<http://sampleapp.tricentis.com/101/app.php>

Selenium

Testing team:

1. To prove that the system is working fine as per the client requirements.
2. Signoff/certify- Production.
3. Requirements----we will write test scenarios and test cases.
4. Identify the issues/defects in the system and log those issues in the system.

* Selenium is open-source test automation tool
* It allows parallel testing.

Key components of selenium are

* selenium web driver-🡪now we are using this
* selenium IDE---🡪it contains less features
* *selenium grid-*🡪*it is used to multiple machines and* *browsers*

->exclusively used for testing web-application on multiple Browsers, like Firefox, Chrome, IE, Edge, Safari, Opera

->selenium supports multiple operating systems (platforms) like **Windows, Mac OS, Linux, and Solaris**

->selenium supports multiple programming languages like java, python, c #, java script, ruby.

**maven project contains:**

src\main\java: -in this we are writing main functionalities & common utilities

src\main\resources: -here external resources purpose we are using

src\test\java: -whatever we are writing all the testcases are written here

src\test\resources: -excel and csv file

pom.xml: -for managing all the decencies all the decencies are written here.

Whatever you want all the decencies are available in maven repository

Web driver is mediator between selenium programme and web browser.

Chrome------chromedriver.exe

Firefox------------geckodriver.exe

Edge---------------edgedriver.exe

Internet explorer----iedriver.exe

Locators:

Selenium Locators are used for identifying the web elements on the web page. To access the HTML elements from a web page locator are used. In Selenium, we can use locators to perform actions on text boxes, checkboxes, links, radio buttons, list boxes, and other web elements. Locators help us in identifying the objects.

1. **id**🡪driver. find Element (By.id(“id-name”))

2. **Name**🡪 driver. find Element (By.name(“name”))

3. **CSSSelector**🡪 click on ...dots>copy>copy CSS Selector

4. **xpath** 🡪 click on ...dots>copy>copy xpath

5**. linkText** 🡪 must has to start with tag Name as <a, static Hyperlink

6. **Partial link Text**--> must has to start with tag Name as <a, Dynamic Hyperlink

7. **className**

8. **TagName**🡪 Whenever u want to find group of similar web Elements on a webpage

linktext --- Inbox(182) ----> Static ---> constant

Partiallinktext ---- Inbox ---> dynamic ---> dynamically changes the values

To writing any text we are using following driver instructions

driver. findElement (By.*id*("txtUsername")).sendKeys("Admin");

For clicking any button, we are using following driver instructions

driver. findElement (By.*name*("Submit")). click ();

For navigate one site to another site the code is

driver. Navigate().to("https://gmail.com/");

driver. Navigate().to("https://google.co.in");

driver. Navigate (). back (); in this case its coming to Gmail

driver. Navigate ().forward (); here its navigate to google

To open any of the website we are using driver instruction like below

driver. get("https://opensource-demo.orangehrmlive.com/");

To open browser:

**import** org.openqa.selenium.chrome.ChromeDriver;

System.*setProperty*("webdriver.chrome.driver","C:\\chromedriver\_win32\\chromedriver.exe");

*driver* =**new** ChromeDriver();

*driver*.manage().window().maximize();

(or)

**import** CommonUtil.TestBrowser;

driver = TestBrowser.*OpenChromeBrowser*();

**For highlighting web element border**

**public** Web Element findElement (By by) **throws** Exception

{

WebElement elem = *driver*.findElement(by);

**if** (*driver* **instanceof** JavascriptExecutor)

{

((JavascriptExecutor)*driver*).executeScript("arguments[0].style.border='3px solid red'", elem);

}

**return** elem;

}

Dropdown:

**import** org.openqa.selenium.support.ui.Select;

In selenium dropdown can be handled in 3 ways

1. SelectByIndex
2. SelectByValue
3. SelectByVisibleText

**SelectByIndex:**

For every dropdown developer will give index we are using that index by using SelecByIndex, index will start with ‘0’.

First, we need to create object for dropdown box

Select dropdown1 = new Select (*driver*. findElement (By.*id*("location\_country")));

Dropdown1.selectByIndex(1);

To find list of web elements in a dropdown

List<WebElement> elements=dropdown1.getOptions();

**for**(WebElement element:elements)

{

System.***out***.println(element.getText());

}

**SelectByValue:**

For every visible text developer will give value, first we need to create object for dropdown

Select dropdown1 = new Select (*driver*. findElement (By.*id*("location\_country")));

Dropdown1.selectByValue(“af”);

Whatever developer will give that value we are assign

<option value=’AF’>Afghanistan</option>

Afghanistan🡪visible text

’AF’🡪value

**3.SelectByVisibleText:**

Select dropdown1 = new Select (*driver*. findElement (By.*id*("location\_country")));

Dropdown1.selectByVisibleText(“India”);

TestNG Data Provider:

**import** org.testng.annotations.DataProvider;

Executing test into multiple times with different sets of input data

We can handle data provider in 3 ways.

1. Providing-data-by-using-Array
2. Providing data in data provider
3. Providing data in excel

**Providing-data-by-using-Array:**

@DataProvider(name = "Locations1")

**public** **static** Object Test1() **throws** Exception {

Object[][] data = **new** Object[2][7];

// 1st row

data[0][0] ="https://opensource-demo.orangehrmlive.com/";

data[0][1] = "Admin";

data[0][2] = "admin123";

data[0][3] = "Kukatpalli2";

data[0][4] = "India";

data[0][5] = "Telanana";

data[0][6] = "Hyderabad";

// 2nd row

data[1][0] ="https://opensource-demo.orangehrmlive.com/";

data[1][1] = "Admin";

data[1][2] = "admin123";

data[1][3] = "Washingtone1";

data[1][4] = "United States";

data[1][5] = "Florida";

data[1][6]= "USA";

**return** data;

}

@Test(dataProvider="Locations1")

**Providing data in data provider:**

@DataProvider(name = "Locations")

**public** **static** Object[][] Test1() **throws** Exception {

**return** **new** Object[][] {

{ "https://opensource-demo.orangehrmlive.com/","Admin", "admin123","Miyapur5","Hyderabad","Telangana" },

{ "https://opensource-demo.orangehrmlive.com/","Admin", "admin123","Nizampet5","Hyderabad","Telangana" },

{ "https://opensource-demo.orangehrmlive.com/","Admin", "admin123","Kuktpalli5","Hyderabad","Telangana" }

};

}

@Test(dataProvider="Locations")

**Providing data in excel:**

**import** ExcelUtil.ExcelApiTest4;

@DataProvider(name = "Locations1")

**public** **static** Object[][] Authentication1() **throws** Exception {

ExcelApiTest4 eat = **new** ExcelApiTest4();

Object[][] testObjArray = eat.getTableArray("C://HTML Report//OrangeHRM6//Locations.xlsx", "Sheet1");

System.***out***.println(testObjArray.length);

**return** (testObjArray);

}

@Test(dataProvider="Locations1")

Data Provider (in one time login executing diff sets of input data):

**public** **static** **int** *iRow* = 0;

Reporter1 R1;

@DataProvider(name ="AddJobs")

**public** **static** Object[][] Authentication1() **throws** Exception {

ExcelApiTest4 eat = **new** ExcelApiTest4();

Object[][] testObjArray = eat.getTableArray("C:\\HTML Report\\OrangeHRM6\\Add\_jobs.xlsx","Sheet2");

System.***out***.println(testObjArray.length);

**return** (testObjArray);

}

@Test(dataProvider="AddJobs")

**public** **void** Login(String URL, String Username, String Password,String JobTitle, String JobDiscription ) **throws** Exception {

**if** (*iRow*==0)

{

String str="Add\_Jobs";

R1= **new** Reporter1(*driver*,str);

*iRow*=*iRow*+1;

Login(URL,Username,Password);

AddJobDetails(JobTitle,JobDiscription );

}

**else**

{

String str="Add\_Jobs";

R1= **new** Reporter1(*driver*,str);

*iRow*=*iRow*+1;

AddJobDetails(JobTitle,JobDiscription );

}

}

Emailable Report:

**import** org.testng.Reporter;

**TestNG Reports** are the default HTML reports which are generated once the test cases are executed using TestNG. These reports help you to identify the information about test cases and the status of a project. TestNG reports in Selenium have three methods passTest, failTest, and skipTest to check the data about test cases. By looking at the result, you can easily identify how many test cases are passed, failed and skipped.

We have to generate testNG.xml for testNG class. By executing testNG.xml, an emailable report will be generated on test output folder.

TestNG.xml:

<suite name=*"Suite13"*>

<test thread-count=*"5"* name=*"Test"*>

<classes>

<class name=*"Day\_008\_Emailable\_Report.TestExample"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

Java Class:

@Test

**public** **void** Test1() **throws** Exception {

driver = TestBrowser.*OpenChromeBrowser*();

String TestURL = "https://opensource-demo.orangehrmlive.com/";

driver.get(TestURL);

Reporter.*log*("Pass- open Orangehrm1");

driver.findElement(By.*id*("txtUsername")).sendKeys("Admin");

Reporter.*log*("Pass- Enter User Name1");

driver.findElement(By.*id*("txtPassword")).sendKeys("admin123");

Reporter.*log*("Pass- Enter Password1");

driver.findElement(By.*name*("Submit")).click();

Reporter.*log*("Pass- Click on Signin1");

driver.close();

}

Test output folder (Emailable report):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test** | **# Passed** | **# Skipped** | **# Retried** | **# Failed** | **Time (ms)** | **Included Groups** | **Excluded Groups** |
| **Suite13** | | | | | | | |
| [**Test**](file:///C:\Users\admin\Desktop\selinium%20classes\Hybrid_45Labs\OrangeHRM13\test-output\emailable-report.html#t0) | 1 | 0 | 0 | 0 | 17,177 |  |  |

| **Class** | **Method** | **Start** | **Time (ms)** |
| --- | --- | --- | --- |
| **Suite13** | | | |
| **Test — passed** | | | |
| Day\_008\_Emailable\_Report.TestExample | [**Test1**](file:///C:\Users\admin\Desktop\selinium%20classes\Hybrid_45Labs\OrangeHRM13\test-output\emailable-report.html#m0) | 1631543342019 | 17051 |

## Test

### Day\_008\_Emailable\_Report.TestExample#Test1

|  |
| --- |
| **Messages** |
| Pass- open Orangehrm1 Pass- Enter User Name1 Pass- Enter Password1 Pass- Click on Signin1 |

Extend report:

Extent Reports class is **used to generate an HTML report on the specified path**. Used to report (or) proof to the client or manager that test is executes successfully.

