configure all parameters related execute testing such as baseURL, Test Data file, sheet name, etc.

+ log4j.xml

* *Importtant note****:*** You can use the parameters in these file by using or writing more core functions base on functions in [PropertiesHelper] in src/core

1. **Resources folder:**

* *Introduction:* This folder contains sub folders:

+ driver: This folder contains the executable files to run many other browsers ( chromedriver.exe, iedriver.exe, …)

+ input-data-excel: This folder contains data excel files

+ reportTemplate

1. **report folder:**

* *Introduction:*after running completely, the test report is generated automatically in this folder.

1. **testSuiteXML:**

* *Introduction:* This folder contains xml files that include which test suite will run and which browser will run

1. **Pom.xml file:**

* *Introduction:* This file contains libraries needed for Selenium project

## 1.3 Test case flow

Core

DriverUtil.class

TakeScreenShot.class

LogUtil.class

PropertieseHelper.class

CommonOperation.class

ExcelUtil.class

Constant.class

Test

Test.class

Test03 method

Test02 method

Test01 method

Page Action

Test data

Testdata.class

Page02\_Action.class

Page01\_Action.class

Page object

Page02.class

Page01.class

## 1.4 Process to design test

Analyze business/ test cases

Break AUT to separated page

Identify the all actions of each page

Identify the web elements need to be used in each action

Build project structure

## 1.5 Steps to create project structure and implement scripts

Step1. Create the repository on the github/bitbucket

Step2. Create Maven project

Step3. Commit and put your project to the repository on step1

Step4. Import core library

Step5. Create the Object repository for each page.

5.1 Create the package page such as org.niteco.[project name].page

5.2 Create each class is one different page

5.3 Define all web elements

Step6. Define actions

6.1 Create package org.niteco.[project name].action

6.1 Create each class is one different page

6.2 Define all actions

Step7. Create excel test data for each test if need

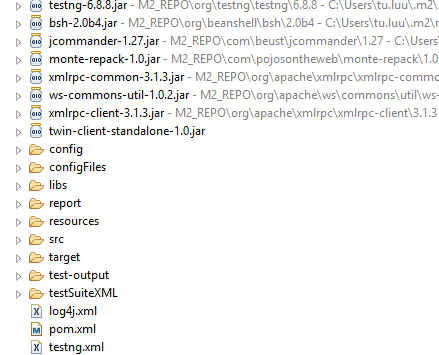
Step8. Create class TestData to define test data for each test

Step9. Update the system.properties/config.properties file

Step10. Create [testsuite].xml to include test classes/methods

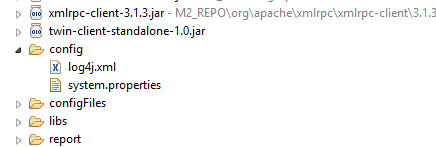
## 1.6. Notes when running scripts

1. *After clone code and import to project, you will not see the libs folder. You can go to: K:\Testers\_Team\Setup\Selenium-Webdriver, cop libs folder and paste into your project*



1. *After clone code and import to project, you will only see [configFiles] folder with empty system.properties, config.properties files. You need to create a new folder [config] folder with log4j.xml, system.properties file edited due to your projects and workspace.*

Structure after created successfully:

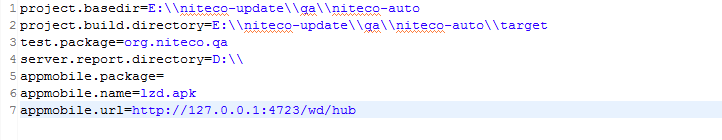


log4j.xml file:



* **Note:** After created log4j.xml file, you need to add the location of your log4j.xml file to the Classpath in Eclipse.

Click Run 🡪 Run Configurations 🡪 Classpath (tab 🡪 User Entries 🡪 Add folder 🡪 navigate to the folder that contains your log4j.xml file (As in my project is: [config] folder) 🡪 Apply 🡪 Run

+ system.properties file: You can add more properties in this file due to your project

# 2. Basic git and guide to clone code, create branch and push code to git

## Download and install gitbash:

* Go to: <https://git-scm.com/downloads> to download and install gitbash

## Create repository, clone source code from bitbucket, create a new branch to code and push code to bitbucket

***+ Step 1: Open gitbash application***

***+ Step 2: Go to the folder that you want to create a repository* by command:**

cd [path-to-folder-A]

***+ Step 3: Initiate git* by command:**

git init

git remote add origin <http://bitbucket.niteco.se:7990/scm/qa/qa.git>

***+ Step 4: Clone source code from gitlab to repository* by command:**

git clone <http://bitbucket.niteco.se:7990/scm/qa/qa.git>

Note: At this step, you maybe have to enter your username and password in bitbucket for authorization

***+ Step 5: Get the latest code and switch to master branch:***

* Go to the folder that you want to create a repository by command:

cd [path-to-folder-A]

* Get latest code by command:

git fetch

Note: At this step, you maybe have to enter your username and password in bitbucket for authorization

* Checkout master branch by command:

git checkout master

* Get the latest code of master branch from bitbucket by command:

git rebase origin/master

***+ Step 6: Create new branch from maser branch to code***

* Go to the folder that you want to create a repository by command:

cd [path-to-folder-A]

* Create new branch name ‘newBranch’ by command:

git branch newBranch

* Switch to newBranch branch to continue coding by command:

git checkout newBranch

***+ Step 7: Coding in* newBranch *branch***

***+ Step 8: After coding finishes, we need to push code from your branch to bitbucket:***

* Go to the folder that you want to create a repository by command:

cd [path-to-folder-A]

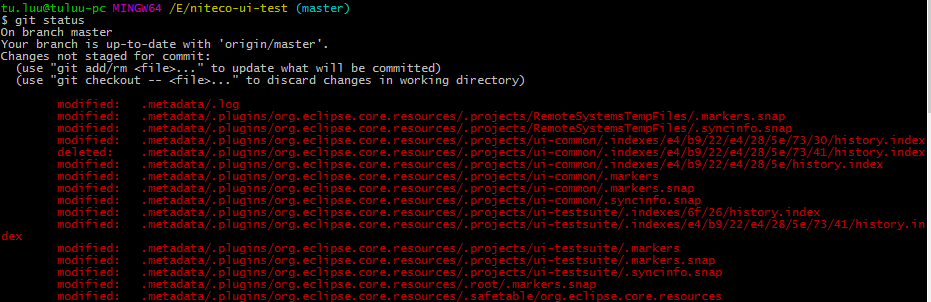
* Check status of branch by command:

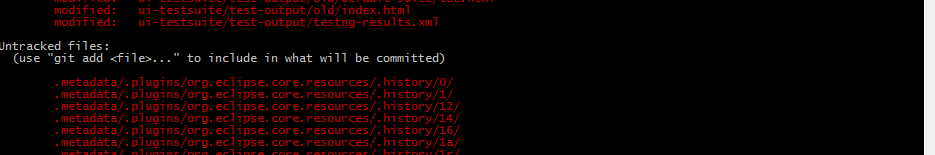
git status

* Add changes by commands:

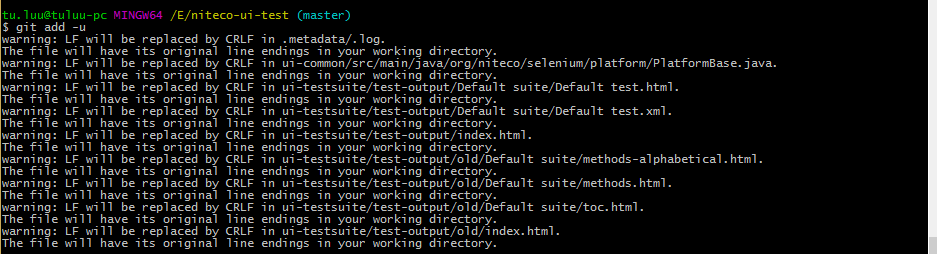
git add –u

(**Note that**: after command “git status”, we will get 2 parts of changed files: *“Changes not staged for commit”* and *“Untracked files:”*