We have to generate testNG.xml for testNG class. After executing testNG.xml, screenshots will be captured in specified path.

//step1

**import** com.aventstack.extentreports.ExtentReports;

**import** com.aventstack.extentreports.ExtentTest;

**import** com.aventstack.extentreports.MediaEntityBuilder;

**import** com.aventstack.extentreports.reporter.ExtentHtmlReporter;

**public** **class** ExtentReport\_Test

{

WebDriver driver;

//step2

ExtentTest logger;

ExtentReports extent;

String screenShotPath;

**public** **static** String *TestScriptName* = "TC01\_Add\_Nationality";

**public** **static** String *TestName*;

@Test

**public** **void** Report3\_Test() **throws** Exception {

driver = TestBrowser.*OpenChromeBrowser*();

// step3 start here - ExtendsHtmlReport starts

SimpleDateFormat sdfDate = **new** SimpleDateFormat("yyyy\_MMM\_dd\_h\_mm\_ss\_SSS\_a");

Date now = **new** Date();

String strDate = sdfDate.format(now);

*TestName*=*TestScriptName*+"\_"+strDate+".html";

*TestScriptName*=*TestScriptName*+"\_"+strDate;

tring TestHtmlName="C:/HTML Report/test-output/ExtentReportScreenShots/"+ *TestScriptName* +"/"+*TestName*;

//String TestHtmlName=TestName;

ExtentHtmlReporter reporter=**new** ExtentHtmlReporter(TestHtmlName);

System.***out***.println("Html Report path is : "+TestHtmlName);

extent=**new** ExtentReports();

extent.attachReporter(reporter);

logger=extent.createTest(*TestName*);

//step3 ends here ExtendsHtmlReport ends

String TestURL = "https://opensource-demo.orangehrmlive.com/";

driver.get(TestURL);

findElement(By.*xpath*(OR.*username\_sendkey*)).sendKeys("Admin");

screenShotPath = ExtentReport.*capture*(driver,*TestScriptName*);

logger.pass("Login Page - Entered user Name",MediaEntityBuilder.

*createScreenCaptureFromPath*(screenShotPath).build());

findElement(By.*xpath*(OR.*password\_sendkey*)).sendKeys("admin123");

screenShotPath = ExtentReport.*capture*(driver,*TestScriptName*);

logger.pass("Login Page - Entered Password",MediaEntityBuilder.

*createScreenCaptureFromPath*(screenShotPath).build());

findElement(By.*xpath*(OR.*login\_click*)).click();

screenShotPath = ExtentReport.*capture*(driver,*TestScriptName*);

logger.fail("Login Page - Clicked on Login",MediaEntityBuilder. *createScreenCaptureFromPath*(screenShotPath).build());

//Step4

extent.flush();

driver.close();

}

Custamized emailable report:

We have to generate testNG.xml for testNG class. After executing testNG.xml, emailable report will generated with screenshots.

The purpose of the testNG report custmaziation is Test methods start time and execution time are shown in milliseconds.

**public** **class** TC01\_Skills

{

**static** WebDriver *driver*;

Reporter1 R1;

@Test

**public** **void** TC01\_Report\_Test() **throws** Exception {

*driver* = TestBrowser.*OpenChromeBrowser*();

String str= "TC01\_Login";

R1= **new** Reporter1(*driver*,str);

String TestURL = "https://opensource-demo.orangehrmlive.com/";

*driver*.get(TestURL);

R1.TakeScreenShotAuto(*driver*,"Opened Orange HRM","Pass");

findElement(By.*id*("txtUsername")).sendKeys("Admin");

R1.TakeScreenShotAuto(*driver*,"Username Entered","Pass");

findElement(By.*id*("txtPassword")).sendKeys("admin123");

R1.TakeScreenShotAuto(*driver*,"Password Entered","Pass");

findElement(By.*name*("Submit")).click();

R1.TakeScreenShotAuto(*driver*,"Clicked on Signin","Fail");

*driver*.close();

}

Screenshot: Captures screenshot and saved in specified path.

**import** org.openqa.selenium.TakesScreenshot;

**import** java.io.File;

**import** org.apache.commons.io.FileUtils;

**import** org.openqa.selenium.OutputType;

*driver*.get("https://opensource-demo.orangehrmlive.com/");

TakesScreenshot scrShot1 =((TakesScreenshot)*driver*);

File SrcFile1=scrShot1.getScreenshotAs(OutputType.***FILE***);

FileUtils.*copyFile*(SrcFile1, **new** File("C:\\TC02\_Login\\TC1\_ScreenShot1.jpg"));

Katalon Recorder:

Productive IDE to generate automated tests easily for all platforms and OSs, regardless of application complexity. Powerful recording utility for effortlessly storing all UI elements to maximize reusability. Codeless experience. Infinite testing extension for expert.

Add katalon recorder extension to chrome.

New---Test case name---ok---Record----operate test case---stop

Export-----Java (Webdriver+testNG)

TestNG Parameter:

We will give input data in testNG.xml. Suppose we want to set the global variables such URL settings, username, password or API Keys, there are some values which are constant in all the test cases, in such case we use the TestNG Parameters.

Q. What is TestNG parameter?

A. If I want to read input data from TestNg.xml, I will write in TestNG.xml like in parameter, name and value. In TestNG class, I will write @parameter above method and I will give all parameter names.

Q. How do you do Parameterization?

TestNG Parameters are the arguments that we pass to the test methods. There are two ways through which we can pass the parameters to the test methods:

* **TestNG Parameters**
* **TestNG DataProviders**

testNG.xml:

<suite name=*"Suite"*>

<test thread-count=*"5"* name=*"Test"*>

<parameter name=*"Browser1"* value=*"Chrome"*/>

<parameter name=*"UserName1"* value=*"Admin"*/>

<parameter name=*"Password1"* value=*"admin123"*/>

<parameter name=*"Nationality1"* value=*"Indian669"*/>

<classes>

<class name=*"Day\_012\_TestNG\_Paraneters.TC02\_Login\_Static\_Paarameters1"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

Q. How can improve browser compatibility by using TestNG.xml parameter?

TestNG.xml(Data Driven test): Test will execute multiple times with different sets of input data

<suite name=*"Suite"*>

<test thread-count=*"5"* name=*"Test1"*>

<parameter name=*"Browser1"* value=*"Chrome"*/>

<parameter name=*"UserName1"* value=*"Admin"*/>

<parameter name=*"Password1"* value=*"admin123"*/>

<parameter name=*"Nationality1"* value=*"Indian761"*/>

<classes>

<class name=*"Day\_012\_TestNG\_Paraneters.TC02\_Login\_Static\_Paarameters2"*/>

</classes>

</test> <!-- Test ENDS-->

<test thread-count=*"5"* name=*"Test2"*>

<parameter name=*"Browser1"* value=*"Firefox "*/>

<parameter name=*"UserName1"* value=*"Admin"*/>

<parameter name=*"Password1"* value=*"admin123"*/>

<parameter name=*"Nationality1"* value=*"Indian762"*/>

<classes>

<class name=*"Day\_012\_TestNG\_Paraneters.TC02\_Login\_Static\_Paarameters2"*/>

</classes>

</test> <!-- Test1 ENDS -->

</suite> <!-- Suite -->

@Parameters({"Browser1","UserName1","Password1","Nationality1"})

@Test

**public** **void** Login\_Test(String Browser,String UserName,String Password,String Nationality) **throws** Exception

{

**if** (Browser.equalsIgnoreCase("Chrome")) {

*driver* = TestBrowser.*OpenChromeBrowser*();

}

**else** **if** (Browser.equalsIgnoreCase("FireFox")) {

*driver* = TestBrowser.*OpenFirefoxBrowser*();

}

**else** **if** (Browser.equalsIgnoreCase("IE")) {

*driver* = TestBrowser.*OpenIEBrowser*();

}

**else** {

*driver* = TestBrowser.*OpenChromeBrowser*();

System.***out***.println("Chrome Browser started :" + Browser);

}

TC02\_Login\_Static\_Paarameters2.*OpenOrangeHRM*();

TC02\_Login\_Static\_Paarameters2.*Login*(UserName,Password);

TC02\_Login\_Static\_Paarameters2.*AddNationalities*(Nationality);

*driver*.quit();

}

File upload and download:

File Upload:

String str1="C:\\HTML Report\\EMP\_Photos\\bsnl.txt";

FileUpload(str1);

}

**public** **void** FileUpload(String PhotoPath) **throws** Exception {

findElement(By.*id*("btnAddAttachment")).click();

Thread.*sleep*(1000);

WebElement ProfilePIC =findElement(By.*name*("ufile"));

Actions actions1 = **new** Actions(driver);

actions1.moveToElement(ProfilePIC).click().build().perform();

StringSelection sel = **new** StringSelection(PhotoPath);

// Copy to clipboard

Toolkit.*getDefaultToolkit*().getSystemClipboard().setContents(sel,**null**);

System.***out***.println("selection" +sel);

// Create object of Robot class

Robot robot = **new** Robot();

Thread.*sleep*(2000);

// Press CTRL+V

robot.keyPress(KeyEvent.***VK\_CONTROL***);

robot.keyPress(KeyEvent.***VK\_V***);

// Release CTRL+V

robot.keyRelease(KeyEvent.***VK\_CONTROL***);

robot.keyRelease(KeyEvent.***VK\_V***);

Thread.*sleep*(2000);

// Press Enter

robot.keyPress(KeyEvent.***VK\_ENTER***);

robot.keyRelease(KeyEvent.***VK\_ENTER***);

Thread.*sleep*(8000);

findElement(By.*id*("txtAttDesc")).sendKeys("File uploaded");

screenShotPath = ExtentReport.*capture*(driver,TestScriptName);

logger.pass("FIle Uploaded",MediaEntityBuilder.*createScreenCaptureFromPath*(screenShotPath).build());

findElement(By.*id*("btnSaveAttachment")).click();

}

Photo upload:

String str2="C:\\HTML Report\\EMP\_Photos\\image1.jpg";

PhotoUpload(str2);

**public** **void** PhotoUpload(String PhotoPath) **throws** Exception {

findElement(By.*id*("empPic")).click();

WebElement Choosephoto= findElement(By.*id*("photofile"));

Actions actions = **new** Actions(driver);

actions.moveToElement(Choosephoto).click().build().perform();

StringSelection sel = **new** StringSelection(PhotoPath);

// Copy to clipboard

Toolkit.*getDefaultToolkit*().getSystemClipboard().setContents(sel,**null**);

System.***out***.println("selection" +sel);

// Create object of Robot class

Robot robot = **new** Robot();

Thread.*sleep*(2000);

// Press CTRL+V

robot.keyPress(KeyEvent.***VK\_CONTROL***);

robot.keyPress(KeyEvent.***VK\_V***);

// Release CTRL+V

robot.keyRelease(KeyEvent.***VK\_CONTROL***);

robot.keyRelease(KeyEvent.***VK\_V***);

Thread.*sleep*(2000);

// Press Enter

robot.keyPress(KeyEvent.***VK\_ENTER***);

robot.keyRelease(KeyEvent.***VK\_ENTER***);

Thread.*sleep*(8000);

screenShotPath = ExtentReport.*capture*(driver,TestScriptName);

logger.pass("Photo Uploaded",MediaEntityBuilder.*createScreenCaptureFromPath*(screenShotPath).build());

findElement(By.*id*("btnSave")).click();

}

File Download: (and moving to desination)

In class we have to declare

**public** String DestinationFile;

String str1="//\*[@id='tblAttachments']/tbody/tr/td[2]";

WebElement Element=findElement(By.*xpath*(str1));

String fname = Element.getText(); // To get inner web element text

findElement(By.*linkText*(fname)).click(); // clicks bsnl.txt

String SrcFile="C:\\Users\\USER\\Downloads\\"+fname;

String DestinationFile="C:\\HTML Report\\"+fname;

Thread.*sleep*(6000);

*moveFile*(SrcFile, DestinationFile);

}

**public** **static** **void** moveFile(String src, String dest ) **throws** InterruptedException {

Path result = **null**;

**try** {

result = Files.*move*(Paths.*get*(src), Paths.*get*(dest));

Thread.*sleep*(5000)

}

**catch** (IOException e) {

System.***out***.println("Exception while moving file: " + e.getMessage());

}

**if**(result != **null**) {

System.***out***.println("File moved successfully.");

}

**else**{

System.***out***.println("File movement failed.");

} }

Web Table:

Web tables are commonly used when information has to be displayed in tabular format. There are two types of HTML tables published on the web-

1. **Static tables**: Data is static i.e. Number of rows and columns are fixed.
2. **Dynamic tables**: Data is dynamic i.e. Number of rows and columns are NOT fixed.

A table consists of rows and columns. The table created for a web page is called a web table. Below are some of the important tags associated with a web table:

* **< table >** – Defines an HTML table
* **< th >** – Contains header information in a table
* **< tr >** – Defines a row in a table
* **< td >** – Defines a column in a table

First I will identify no. of columns and no. of rows by using xpath, after that I will iterate all cell values.

**public** **int** iRow;

**public** **void** ExportEmployees(WebDriver driver )**throws** Exception

{

**this**.driver=driver;

}

@Test

**public** **void** ExportAllEmployees( )**throws** Exception

{

driver=TestBrowser.*OpenChromeBrowser*();

call\_allmethods(1);

driver.close();

}

**public** **void** call\_allmethods(**int** iRow )**throws** Exception

{

String str=String.*valueOf*(iRow);

driver.get("https://opensource-demo.orangehrmlive.com/");

findElement(By.*id*("txtUsername")).sendKeys("Admin");

findElement(By.*id*("txtPassword")).sendKeys("admin123");

findElement(By.*id*("btnLogin")).click();

WebElement pim=findElement(By.*xpath*("//\*[@id='menu\_pim\_viewPimModule']/b"));

WebElement emplist=findElement(By.*id*("menu\_pim\_viewEmployeeList"));

Actions actions = **new** Actions(driver);

actions.moveToElement(pim).

moveToElement(emplist).click().build().perform();

ExportEmployees Ae=**new** ExportEmployees();

Ae.ExportEmployees(driver);

Ae.ExportEmployees();

System.*gc*();

}

**public** **void** Export\_Employees()**throws** Exception

{

//No. of columns

String Empcoloums=" //\*[@id='resultTable']/thead/tr/th";

List<WebElement> columns = driver.findElements(By.*xpath*(Empcoloums));

System.***out***.println("No of columns in WebTable : " + columns.size());

//\*[@id="resultTable"]/tbody/tr[1]/td[2]/a

//No.of rows

String EmpRows="//\*[@id='resultTable']/tbody/tr/td[2]";

List<WebElement> rows = driver.findElements(By.*xpath*(EmpRows));

System.***out***.println("No of rows in WebTable : " + rows.size());

ExcelApiTest3 eat = **new** ExcelApiTest3();

**for** ( **int** i=1 ; i<=rows.size() ;i++) // i=1; 1<=40 ; i=i+1

{

**for** ( **int** j=2 ,k=0; j<=columns.size() ;j++,k++) //j=2 ; J<=8 ; j++

{

String str1="//\*[@id='resultTable']/tbody/tr[" + i + "]" + "/td" + "[" + j +"]";

//\*[@id="resultTable"]/tbody/tr[1]/td[2]

WebElement Ele=findElement(By.*xpath*(str1));

Thread.*sleep*(50);

String WebElementText = Ele.getText();

//System.out.println("Get Text Value is from the WebElement: " + valueIneed1);

**if** (WebElementText !=**null**)

eat.PutCellData( "C://HTML Report//OrangeHRM6//Employee-Export.xls","Sheet3",i,k,WebElementText);

**else**

eat.PutCellData( "C://HTML Report//OrangeHRM6//TC01\_EMPExport.xls","Sheet3",i,k,"Blank Data");

}

}

Broken Links:

If hyper link in a web page is not working fine that hyperlink we called it as broken link.

That means we are not able to navigating to destination page.

Q: How to Identify Broken links in a web page?

**Import** java.net.HttpURLConnection;

**Import** java.net.URL;

**Import** java.util.List; --------To find no. of Hyper links

First we have to find no. of hyperlinks are there in a web page.

List<WebElement> links = driver.findElements(By.*tagName*("a"));

System.***out***.println ("Total links are:"+links.size ());

We have to iterate each and every hyper link to check that hyperlink is working fine or not by using for loop.

**for** (**int** i=0; i<links.size (); i++) {

WebElement element = links.get (i);

//By using "href" attribute, we could get the url of the required link

String url=element.getAttribute ("href");

*verifyLink*(url);

}

//Verify Link method starts

We have to write a method to verify a link.

**public** **static** **void** verifyLink (String urlLink) {

//Sometimes we may face exception "java.net.MalformedURLException". Keep the code in try catch block to continue the broken link analysis

**try** {

//Use URL Class - Create object of the URL Class and pass the urlLink as parameter

URL link = **new** URL (urlLink);

// create a connection using URL object (i.e., link)

HttpURLConnection httpConn = (HttpURLConnection)link.openConnection();

//Set the timeout for 2 seconds

httpConn.setConnectTimeout(2000);

//connect using connect method

httpConn.connect ();

//use getResponseCode() to get the response code.

**if**(httpConn.getResponseCode()== 200) {

System.***out***.println(urlLink+" - "+httpConn.getResponseMessage())

}

**if(**httpConn.getResponseCode() == 404) {

System.***out***.println(urlLink+" - "+httpConn.getResponseMessage())}

}

//getResponseCode method returns = IOException - if an error occurred connecting to the server.

**catch** (Exception e) {

//e.printStackTrace();

}

}

Smart Search:

If you searching for something in a search bar you will get some keywords related to what we are searching. In this type of scenario we will use smart search.

For similar keywords we will write locator as xpath of starts with.

StringGoogle\_Input\_text="/html/body/div[1]/div[3]/form/div[1]/div[1]/div[1]/div/div[2]/input";

String KeyWordText ="//\*[starts-with(text(),'selenium')]";

findElement (By.*xpath*(Google\_Input\_text)).sendKeys("Selenium");

First we have to get no. of web elements.

List<WebElement> rows = driver.findElements(By.*xpath*(KeyWordText));

Next iterate all the web elements by using for loop

**for**(WebElement element:rows)

{

String WebElementText = element.getText();

System.***out***.println("Get Text Value is from the WebElement: " + WebElementText);

**if**(WebElementText.equals("seleniumhq"))

{

element.click();

**break**;

}

Method Overloading:

We can write multiple methods with same method name with different input arguments in the same class.

Method overloading is used to search existing product or existing customer or existing employee.

Method overloading increases the readability of the program.

There are two ways to overload the method in java

1. By changing number of arguments

2. By changing the data type.

**int** length = EmployeeID.length();

System.***out***.println("Length of Employee id:"+length);

**if**(length>0)

{

System.***out***.println("Search EmployeeID, EmployeeStatus method invoked");

Search Employee(EmployeeID, EmployeeStatus);

}

**else**

{

System.***out***.println("Search by only EmployeeStatus method invoked");

Search\_Employee(EmployeeStatus);

}

Method Overriding:

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**

Usage of Java Method Overriding:

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

#### Rules for Java Method Overriding:

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

class Walking //parent class

{

void walk()

{

System.out.println("Man walking fastly");

}

}

class Man extends walking //child class

{

void walk()

{

System.out.println("Man walking slowly");

}

}

class OverridingDemo

{

public static void main(String args[])

{

Man obj = new Man();

obj.walk();

}

}

Output is Man walking slowly.

Note: Whenever we are calling overridden method using derived class object reference the highest priority is given to current class (derived class). We can see in the above example high priority is derived class.

Q. Difference between Overloading and Overriding'

* Whenever same method or Constructor is existing multiple times within a class either with different number of parameter or with different type of parameter or with different order of parameter is known as Overloading.

Whenever same method name is existing multiple time in both base and derived class with same number of parameter or same type of parameter or same order of parameters is known as Overriding.

* Arguments of method must be different at least arguments. Argument of method must be same including order.
* Method signature must be different.

Method signature must be same.

* Private, static and final methods can be overloaded.

Private, static and final methods cannot be override.

* Also known as compile time polymorphism or static polymorphism or early binding.

Also known as run time polymorphism or dynamic polymorphism or late binding.

* Overloading can be exhibited both are method and constructor level.

Overriding can be exhibited only at method leave.

* The scope of overloading is within the class.

The scope of Overriding is base class and derived class.

* Overloading can be done at both static and non-static methods.

Overriding can be done only at non-static method.

* For overloading methods return type may or may not be same.

For overriding method return type should be same.

Java Script Executor:

**Import** org.openqa.selenium.JavascriptExecutor;

Selenium WebDriver can encounter problems interacting with a few web elements. For instance, the user opens a URL and there is an unexpected pop-up that will prevent the WebDriver from locating a specific element and produce inaccurate results. This is where JavaScript Executor comes into the picture.