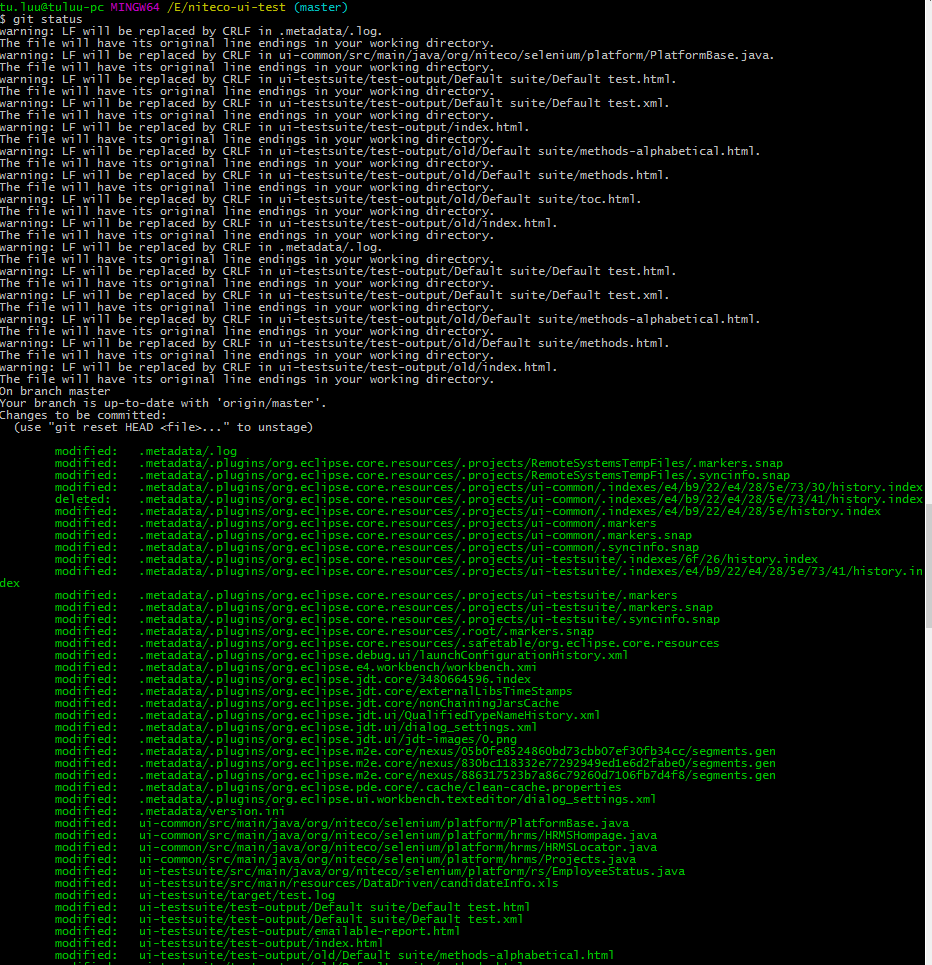


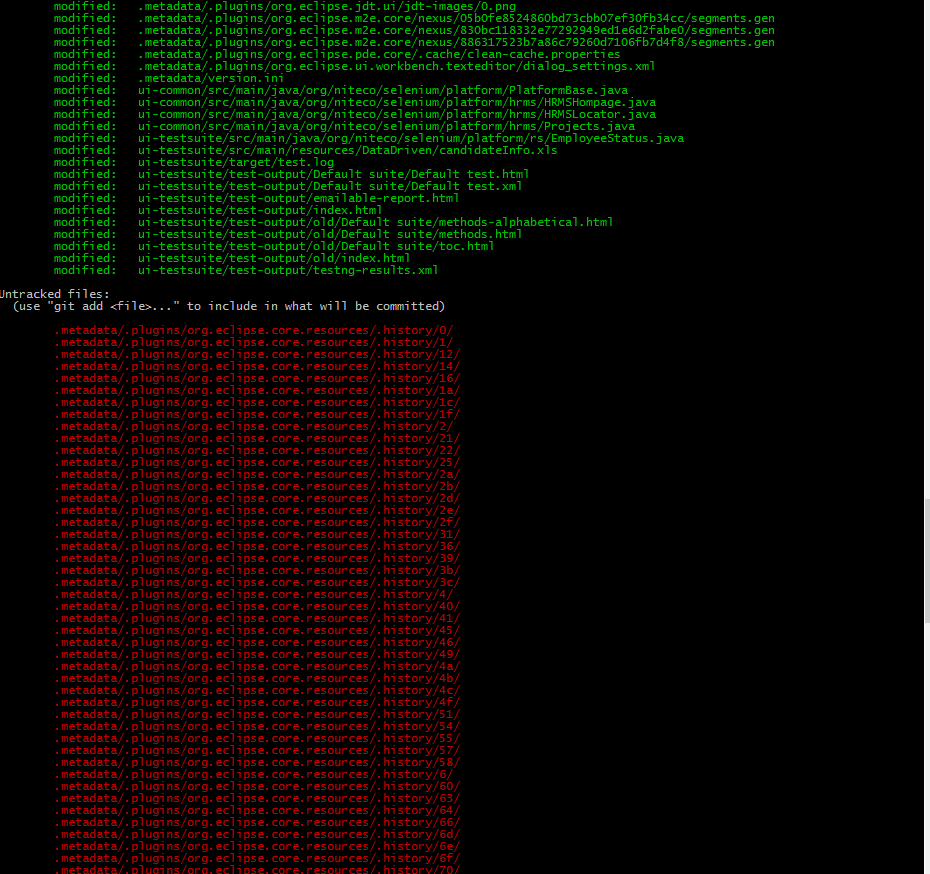


*Command “git add -u” only add ALL the changed files in “Changes not staged for commit”.*



*Check which changed files added successfully (the green files are added, the red files are not added) by command: git status*





*We have to check in “Untracked files:” part and add each changed file that we want to add by command: git add [file]. Then check to make sure that file was added successfully by command: git status*



* Commit changes:

git commit –m “description of this commit”



* Push commit to your branch in gitlab:

git push origin newBranch:newBranch

**Note: Some other git commands:**

+ Command to check which branch you are in: git branch

+ Command to delete a branch name ‘branchA’: git branch branchA -D

+ Command to see the commits: git log –oneline

+ Command to revise your code to origin: git stash

# 3. Example script by using Niteco – Test Automation Framework

* Assume that we have 3 test cases as bellow:

TC\_01:

Steps:

1. Open site: <http://newtours.demoaut.com/>
2. Input valid user, password
3. Click Login

Expected Results:

System redirect to <http://newtours.demoaut.com/mercuryreservation.php>

TC\_02:

Steps:

1. Open site: <http://newtours.demoaut.com/>
2. Input invalid user, valid password
3. Click Login

Expected Results:

System redirect to <http://newtours.demoaut.com/mercurysignon.php>

TC\_03:

Steps:

1. Open site: <http://newtours.demoaut.com/>
2. Input valid user, invalid password
3. Click Login

Expected Results:

System redirect to <http://newtours.demoaut.com/mercurysignon.php>

* Three test cases have the same steps with different data. So we can group them into one test case (test login) by writing one method with three different data.

## 3.1 Use data driven with excel and @Dataprovider

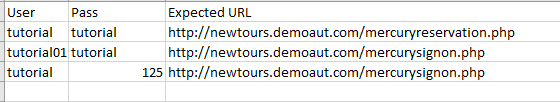
* Create class LoginPage

|  |
| --- |
| **package** org.niteco.qa.newtoursdemo.page;  **import** org.openqa.selenium.By;  **public** **class** LoginPage {  **private** By txtUser = By.*name*("userName");  **private** By txtPwd = By.*name*("password");  **private** By btnSignIn = By.*name*("login");  **public** By getTxtUser() {  **return** txtUser;  }  **public** **void** setTxtUser(By txtUser) {  **this**.txtUser = txtUser;  }  **public** By getTxtPwd() {  **return** txtPwd;  }  **public** **void** setTxtPwd(By txtPwd) {  **this**.txtPwd = txtPwd;  }  **public** By getBtnSignIn() {  **return** btnSignIn;  }  **public** **void** setBtnSignIn(By btnSignIn) {  **this**.btnSignIn = btnSignIn;  }  } |