By using JavaScript executor we can perform following operations:

1. We can launch new tab

2. Scroll at particular web Element

3. Send Keys

4. Click

5. Highlight a web Element border

Launch a new tab :( Multiple Windows)

To open a new tab:

((JavascriptExecutor)driver).executeScript("window.open()”); //tab2

((JavascriptExecutor)driver).executeScript("window.open ()"); //tab1

//strore all the open windows in ArrayList string

ArrayList<String> tabs = **new** ArrayList<String>(driver.getWindowHandles());

//Switch to new tab

driver.switchTo().window(tabs.get(1));

driver.get("http://google.com");

//back to MainWindow (Main window is Orange HRM)

driver.switchTo().window(tabs.get(0));

driver.findElement(By.*name*("txtUsername")).sendKeys("Admin");

driver.findElement(By.*name*("txtPassword")).sendKeys("admin123");

findElement(By.*id* ("btnLogin")).click();

driver.switchTo().window(tabs.get(2));

driver.get("https://facebook.com");

2. Scroll at particular web Element:

If you want to take screenshot at bottom of the page or at particular web element, we will use scroll

First we have to create web element object references for the web element which we have to scroll

WebElement Element = findElement(By.*linkText*("Conditions of Use & Sale"));

Next create an object for java script executor

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].scrollIntoView();", Element);-To Scroll

3. Send Keys:

First we have to create web element object reference for which of web element send key is not working.

WebElement Username = findElement(By.*id* ("txtusername"));

Next create an object for java script executor

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].setAttribute('value','Admin')",Username);

4. Click:

First we have to create web element object reference for which of web element click is not working.

WebElement button= findElement(By.*id* ("btnlogin"));

Next create an object for java script executor

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].click();", button);

5. Highlight a web Element border:

First we have to create web element object reference for which web element border is to highlight.

WebElement Username = findElement(By.*id* ("txtusername"));

Next create an object for java script executor

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].style.border='3px solid red'", UserName);

Scroll down the page by 1000 pixel vertical:

It will scroll at some (x,y) position. We cannot measure where it will scroll exactly.

js.executeScript("window.scrollBy(0,1000)");

Action Class:

Actions class is **an ability provided by Selenium for handling keyboard and mouse events**. In Selenium WebDriver, handling these events includes operations such as drag and drop, clicking on multiple elements with the control key

**import** org.openqa.selenium.interactions.Actions;

1. Mouse Hover action

2. Double click

3. Right Click

4. Drag and drop

1. Mouse Hover action:

For some web elements no need to perform click, just need to move mouse on web element , in that cases we will use mouse hover action.

We will use it for web menu options.

We have to do click action on final web element where we need to click.

First we have to create web element object reference for which web element no need to perform click.

WebElement Admin=findElement (By.*xpath*("//\*[@id='menu\_admin\_viewAdminModule']/b"));

Next create an object for action class

Actions actions = **new** Actions(*driver*);

actions.moveToElement(Admin).

MoveToElement(Users).click().build().perform();

2. Double click:

For some web elements we need to perform double click.

First we have to create web element object reference for which web element need to perform double click.

WebElement Copy\_Text\_Button= findElement(By.*xpath*("//\*[@id='HTML10']/div[1]/button"));

Next create an object for action class

Actions actions = **new** Actions(*driver*);

actions.doubleClick(Copy\_Text\_Button).perform();

3. Right Click:

For some web elements we need to perform Right click.

First we have to create web element object reference for which web element need to perform Right click.

WebElement RButton=findElement(By.*xpath*("//span[text()='right click me']"));

Next create an object for action class

Actions actions = **new** Actions(*driver*);

actions.contextClick(RButton).perform();

4. Drag and drop:

To drag and drop a web element to target.

First we have to create web element object references for which web elements need to Drag and drop

WebElement Source\_Drag\_Button= findElement(By.*xpath*("//\*[@id='draggable']"));

WebElement Target\_Drag\_Button= findElement(By.*xpath*("//\*[@id='droppable']"));

Next create an object for action class

Actions actions = **new** Actions(*driver*);

actions.dragAndDrop(Source\_Drag\_Button, Target\_Drag\_Button).perform();

Alert:

An **Alert in Selenium** is a small message box which appears on screen to give the user some information or notification. It notifies the user with some specific information or error, asks for permission to perform certain tasks and it also provides warning messages as well.

Alert interface provides the below few methods which are widely used in [Selenium Webdriver](https://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html).

**import** org.openqa.selenium.Alert;

// Switching to Alert

Alert alert = *driver*.switchTo().alert();

alert.sendKeys("Sudhakar");

// capturing alert message.

String alertMessage= alert.getText();

// displaying alert message

System.***out***.println("ALERT DISPLAYED as: "+alertMessage);

Thread.*sleep*(5000);

// Accepting alert

alert.accept();

driver.switchTo(). alert(). accept();

//Dismiss alret

alert.dismiss();

driver. switchTo (). alert (). dismiss ();

CSS Selector:

Copy element

<input name="txtUsername" id="txtUsername" type="text">

<input name="txtPassword" id="txtPassword" autocomplete="off" type="password">

By id:

Input# txtUsername

Input# txtPassword

findElement(By.*cssSelector*("input#txtUsername")).sendKeys("Admin");

findElement(By.*cssSelector*("input#txtPassword")).sendKeys("admin123");

By class:

<input type="submit" name="Submit" class="button" id="btnLogin" value="LOGIN">

Input.button

findElement(By.*cssSelector*("input.button")).click();

By Attribute

input[id='txtUsername']

findElement(By.*cssSelector*("input[id='txtUsername']")).sendKeys("Admin");

input[id='txtUsername'] [name='txtUsername'] [type=’text’]

By sub string

input[id^='txtUser] --------------Suffix

input[id$='Password'] -------------Prefix

input[id\*='Login'] -------------Suffix or Prefix

By Inner text

In html latest version (3.0) inner text css selector is not allowed. Version (2.0) is allowed.

Incognito:

System cannot maintain any information like last time visited pages means history and any client information and creditanls, which are automatically deleted. It is secure browsing.

It is used only when we need to test secure conditions.

System.*setProperty*("webdriver.chrome.driver","C:\\chromedriver\_win32\\chromedriver.exe");

ChromeOptions options = **new** ChromeOptions();

options.addArguments("incognito");

DesiredCapabilities cap = DesiredCapabilities.*chrome*();

cap.setCapability(ChromeOptions.***CAPABILITY***, options);

driver = **new** ~~ChromeDriver~~(cap);

driver.manage().window().maximize();

driver.get("https://opensource-demo.orangehrmlive.com/");

driver.findElement(By.*name*("txtUsername")).sendKeys("Admin");

driver.findElement(By.*name*("txtPassword")).sendKeys("admin123");

driver.findElement(By.*id*("btnLogin")).click();

Event firing Web driver:

**Import** org.openqa.selenium.support.events.EventFiringWebDriver;

**Import** CommonUtil.EventHandler;

WebDriverEventListener is class in selenium.

Event handler contains various predefined events (methods)

For every action there are before and after events (WebDriverEventListener) which are predefined events.

First we have to create an object driver1

That object (driver1) will monitor events in background whenever particular web element action is performing.

EventFiringWebDriver driver1 = **new** EventFiringWebDriver(*driver*);

EventHandler handler = **new** EventHandler();

driver1.register(handler);

driver1.get("https://opensource-demo.orangehrmlive.com/");

driver1.findElement(By.*name*("txtUsername")).sendKeys("Admin"); driver1.findElement(By.*name*("txtPassword")).sendKeys("admin123");

driver1.findElement(By.*name*("Submit")).click();

Predefined WebDriverEventListener:

**public** **class** EventHandler **implements** WebDriverEventListener{

(Add Unmplemented methods)

@Override

**public** **void** afterAlertAccept(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterAlertDismiss(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**Public** **void** beforeAlertDismiss(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeNavigateTo(String url, WebDriver driver) {

// **TODO** Auto-generated method stub

System.***out***.println("Hi.....beforeNavigateTo Invoked");

}

@Override

**public** **void** afterNavigateTo(String url, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeNavigateBack(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterNavigateBack(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeNavigateForward(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterNavigateForward(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeNavigateRefresh(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterNavigateRefresh(WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeFindBy(By by, WebElement element, WebDriver driver) {

// **TODO** Auto-generated method stub

WebElement elem = driver.findElement(by);

**if** (driver **instanceof** JavascriptExecutor)

{

((JavascriptExecutor)driver).executeScript("arguments[0].style.border='3px solid red'", elem);

}

}

@Override

**public** **void** afterFindBy(By by, WebElement element, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeClickOn(WebElement element, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterClickOn(WebElement element, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeChangeValueOf(WebElement element, WebDriver driver, CharSequence[] keysToSend) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterChangeValueOf(WebElement element, WebDriver driver, CharSequence[] keysToSend) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeScript(String script, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterScript(String script, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeSwitchToWindow(String windowName, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterSwitchToWindow(String windowName, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onException(Throwable throwable, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** <X> **void** beforeGetScreenshotAs(OutputType<X> target) {

// **TODO** Auto-generated method stub

}

@Override

**public** <X> **void** afterGetScreenshotAs(OutputType<X> target, X screenshot) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeGetText(WebElement element, WebDriver driver) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterGetText(WebElement element, WebDriver driver, String text) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** beforeAlertAccept(WebDriver driver) {

// **TODO** Auto-generated method stub

}

}

Keyword Driven:

It is not suitable for big scale project.

We can write multiple methods in a class and we will put this method names (Keywords) in excel sheet.

In another class we have to import supporting class file driverscript. We have to create an object for driverscript and we have to mention excel sheet path. It will execute as per set of keywords in excel sheet.

@Test

**public** **void** TC01\_Test() **throws** Exception

{

DriverScript DS4= **new** DriverScript();

DS4.*mainTest*("C://HTML Report//OrangeHRM6//TC01\_Nationality10.xlsx","Sheet1");

}

Reporter Word:

**import** Reporter\_Example\_Word.ImageAttachmentInDocument1;

**import** Reporter\_Example\_Word.Reporter1;

**import** Reporter\_Example\_Word.WordConvertPDF;

We can get word document and pdf documents in specified path. Once script is executed, all screenshots captured in word document.

It is used to provide execution test evidence to the client and to prove text automation is securely completed.

In html reporting, we have to click each and every screenshot, but here all screenshots captured in a word document and also we can upload this document where ever we want.

Whenever executing script a new word document will create.