* Create class LoginAction

|  |
| --- |
| **package** org.niteco.qa.newtoursdemo.action;  **import** org.niteco.qa.core.CommonOperation;  **import** org.niteco.qa.newtoursdemo.page.LoginPage;  **public** **class** LoginAction **extends** CommonOperation {  LoginPage objLoginPage = **new** LoginPage();  **public** LoginAction(String br){  *browser* = br;  }  **public** **void** setUser(String user){  *driver*.findElement(objLoginPage.getTxtUser()).sendKeys(user);  }  **public** **void** setPwd(String pwd){  *driver*.findElement(objLoginPage.getTxtPwd()).sendKeys(pwd);  }  **public** **void** clickSignin(){  click(*browser*,objLoginPage.getBtnSignIn());  }  **public** **void** login(String user, String pwd){  setUser(user);  setPwd(pwd);  clickSignin();  }  } |

* Create file excel NewTours\_TestData.xlsx with sheet’s name is “TestLogin”



* Update the config.properties file to define test data file and sheetName

|  |
| --- |
| #Base URL  baseURL=http://newtours.demoaut.com/  #Define test data file  dataFile =E:/Automation/Selenium/Project/qa/niteco-auto/resources/input-data-excel/NewTours\_TestData.xlsx  #Define sheet name  loginData=TestLogin |

* Create class TestData to define test data for this test case

|  |
| --- |
| **package** org.niteco.qa.database.kpi;  **import** java.io.IOException;  **import** java.util.Properties;  **import** org.niteco.qa.core.ExcelUtil;  **import** org.niteco.qa.core.PropertiesHelper;  **import** org.testng.annotations.DataProvider;  **public** **class** TestData {  **static** Properties *p*;  @DataProvider(name="loginData")  **public** **static** Object[][] getLoginData() **throws** IOException {  *p* = PropertiesHelper.*readConfig*();  Object[][] data = ExcelUtil.*getExcelData*(*p*.getProperty("dataFile"),*p*.getProperty("loginData"));  **return** data;  }  } |