We can also convert word document to pdf format.

Create an object for ImageattachmentIndocument

ImageAttachmentInDocument1 IA = **new** ImageAttachmentInDocument1();

IA.CreateHeader("Daily Status Report");

SimpleDateFormat sdfDate1 = **new** SimpleDateFormat("yyyy\_MMM\_dd\_h\_mm\_ss\_a");

Date now1 = **new** Date();

String strDate1 = sdfDate1.format(now1);

System.***out***.print(strDate1);

String TestName="TC01\_Login\_"+strDate1;

Reporter1 R1= **new** Reporter1(driver,"TC01\_Login");

driver = TestBrowser.*OpenChromeBrowser*();

String TestURL = "https://opensource-demo.orangehrmlive.com/";

driver.get(TestURL);

String str=R1.getScreenShotPath(driver);

IA.PasteImage("Pass- open Orangehrm",str);

findElement(By.*xpath*(OR.*username\_sendkey*)).sendKeys("Admin");

str=R1.getScreenShotPath(driver);

IA.PasteImage("Pass- Enter User Name",str);

driver.close();

String Filename=IA.WordOutput\_File(TestName);

WordConvertPDF.*main*(Filename);

In ImageattachmentIndocument, we have specify the path where the word document is to create.

String fileName = "C:\\Screenshots\\" + TestName + ".docx";

Excelutil Apache POI: (Advanced data provider)

To read the input data from excel.

We have to import Excelutil Apache POI supporting Libraries.

1. for .xlsx --->org.apache.poi.XSSF--->Office 2007,2010,2013,2016

2. for .xls---->org.apache.poi.HSSF---> Msoffice-97-2003

import java.io.FileInputStream;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import org.apache.poi.xssf.usermodel.XSSFSheet;

import org.apache.poi.xssf.usermodel.XSSFRow;

import org.apache.poi.xssf.usermodel.XSSFCell;

For all the classes we have to declare object as null

public FileInputStream fis = null;

public XSSFWorkbook workbook = null;

public XSSFSheet sheet = null;

public XSSFRow row = null;

public XSSFCell cell = null;

@Test

**public** **void** hello()**throws** Exception

{

//Create an object foe current class

TC02\_Excel\_Test\_xlsx eat=**new** TC02\_Excel\_Test\_xlsx();

TestURL=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",1,0);

UserName=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",1,1);

Password=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",1,2);

Nationality\_text=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet4",1,3);

// eat.readinputdata();

System.***out***.println("TestURL Cell Value is :"+TestURL);

System.***out***.println("UserName Cell Value is:"+UserName);

System.***out***.println("Password Cell Value is :"+Password);

System.***out***.println("NationalityText Cell Value is:"+Nationality\_text);

System.*setProperty*("webdriver.chrome.driver","C:\\chromedriver\_win32\\chromedriver.exe");

driver =**new** ChromeDriver();

driver.manage().window().maximize();

driver.get(TestURL);

findElement(By.*name*("txtUsername")).sendKeys(UserName); findElement(By.*name*("txtPassword")).sendKeys(Password);

findElement(By.*id*("btnLogin")).click();

driver.quit();

}

Public void means no value return from this method to main method

Public String means value return from this method to main method

**public** String getCellData(String xlFilePath, String sheetName,**int** rowNum,**int** column) **throws** Exception

{

//To highlight the particular cell data

fis = **new** FileInputStream(xlFilePath); //excel File path

workbook = **new** XSSFWorkbook(fis); // workbook open

sheet = workbook.getSheet(sheetName); //Sheet1 open

row = sheet.getRow(rowNum); // 1st will be highlighted

cell = row.getCell(column); // 1th column will be highlighted

//to convert numeric value to string value

**if**(cell.getCellTypeEnum() == CellType.***STRING***)

{

String str6=cell.getStringCellValue();

System.***out***.println("Cell Value is :"+str6);

workbook.close();

fis.close();

**return** str6;

}

**else** **if**(cell.getCellTypeEnum() == CellType.***NUMERIC***)

{

**int** i = (**int**)cell.getNumericCellValue();

String str6 = String.*valueOf*(i);

workbook.close();

fis.close();

**return** str6;

}

**else**

{

String str6=cell.getStringCellValue();

workbook.close();

fis.close();

**return** str6;

}

}

Put Cell Data:

Export data to excel. As it can serve to save the test results back in the Excel sheets

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream; ----------To save Data in Excel sheet

**import** org.apache.poi.xssf.usermodel.XSSFSheet;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**import** org.apache.poi.xssf.usermodel.XSSFRow;

**import** org.apache.poi.xssf.usermodel.XSSFCell;

For all the classes we have to declare object as null

**public** XSSFWorkbook workbook = **null**;

**public** XSSFSheet sheet = **null**;

**public** XSSFRow row = **null**;

**public** XSSFCell cell = **null**;

**public** FileOutputStream fout=**null**;

**public** FileInputStream fis = **null**;

@Test

**public** **void** hello() **throws** Exception

{

TC04\_Excel\_Test\_xlsx eat=**new** TC04\_Excel\_Test\_xlsx();

eat.PutCellData(“C://HTML Report//OrangeHRM6//Test62.xlsx","Sheet1",1,0,"Admin11");

eat.PutCellData( "C://HTML Report//OrangeHRM6//Test62.xlsx","Sheet1",1,1,"admin11");

}

@Test

**public** **void** hello1() **throws** Exception

{

TC04\_Excel\_Test\_xlsx eat=**new** TC04\_Excel\_Test\_xlsx();

eat.PutCellData( "C://HTML Report//OrangeHRM6//Test62.xlsx","Sheet2",1,0,"Admin12");

eat.PutCellData( "C://HTML Report//OrangeHRM6//Test62.xlsx","Sheet2",1,1,"admin12");

}

@Test

**public** **void** hello2()**throws** Exception

{

TC04\_Excel\_Test\_xlsx eat=**new** TC04\_Excel\_Test\_xlsx();

eat.PutCellData(“C://HTML Report//OrangeHRM6//TC01\_EMPExport3.xlsx","Sheet3",1,0,"Admin3");

eat.PutCellData( "C://HTML Report//OrangeHRM6//TC01\_EMPExport3.xlsx","Sheet3",1,1,"admin13");

}

**public** **synchronized** **void** PutCellData(String xlFilePath,String sheetName,**int** rowNum,**int** column,String Text)**throws** Exception

{

Thread.*sleep*(15000);

fis = **new** FileInputStream(xlFilePath);

workbook = **new** XSSFWorkbook(fis);

sheet = workbook.getSheet(sheetName);

**if**(sheet.getRow(rowNum)==**null**)

{

row=sheet.createRow(rowNum);

}

**else**

{

row=sheet.getRow(rowNum);

}

**if**(row.getCell(column)==**null**)

{

cell=row.createCell(column);

}

**else**

{

cell=row.getCell(column);

}

cell = sheet.getRow(rowNum).getCell(column);

cell.setCellValue(Text);

CellStyle cs1 = workbook.createCellStyle();

cs1.setFillForegroundColor(IndexedColors.***WHITE***.getIndex());

cs1.setFillPattern(FillPatternType.***SOLID\_FOREGROUND***);

Font font = workbook.createFont();

font.setColor(IndexedColors.***BLUE***.getIndex());

font.setBold(**false**);

cs1.setFont(font);

System.***out***.println("Text:"+Text);

cell.setCellStyle(cs1);

cell.setCellValue(Text);

fout= **new** FileOutputStream(xlFilePath);

workbook.write(fout);

fout.flush();

fout.close();

workbook.close();

fis.close();

Thread.*sleep*(20000);

}

Data Driven: (ExcelUtil Apache POI)

Executing the test for different sets of input data.

@Test

**public** **void** hello() **throws** Exception

{

ExcelApiTest4 eat=**new** ExcelApiTest4();

**int** RowCount=eat.getRowCount("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1");

System.***out***.println("Row Count :"+RowCount);

RowCount=RowCount-1; // Excluding column headers

**for** (**int** i=1;i<=RowCount;i++)

{

TestURL=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",i,0);

UserName=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",i,1);

Password=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",i,2); Nationality\_Text=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",i,3);

System.*setProperty*("webdriver.chrome.driver","C:\\chromedriver\_win32\\chromedriver.exe");

driver =**new** ChromeDriver();

driver.manage().window().maximize() ;

driver.get(TestURL);

driver.manage().timeouts().implicitlyWait(20, TimeUnit.***SECONDS***);

findElement(By.*name*("txtUsername")).sendKeys(UserName);

findElement(By.*name*("txtPassword")).sendKeys(Password); findElement(By.*id*("btnLogin")).click();

//driver.findElement(By.id("menu\_admin\_viewAdminModule")).click();

WebElement Admin=findElement(By.*id*("menu\_admin\_viewAdminModule"));

//js.executeScript("arguments[0].click();", Admim);

Actions actions = **new** Actions(driver);

actions.moveToElement(Admin).click().build().perform();

driver.findElement(By.*id*("menu\_admin\_nationality")).click();

findElement(By.*id*("btnAdd")).click();

findElement(By.*id*("nationality\_name")).sendKeys(Nationality\_Text);

findElement(By.*id*("btnSave")).click();

driver.quit();

}

Page Object Model:

Page Object Model, also known as POM, is a design pattern in Selenium that creates an object repository for storing all web elements. It is useful in reducing code duplication and improves test case maintenance.

For every web page we will write separate java class and corresponding methods will be implemented. We will create object for every java class in our main test.

(Class name) (Java class) (Method name)

Login page--------------login.java-------------Login();

Nationality page--------Nationality.java-------Add\_Nationality();

Home page---------------Hompage.java-----------Logout();

Uses:

1. Code reusability
2. Easy to maintain Object repository (Web elements)

@BeforeTest

**public** **void** TestSetup()**throws** Exception {

driver = TestBrowser.*OpenChromeBrowser*();

String TestURL = "https://opensource-demo.orangehrmlive.com/";

driver.get(TestURL);

}

@Test

**public** **void** Test() **throws** Exception {

driver.manage().timeouts().implicitlyWait(60, TimeUnit.***SECONDS***);

LoginPage L1 = **new** LoginPage(); //Class name

L1.LoginPage(driver); //Pass the driver in loginpage class

**/\*public** **void** Loginpage(WebDriver driver)

{

**this**.driver = driver;

}\*/

L1.Login(); //Method name

Nationalities N1= **new** Nationalities();

N1.Nationalities(driver);

N1.AddNationality();

HomePage H1= **new** HomePage();

H1.HomePage(driver);

H1.Logout();

}

@AfterTest

**public** **void** TestCloser()**throws** Exception {

driver.quit();

}

Object repository management (While parameterizing)

Public static String uname=” txtUsername”;

Public static String pwd=” txtPassword”;

findElement(By.*name*(uname)).sendKeys(UserName);

findElement(By.*name*(pwd)).sendKeys(Password);

Page Factory Model:

Page Factory is a class provided by [Selenium WebDriver](https://www.browserstack.com/guide/selenium-webdriver-tutorial) to support Page Object Design patterns. In Page Factory, testers use **@FindBy** annotation. The **initElements** method is used to initialize web elements.