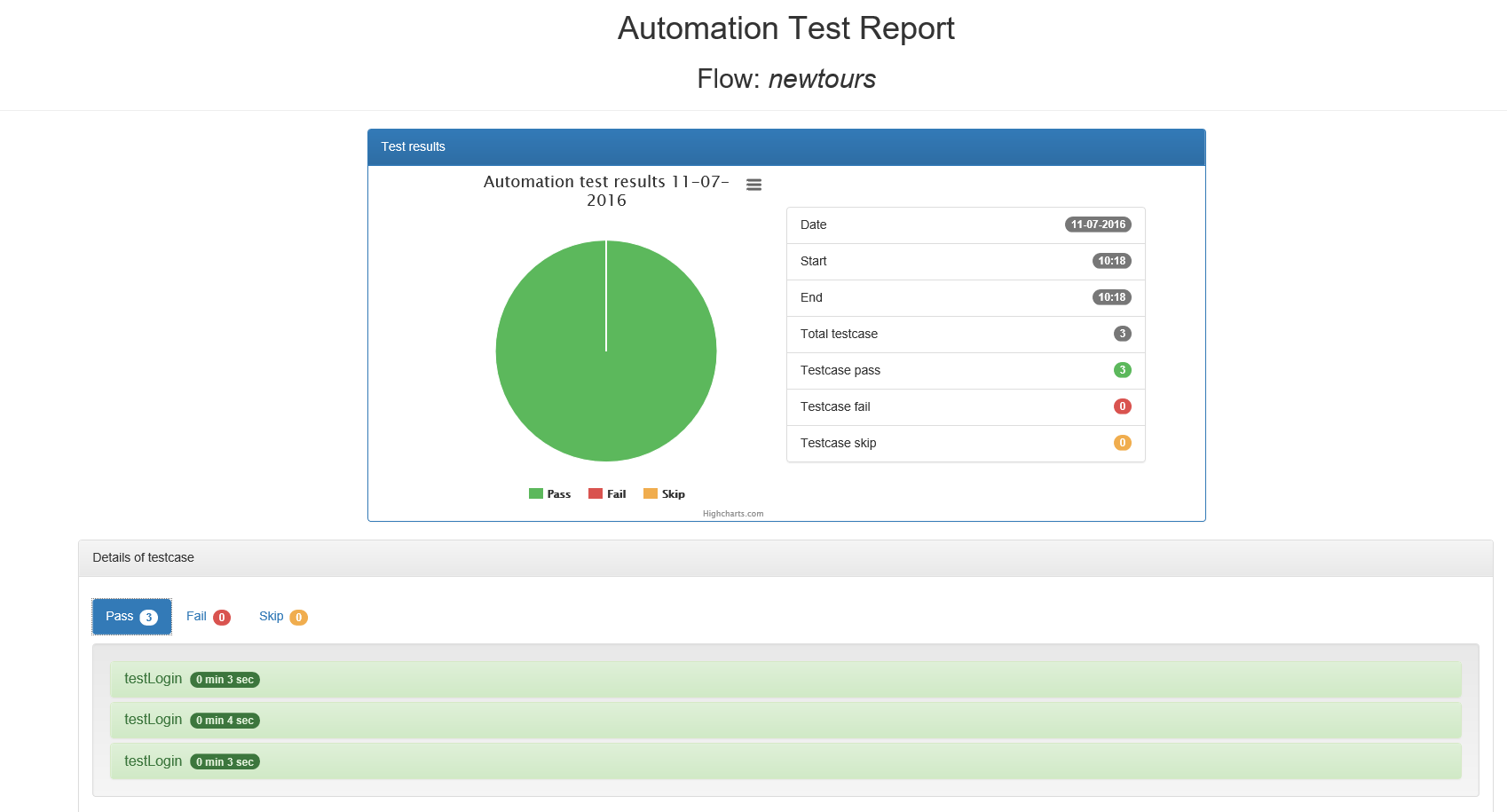
Create class TestLogin

|  |
| --- |
| **package** org.niteco.qa.newtoursdemo;  **import** java.io.IOException;  **import** java.util.Properties;  **import** org.niteco.qa.core.CommonOperation;  **import** org.niteco.qa.core.DriverUtil;  **import** org.niteco.qa.core.PropertiesHelper;  **import** org.niteco.qa.database.kpi.TestData;  **import** org.niteco.qa.newtoursdemo.action.LoginAction;  **import** org.testng.Assert;  **import** org.testng.annotations.AfterMethod;  **import** org.testng.annotations.BeforeMethod;  **import** org.testng.annotations.Parameters;  **import** org.testng.annotations.Test;  **public** **class** TestLogin **extends** CommonOperation {  LoginAction objLoginAction;  @Parameters({"browser"})  @BeforeMethod  **public** **void** setupBeforeTest(String browser) **throws** IOException{  Properties p = PropertiesHelper.*readConfig*();  *driver* = DriverUtil.*getInstance*(browser);  objLoginAction = **new** LoginAction(browser);  *driver*.get(p.getProperty("baseURL"));  }  @Test(dataProvider="loginData", dataProviderClass=TestData.**class**)  **public** **void** testLogin(String user, String pwd, String expectedURL){  objLoginAction.login(user, pwd); Assert.*assertTrue*(*driver*.getCurrentUrl().contains(expectedURL));  }  @AfterMethod  **public** **void** teardownMethod(){  *driver*.quit();  }  } |

* Create the test suite

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  <suite name=*"newtours"* parallel=*"false"*>  <listeners>  <listener class-name=*"org.niteco.qa.listener.LocalTestListener"* />  <listener class-name=*"org.niteco.qa.listener.LocalSuiteListener"* />  <listener class-name=*"org.niteco.qa.listener.LocalReportListener"* /> </listeners>  <test name=*"login"*>  <parameter name=*"browser"* value=*"chrome"*></parameter>  <classes>  <class name=*"org.niteco.qa.newtoursdemo.TestLogin"*/>  </classes>  </test>  </suite> |