**public** **void** LoginPage(WebDriver driver)

{

**this**.driver = driver;

PageFactory.*initElements*(driver, **this**); // initialize web elements.

// To check web elements available or not

}

@FindBy(name="txtUsername")

WebElement UserName1;

@FindBy(name="txtPassword")

WebElement Password1;

@FindBy(id="btnLogin")

WebElement Button1;

**public** **void** Login() **throws** Exception {

UserName1.sendKeys("Admin");

Password1.sendKeys("admin123");

Button1.click();

}

For Dropdown:

@FindBy(id="FirstName") WebElement fname;

Select dropdown = new Select(fname);

dropdown.selectByVisibleText("UNITED STATES");

Q. What is the difference between page factory and page object model?

1.We will write saperate java class file for each web page in both the models.

2. Main test case development is same for both models.

3. In page factory model, in web page classes, while passing driver we will write pageFactory.inintElement(driver, this);

4. Object repojetry changing.

Waits: (Synchronization issue)

Q. How do you handle synchronization in selenium?

Q. What are different types of waits in selenium?

1. Implicit Wait
2. Explicit Wait
3. Fluent wait
4. Implicit Wait: Implicit wait is global rule applicable for all web elements whenever performance issues, driver will wait for the web element as per implicit time specified.

**import** java.util.concurrent.TimeUnit;

*driver*.manage().timeouts().implicitlyWait(10, TimeUnit.***SECONDS***);

1. Explicit Wait: Whenever the web elements not loading with in the specified implicit wait time, such web elements we need to handle by using explicit wait.

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**import** org.openqa.selenium.support.ui.WebDriverWait;

WebDriverWait wait= new WebDriverWait(*driver*,120);

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*linkText*("Logout"))) ;

If web element loading in 30 sec then 31st sec it will click on web element, no need to wait 120 sec (It is max. explicit wait time)

If we write 120 sec in implicit wait, it will take more time when we write incorrect we id (troughs no such element found in console after 120 sec). So, it is not recommended.

3. Fluent wait: It is also used to handle specific web elements on the web page. The webdriver will look for the web element at regular time intervals as per pooling frequency time.

It is depreciated method—not using.

**import** org.openqa.selenium.support.ui.FluentWait;

**import** org.openqa.selenium.support.ui.Wait;

Wait<WebDriver> wait = **new** FluentWait<WebDriver>(*driver*)

.~~withTimeout~~(120, TimeUnit.***SECONDS***)

.~~pollingEvery~~(40, TimeUnit.***SECONDS***)

.ignoring(NoSuchElementException.**class**);

WebElement Logout1 = wait.until(**new** Function<WebDriver,WebElement>() {

**public** WebElement apply(WebDriver driver) {

**return** driver.findElement(By.*xpath*("/html/body/div[1]/div[1]/div[2]/ul/li[2]/a"));

}

});

Pooling frequency time (40 sec) means for every 40 sec web driver will go and check html page. If first 40 sec it is not loaded, it will go for next 40 sec...If it loaded in the first 40 sec, it won’t go for next 40 sec.

Priority:

We can write multiple main test methods by using priority.

@Test(Priority=1)

We can also write multiple main test methods without using priority, we have to follow numerical order or alphabetical order in method name.

Frames: Frame is a separate html page calling into main page.

Right click on frame------->Click on view page source.

Frame is used to call another html page in the current web page.

Frames can be handled in 3 ways:

1. Switch into frame by Index
2. Switch into frame by name
3. Switch into frame by webelement reference

1.Switch into frame by Index:

To find out the no. of frmaes on a web page:

List<WebElement> frames = *driver*.findElements(By.*tagName*("iframe"));

System.***out***.println("Numnber of frames: " + frames.size());

**int** count=frames.size();

*driver*.switchTo().frame(frames.get(0));

findElement(By.*name*("firstname")).sendKeys("Ravi");

findElement(By.*name*("lastname")).sendKeys("Selenium");

findElement(By.*id*("subject1")).sendKeys("hi");

*driver*.switchTo().defaultContent();

*driver*.switchTo().frame(frames.get(1));

findElement(By.*name*("firstname")).sendKeys("Kiran");

findElement(By.*name*("lastname")).sendKeys("Java");

*driver*.switchTo().defaultContent();

//driver.quit();

2. Switch into frame by name:

*driver*.switchTo().frame("Google\_ContactForm");

findElement(By.*name*("firstname")).sendKeys("Selenium");

findElement(By.*name*("lastname")).sendKeys("Sudhakar");

findElement(By.*name*("subject1")).sendKeys("Selenium Sudhakar");

*driver*.switchTo().defaultContent();

*driver*.switchTo().frame("Amazon\_ContactForm");

findElement(By.*name*("firstname")).sendKeys("Ramu");

findElement(By.*name*("lastname")).sendKeys("TCS");

findElement(By.*name*("subject")).sendKeys("Ramu TCS");

*driver*.switchTo().defaultContent();

3. Switch into frame by webelement reference:

*driver*.switchTo().frame(*driver*.findElement(By.*id*("GoogleContactForm")));

findElement(By.*name*("firstname")).sendKeys("Selenium");

findElement(By.*name*("lastname")).sendKeys("Sudhakar");

findElement(By.*name*("subject1")).sendKeys("Selenium Sudhakar");

*driver*.switchTo().defaultContent();

*driver*.switchTo().frame(findElement(By.*id*("AmazonContactForm")));

findElement(By.*name*("firstname")).sendKeys("Sunita");

findElement(By.*name*("lastname")).sendKeys("TCS");

findElement(By.*name*("subject")).sendKeys("Sunita TCS");

*driver*.switchTo().defaultContent();-----Driver will exit from current frame and it will back to main page.

We cannot jump from one from to another, we have to exit from current frame.

View frame source----Web elements are available inside a frame.

Parallel Testing:

Q. How to implement parallel testing?

Parallel testing in Selenium is **a process where you run the same tests simultaneously in different environments**. The primary purpose of executing tests in parallel is to reduce the overall time and efforts of automated browser testing.

1.Parallel Class: We can execute Multiple TestNG classes parallely.

Generate TestNG.Xml

Parallel mode=classes

Tread count= 3 or 4--🡪Means if TestNG.xml contain 10 TestNG classes, first 4 TestNG classes execute parallely and next 4 TestNG classes and next remaining 2 TestNG classes execute.

Tread count is how many TestNG classes execute parallely.

TestNG.xml contain multiple TestNG classes in single test.

We have to remove thread-count=*"4"* parallel=*"classes" in test*

<suite thread-count=*"4"* parallel=*"classes"* name=*"Suite"*>

<test name=*"Test"*>

<parameter name=*"Brwoser"* value=*"chrome"*/>

<classes>

<class name=*"Day\_016\_ParallelClasses.TC01\_Add\_Locations"*/>

<class name=*"Day\_016\_ParallelClasses.TC02\_Add\_Jobs"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

2. Parallel Test:

Generate TestNG.Xml

Parallel mode=Tests

Tread count= 3 or 4

TestNG.xml contain multiple tests, we have to give different test name (Test1, Test2…)

We can give different packages also.

<suite thread-count=*"3"* parallel=*"tests"* name=*"Suite"*>

<test name=*"Test1"*>

<parameter name=*"Brwoser"* value=*"chrome"*/>

<classes>

<class name=*"Day\_015\_ParallelTests.OrangeHRMTest"*/>

</classes>

</test> <!-- Test -->

<test name=*"Test2"*>

<parameter name=*"Brwoser"* value=*"FireFox"*/>

<classes>

<class name=*"Day\_015\_ParallelTests.MercuryTest"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

3.Parallel Method:

Generate TestNG.Xml

Parallel mode=Methods

Tread count= 3 or 4

If a class contain multiple methods we can execute parallely by generating TestNG.xml using parallel method.

<suite thread-count=*"4"* parallel=*"methods"* name=*"Suite"*>

<test name=*"Test"*>

<classes>

<class name=*"Day\_014\_ParallelMethods.AllTests"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

Multiple Suits:

Q. How to integrate multiple TestNg.xml?

<suite name=*"allSuites"*>

<suite-files>

<suite-file path=*"./suite1.xml"* />

<suite-file path=*"./suite2.xml"* />

</suite-files>

</suite>

Suit1.xml contain multiple TestNG classes

Suit2.xml contain multiple TestNG classes

. ----🡪 It represents current package name.

Advanced xpath:

1. Absolute xpath-->Copy Full xapth--identifying the xpath from root node to current node is nothing but full xpath. If developer made any changes on one xpath, it will impact on all other web elements. Thatsy absolute xpath is not recommended.

When there is no solution at all and we are not able to locate the web element by using any other relative xpath or other locators, then we will use absolute xpath.

2. Relative xpath-->Copy xpath---> identifying the xpath based on object properties is known as relative xpath.

Copy element

<input name="txtUsername" id="txtUsername" type="text">

//input[@name='txtUsername']

(Tag name) (Object properties)

//input[@name=‘txtUsername’ and @id='txtUsername' and @type='text']

//\*[@name='txtUsername']

For Anchor tags (Hyperlinks): If there is no object properties

<a href="/index.php/admin/viewAdminModule" id="menu\_admin\_viewAdminModule"class="firstLevelMenu"><b>Admin</b></a>

//a[@id="menu\_admin\_viewAdminModule"]/b

//\*[text()='Admin']

//b[text()='Admin']

Contains: If label name changing

//\*[contains(text(),'Nationalit')]

//\*[contains(text(),'Welcome')] (Welcome Ravi)

Q. How do handle dynamic web elements?

A: By using partial link test and contains test().

If id (object property) is changing every time when you login.

//\*[contains(@id,'txtUSer')]---Object attribute is changing dynamically everytime (id=txtusername1234)

Starts-with:

//\*[starts-with(text(),'Inbox')]-----It locates Inbox (123)

My Inbox(123)

Inbox(123)

//\*[contains(text(),'Inbox')]--------It locates both My Inbox(123) and Inbox(123)

Following:

<input name="txtUsername" id="txtUsername" type="text">

//\*[@id='txtUsername']------> Current webelement xpath

//\*[@id='txtUsername']//following::input --1 of 2

//\*[@id='txtUsername']//following::input[1]---Password

//\*[@id='txtUsername']//following::input[2]--->loginbutton

How many anchor tags are following:

//\*[@id='txtUsername']//following::a ---1 of 6

Preceding:

//\*[@id="btnLogin"]------> Current webelement xpath

//\*[@id="btnLogin"]//preceding::input --> 1 of 7

//\*[@id="btnLogin"]//preceding::input[2] ----Username

//\*[@id="btnLogin"]//preceding::input[1] ----Password

Parent :( Descendant)

//\*[@id="divUsername"] -----Path of parent web element

It contains 2 child web elements (input, span)

//\*[@id="divUsername"]//child::input

//\*[@id="divUsername"]//child::input[1]

//\*[@id="divUsername"]//child::span[1]

//\*[@id='divUsername']//descendant::input[1]

//\*[@id="divUsername"]//descendant::span[1]

Child:

//\*[@id="txtUsername"] -----Path of child web element

//\*[@id="txtUsername"]//parent::div

Ancestor: Parent+ Grand Parents

//\*[@id='txtUsername']//ancestor::div

//\*[@id='txtUsername']//ancestor::a

//\*[@id='txtUsername']//ancestor::label

Following siblings:

Finding the web elements xpaths of other web elements which are on the same level of Hierarchy

//\*[@id='divUsername']//following-sibling::div

Failed Test cases:

If I want to execute no. of test cases parallely, if one of the test case failed then we will use this method for executing only that failed test case.

First we have to generate testNG.xml and we have to give suit name (name=suit123). After executing testNG.xml. To see the which test case is failed go to

Test output----suit123----test.html

To see at which line it is failed, we can see in emailable report.

After fixing error, we can execute only failed test case

Test output-----suit123----testing-failed.xml

So, we execute teting-failed.xml (only failed test case will be executed).

TestNG Groups:

* We don’t want to define test methods separately in different classes (depending upon functionality) and
* At the same time want to ignore (not to execute) some test cases as if they does not exist in the code.
* So to carry out this we have to Group them. This is done by using “include” and “exclude” mechanism supported in testNG.

TestNG is a testing framework that covers different types of test designs like unit, functional, end to end, UI and integration test.

We can execute specific group of test cases. First we have to create testing.xml.

testing.xml:

<suite name=*"Suite2"*>

<test name = *"test2"*>

<groups>

<run>

<include name=*"checkintest"* />

<exclude name=*"functest"*/>

<exclude name=*"GoogleGroup"*/>

</run>

</groups>

<classes>

<class name = *"Day\_032\_TestNG\_Groups.AllTests"* />

</classes>

</test>

</suite> <!-- Suite -->

And we will mention that group name in above method in testNG class.

@Test(groups = { "checkintest" })

TestNG Annotations:

TestNG Annotations are used to control the next method to be executed in the test script. TestNG annotations are defined before every method in the test code. In case any method is not prefixed with annotations, it will be ignored and not be executed as part of the test code. To define them, methods need to be simply annotated with ‘@Test‘.

* @BeforeSuite
* @BeforeTest
* @BeforeClass
* @BeforeMethod
* @Test
* @AfterMethod
* @AfterClass
* @AfterTest
* @AfterSuite
* **@BeforeMethod**: This will be executed before every @test annotated method.
* **@AfterMethod:** This will be executed after every @test annotated method.
* **@BeforeClass:** This will be executed before first @Test method execution. It will be executed one only time throughout the test case.
* **@AfterClass:** This will be executed after all test methods in the current class have been run
* **@BeforeTest:** This will be executed before the first @Test annotated method. It can be executed multiple times before the test case.
* **@AfterTest**: A method with this annotation will be executed when all @Test annotated methods complete the execution of those classes inside the <test> tag in the TestNG.xml file.
* **@BeforeSuite**: It will run only once, before all tests in the suite are executed.
* **@AfterSuite:** A method with this annotation will run once after the execution of all tests in the suite is complete.
* **@BeforeGroups**: This method will run before the first test run of that specific group.
* **@AfterGroups**: This method will run after all test methods of that group complete their execution.

If a class contain multiple @Test (Main tests) then @AfterTest and @BeforeTest will executed for every @Test.

TestNG Listener:

Listener is defined as interface that modifies the default TestNG’s behavior. As the name suggests Listeners “listen” to the event defined in the selenium script and behave accordingly. It is used in selenium by implementing Listeners Interface. It allows customizing TestNG reports or logs. There are many types of TestNG listeners available.

1. IAnnotationTransformer ,
2. IAnnotationTransformer2 ,
3. IConfigurable ,
4. IConfigurationListener ,
5. IExecutionListener,
6. IHookable ,
7. IInvokedMethodListener ,
8. IInvokedMethodListener2 ,
9. IMethodInterceptor ,
10. IReporter,
11. ISuiteListener,
12. ITestListener .(interface class)

It is used to generate custom report log statements or customized messages in emailabe report. Every test execution progress is monitored in background by listnertest.java. Listnertest.java contain predefined methods.

ITestListener has following methods

* **OnStart-** OnStart method is called when any Test starts.
* **onTestSuccess-** onTestSuccess method is called on the success of any Test.
* **onTestFailure-** onTestFailure method is called on the failure of any Test.
* **onTestSkipped-**onTestSkippedmethod is called on skipped of any Test.
* **onTestFailedButWithinSuccessPercentage-**method is called each time Test fails but is within success percentage.
* **onFinish-**onFinish method is called after all Tests are executed.

**Testlistner.java:**

**public** **class** ListenerTest **implements** ITestListener (Add unimplemented methods)

{

@Override

**public** **void** onFinish(ITestContext Result)

{

}

@Override

**public** **void** onStart(ITestContext Result)

{

}

@Override

**public** **void** onTestFailedButWithinSuccessPercentage(ITestResult Result)

{

}

// When Test case get failed, this method is called.

@Override

**public** **void** onTestFailure(ITestResult Result)

{

System.***out***.println("The name of the testcase failed is :"+Result.getName());

Reporter.*log*("The name of the testcase failed is :"+Result.getName());

}

// When Test case get Skipped, this method is called.

@Override

**public** **void** onTestSkipped(ITestResult Result)

{

System.***out***.println("The name of the testcase Skipped is :"+Result.getName());

}

// When Test case get Started, this method is called.

@Override

**public** **void** onTestStart(ITestResult Result)

{

System.***out***.println(Result.getName()+" test case started");

}

// When Test case get passed, this method is called.

@Override

**public** **void** onTestSuccess(ITestResult Result)

{

System.***out***.println("The name of the testcase passed is :"+Result.getName());

Reporter.*log*("The name of the testcase Passed is :"+Result.getName());

}

}

testNG.xml:

<suite name=*"Suite"*>

<listeners>

<listener class-name=*"Day\_034\_TestNG\_Listeners.ListenerTest"*/>

</listeners>

<test name=*"Batch-Listeners"*>

<classes>

<class name=*"Day\_034\_TestNG\_Listeners.TestCases"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

Hybrid Frame Work:

Packages:

1. Common Util
2. Excel Util
3. Extent Reports
4. Pages(Separate java classes for each web page like login page, Home page, etc…
5. Test Cases

Models we used in Hybrid frame work are:

* Page Object Model
* Extent report/Emailable report/reporter-Word
* TestNG Frame work
* Data Driven Method

We have to know following points for hybrid frame work.

How to launch the browser?

How the test data is managing?

How the reporter is managing?

Cucumber Frame Work:

<https://www.axelerant.com/blog/setup-for-selenium-with-cucumber-using-maven>

3 components of cucumber frame work are:

1. Feature file
2. Step\_Defination.java
3. Testrunner.java
4. Feature file: (src/test/resources)

Feature file may contain multiple scenarios.

We will write scenarios using keywords like Given, When, Then, And, But.

Feature: Orange HRM Application - Create Nationality Record

Scenario: TC01\_ Create Nationality Record

When I open OpenOrangeHRM website

Given Sign in Details

And I created Nationality Record

Then I sign out

1. Step\_Defination.java:(src/test/java)

seleniumgluecode (package)--🡪TestNG Class (Test.java)

**public** **class** Test1 {

**public** WebDriver driver;

@When("^I open OpenOrangeHRM website$")

**public** **void** i\_open\_automationpractice\_website() **throws** Throwable {

System.*setProperty*("webdriver.chrome.driver","C:\\chromedriver\_win32\\chromedriver.exe");

driver =**new** ChromeDriver();

driver.manage().window().maximize() ;

driver.manage().deleteAllCookies();

driver.get("https://opensource-demo.orangehrmlive.com/");

}

1. Testrunner.java:(src/test/java)

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = {"classpath:features/Nationality.feature","classpath:features/EmployeeDetails.feature"}

,glue= {"seleniumgluecode2"},

//plugin = {"pretty", "html:target/cucumber-html-report"},

plugin = { "com.cucumber.listener.ExtentCucumberFormatter:target/cucumber-reports/report.html"},

tags = {}

)

**public** **class** testrunner {

}

Cucumber\_Scenario\_Outline:

We can use Data driven test in cucumber frame work. We can execute a test into multiple times with different sets of input data by using parameter.