* Run test: *The test will be repeated 3 times with 3 rows data in excel file. the report file is generated automatically.*



## 3.2 Use Data driven with excel

You only change the class the config file, TestLogin, TestData and change the file NewTours\_TestData.xlsx to .xls

* Change the config file by adding new parameter

|  |
| --- |
| dataFileXLS =E:/Automation/Selenium/Project/qa/niteco-auto/resources/input-data-excel/NewTours\_TestData.xls |

* Create class TestData

|  |
| --- |
| **package** org.niteco.qa.database.kpi;  **import** java.io.IOException;  **import** java.util.ArrayList;  **import** java.util.Properties;  **import** org.niteco.qa.core.ExcelUtil;  **import** org.niteco.qa.core.PropertiesHelper;  **import** org.testng.annotations.DataProvider;  **public** **class** TestData {  **public** ArrayList<String> user;  **public** ArrayList<String> password;  **public** ArrayList<String> exptURL;  /\*\*  \* UserDatabase  \* **@param** user  \* **@param** password  \*/  **public** TestData(ArrayList<String> user, ArrayList<String> password, ArrayList<String> exptURL){  **this**.user = user;  **this**.password = password;  **this**.exptURL = exptURL;  }  /\*\*  \* UserDatabase  \*/  **public** TestData(){  user = **new** ArrayList<String>();  password = **new** ArrayList<String>();  exptURL = **new** ArrayList<String>();    }  **public** **void** setData(String DataFile, String Sheet, Object... opParams){  System.***out***.println("file is:" + DataFile);  String[][] testData;  **try** {  testData = ExcelUtil.*getDataFromExcelFile*(DataFile,Sheet,opParams);  **for**(**int** i = 0; i<testData.length; i++)  {  //type.add(Integer.valueOf(testData[i][0]));  user.add(testData[i][0]);  password.add(testData[i][1]);  exptURL.add(testData[i][2]);  }  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }    /\*\*  \* Get user By Index  \* **@param** index  \*/  **public** String getUserByIndex(**int** index){  **return** user.get(index);  }    /\*\*  \* Get password by Index  \*/  **public** String getPasswordByIndex(**int** index){  **return** password.get(index);  }    /\*\*  \* Get expected URL by Index  \*/  **public** String getExptURLByIndex(**int** index){  **return** exptURL.get(index);  }    } |

Create class Test Login

|  |
| --- |
| **package** org.niteco.qa.newtoursdemo;  **import** java.io.IOException;  **import** java.util.Properties;  **import** org.niteco.qa.core.CommonOperation;  **import** org.niteco.qa.core.DriverUtil;  **import** org.niteco.qa.core.PropertiesHelper;  **import** org.niteco.qa.database.kpi.TestData;  **import** org.niteco.qa.newtoursdemo.action.LoginAction;  **import** org.testng.Assert;  **import** org.testng.annotations.AfterMethod;  **import** org.testng.annotations.BeforeMethod;  **import** org.testng.annotations.Parameters;  **import** org.testng.annotations.Test;  **public** **class** TestLogin **extends** CommonOperation {  LoginAction objLoginAction;  TestData data;    @Parameters({"browser"})  @BeforeMethod  **public** **void** setupBeforeTest(String browser) **throws** IOException{  Properties p = PropertiesHelper.*readConfig*();  *driver* = DriverUtil.*getInstance*(browser);  objLoginAction = **new** LoginAction(browser);  data = **new** TestData();  data.setData(p.getProperty("dataFileXLS"), p.getProperty("loginData"));  *driver*.get(p.getProperty("baseURL"));  }  @Test  **public** **void** testLogin(){  //Define test data  String user= data.getUserByIndex(0);  String pwd = data.getPasswordByIndex(0);  String exptURL = data.getExptURLByIndex(0);  objLoginAction.login(user, pwd);  Assert.*assertTrue*(*driver*.getCurrentUrl().contains(exptURL));  }  @AfterMethod  **public** **void** teardownMethod(){  *driver*.quit();  }  } |

* Run test, the test will execute one time with the first row data