Feature file:

Feature: Title of your feature

I want to use this template for my feature file

@tag2

Scenario Outline: TC01\_ Create Nationality Record

Given I open Orange HRM Website

When I enter valid "Admin" and valid "admin123"

Then I created Nationality Record "<NationalityRecord>"

And I sign out

Examples:

| NationalityRecord |

| Indian208 |

| Indian209 |

| Indian210 |

Step definition:

@When("^I enter valid \"(.\*?)\" and valid \"(.\*?)\"$")

**public** **void** i\_sign\_in(String UserName,String Password) **throws** Throwable {

findElement(By.*name*("txtUsername")).sendKeys(UserName);

findElement(By.*name*("txtPassword")).sendKeys(Password);

findElement(By.*id*("btnLogin")).click();

}

@Then("^I created Nationality Record \"(.\*?)\"$")

**public** **void** I\_created\_Nationality\_Record(String NationalityName1) **throws** Throwable {

driver.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

findElement(By.*id*("menu\_admin\_viewAdminModule")).click();

findElement(By.*id*("menu\_admin\_nationality")).click();

findElement(By.*id*("btnAdd")).click();

findElement(By.*id*("nationality\_name")).sendKeys(NationalityName1);

findElement(By.*id*("btnSave")).click();

}

Testrunner.java:

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = "classpath:features"----🡪it runs all feature files

,glue= {"step\_definitions"},

plugin = {"pretty", "html:target/cucumber-html-report"},

tags = {}

)

**public** **class** testrunner {

}

Cucumber Data table:

**import** cucumber.api.DataTable;

Feature file:

Feature: Orange HRM Application - Create Nationality Record

Scenario: TC01\_ Create Nationality Record

Given I open OpenOrangeHRM website

When User Navigate to LogIn Page

|Admin|admin123|

And I created Nationality Record

|Indian79|

Then User LogOut from the Application

Step Definition:

@When("^User Navigate to LogIn Page$")

**public** **void** user\_enters\_testuser\_\_and\_Test(DataTable usercredentials) **throws** Throwable {

List<List<String>> data = usercredentials.raw();

findElement(By.*id*("txtUsername")).sendKeys(data.get(0).get(0)); findElement(By.*id*("txtPassword")).sendKeys(data.get(0).get(1));

findElement(By.*id*("btnLogin")).click();

}

@Then("^I created Nationality Record$")

**public** **void** I\_created\_Nationality\_Record(DataTable NationalityData) **throws** Throwable {

List<List<String>> data = NationalityData.raw();

*driver*.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

findElement(By.*id*("menu\_admin\_viewAdminModule")).click();

findElement(By.*id*("menu\_admin\_nationality")).click();

findElement(By.*id*("btnAdd")).click();

findElement(By.*id*("nationality\_name")).sendKeys(data.get(0).get(0));

findElement(By.*id*("btnSave")).click();

}

Testrunner:

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = {"classpath:features/Nationality.feature"},

glue= {"seleniumgluecode"},

plugin = {"pretty", "html:target/cucumber-html-report"},

tags = {}

)

**public** **class** testrunner {

}

Cucumber Data table with Header:

Feature file:

Feature: Orange HRM Application - Create Nationality Record

Scenario: TC01\_ Create Nationality Record

Given I open OpenOrangeHRM website

When User Navigate to LogIn Page

|Username|Password|

|Admin|admin123|

And I created Nationality Record

|Nationality|

|Indian16|

|Indian17|

Then User LogOut from the Application

Step definition:

@When("^User Navigate to LogIn Page$")

**public** **void** user\_enters\_testuser\_\_and\_Test(DataTable usercredentials) **throws** Throwable {

//List<List<String>> data = usercredentials.raw();

List<Map<String,String>> data = usercredentials.asMaps(String.**class**,String.**class**);

findElement(By.*id*("txtUsername")).sendKeys(data.get(0).get("Username"));

findElement(By.*id*("txtPassword")).sendKeys(data.get(0).get("Password"));

findElement(By.*id*("btnLogin")).click();

}

@Then("^I created Nationality Record$")

**public** **void** I\_created\_Nationality\_Record(DataTable Nationality1) **throws** Throwable {

**for** (Map<String, String> data : Nationality1.asMaps(String.**class**,String.**class**))

{

//List<Map<String,String>> data = usercredentials.asMaps(String.class,String.class);

//System.out.println("Nation1 text is :"+data.get(0)("Nationality"));

*driver*.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

findElement(By.*id*("menu\_admin\_viewAdminModule")).click();

findElement(By.*id*("menu\_admin\_nationality")).click();

findElement(By.*id*("btnAdd")).click();

findElement(By.*id*("nationality\_name")).sendKeys(data.get("Nationality"));

findElement(By.*id*("btnSave")).click();

Thread.*sleep*(3000);

}

}

Testrunner:

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = {"classpath:features/Nationality.feature"},

glue= {"seleniumgluecode"},

//plugin = {"pretty", "html:target/cucumber-html-report"},

plugin = { "com.cucumber.listener.ExtentCucumberFormatter:target/cucumber-reports/report.html"},

tags = {}

)

**public** **class** testrunner {

}

Q. How we can achieve parameterization?

A: By using scenario outline and data table concept.

Cucumber tags:

<https://www.toolsqa.com/cucumber/cucumber-tags/>

We can write no. of scenarios in feature file. Every scenario belongs to specific groups (Like smoke test, regression test…).

Cucumber tags concept is used for executing specific group of test cases.

Feature file:

@FunctionalTest

Feature: ECommerce Application

@SmokeTest @RegressionTest

Scenario: Successful Login

Given This is a blank test

@RegressionTest

Scenario: UnSuccessful Login

Given This is a blank test

@SmokeTest

Scenario: Add a product to bag

Given This is a blank test

Scenario: Add multiple product to bag

Given This is a blank test

@SmokeTest @RegressionTest

Scenario: Remove a product from bag

Given This is a blank test

@RegressionTest

Scenario: Remove all products from bag

Given This is a blank test

@SmokeTest

Scenario: Increase product quantity from bag page

Given This is a blank test

Scenario: Decrease product quantity from bag page

Given This is a blank test

@SmokeTest @End2End

Scenario: Buy a product with cash payment

Given This is a blank test

@SmokeTest @End2End

Scenario: Buy a product with CC payment

Given This is a blank test

@End2End

Scenario Outline: Payment declined

Given This is a blank test

Examples:

|PaymentMethod|

|CC Card|

|DD Card|

|Bank Transfer|

|PayPal|

|Cash|

* *Few scenarios are part of the Smoke Test, Regression Test, and End2End Test.*
* *Few scenarios are part of two or more Test Types. For example, the first test is considered as Smoke as well as Regression*.
* *Few scenarios are not at all tagged*
* *The last scenario of Payment Declined, it is a single scenario but has five different test data. So this will be considered as five different scenarios*.
* Special Character ~ (tilde) is used to skip the tags.
* tags = {"@SmokeTest, @ RegressionTest "} OR means scenarios that are tagged either as @SmokeTest OR @RegressionTest.
* tags = {"@SmokeTest”, “@ RegressionTest "} There are only two scenarios in our feature file which have both tags together.
* tags = {"@FunctonalTest”, “~@ SmokeTest "} This is AND condition, which means all the scenarios tagged as @FunctionalTest but not @SmokeTest. So total tests are 15 and smoke tests are 6, so it ran just 9 tests.
* tags = {"@FunctonalTest"} All the test exists in the feature file are executed.

Testrunner:

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = "classpath:features"

,glue= {"step\_definitions"},

plugin = {"pretty", "html:target/cucumber-html-report"},

tags = {"@SmokeTest"}

//Running single Cucumber Feature file or single Cucumber Tag

//Execute all tests tagged as @SmokeTests

//tags = {"@End2End"}

//Execute all tests tagged as @End2End

)

**public** **class** testrunner {

}

Cucumber Hooks:

To delete cookies:

driver.manage().deleteAllCokkies();

Cucumber supports ***hooks***, which are blocks of code that run ***before*** or ***after*** each scenario

In the world of testing, you must have encountered the situations where you need to perform the prerequisite steps before testing any test scenario. This prerequisite can be anything from:

* *Starting a webdriver*
* *Setting up DB connections*
* *Setting up test data*
* *Setting up browser cookies*
* *Navigating to certain page*
* *or anything before the test*

In the same way, there are always after steps as well of the tests like:

* *Killing the webdriver*
* *Closing DB connections*
* *Clearing the test data*
* *Clearing browser cookies*
* *Logging out from the application*
* *Printing reports or logs*
* *Taking screenshots on error*
* or anything after the test

Feature file:

Feature: Test Hooks

Scenario: This scenario is to test hooks functionality

Given this is the first step

When this is the second step

Then this is the third step

Step definition:

**public** **class** Hooks\_Steps {

@Given("^this is the first step$")

**public** **void** This\_Is\_The\_First\_Step(){

System.***out***.println("This is the first step");

}

@When("^this is the second step$")

**public** **void** This\_Is\_The\_Second\_Step(){

System.***out***.println("This is the second step");

}

@Then("^this is the third step$")

**public** **void** This\_Is\_The\_Third\_Step(){

System.***out***.println("This is the third step");

}

}

Hooks.java:

**public** **class** Hooks {

@Before

**public** **void** beforeScenario(){

System.***out***.println("This will run before the Scenario");

}

@After

**public** **void** afterScenario(){

System.***out***.println("This will run after the Scenario");

}

}

Testrunner:

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = "classpath:features"

,glue= {"step\_definitions"},

plugin = {"pretty", "html:target/cucumber-html-report"},

tags = {}

)

**public** **class** testrunner {

}

Cucumber project (Using page Object model):

Background: Background will execute before every scenario. It is common re-usable steps for every scenario.

Feature file1:

Feature: Post feature of facebook

This will test the post functionality at the user wall

Background: Common step

Given User launch chrome browser

When User opens URL "https://opensource-demo.orangehrmlive.com/index.php/auth/login"

Scenario: Successful Login with valid page

And User enters username as "Admin" and password as "admin123"

And Click on login button

Then Page URL Should be "https://opensource-demo.orangehrmlive.com/index.php/dashboard"

When Click on logout button

Then Close Browser

Scenario: Login with Invalid Credentials

And User enters username as "Admin" and password as "admin"

And Click on login button

Then Error message should displayed "Invalid credentials"

Then Close Browser

Feature file2:

Feature: Addskills

Scenario Outline: Add skills with description

Given User launch chrome browser

When User opens URL "https://opensource-demo.orangehrmlive.com/index.php/auth/login"

And User enters username as "Admin" and password as "admin123"

And Click on login button

Then Page URL Should be "https://opensource-demo.orangehrmlive.com/index.php/dashboard"

Then Go to Skill Page

When Add skills with "<skillname>" and "<skilldesc>"

Then Click on Save button

Then Click on logout button

Then Close Browser

Examples:

| skillname | skilldesc |

| Appium | Mobile Automation Testing |

| Java | Core Java |

| Selenium | web Automation Testing |

Scenario: Delete Skill record

Given User launch chrome browser

When User opens URL "https://opensource-demo.orangehrmlive.com/index.php/auth/login"

And User enters username as "Admin" and password as "admin123"

And Click on login button

Then Page URL Should be "https://opensource-demo.orangehrmlive.com/index.php/dashboard"

Then Go to Skill Page

Then Select "selenium" record to delete

Then Click on Delete button to delete selected record

TestRunner:

@RunWith(Cucumber.**class**)

@CucumberOptions(

features="D://CucumberProject1//Features//Login.feature",

//features = "classpath:features",

glue="stepDefinitions",

strict=**true**,// it will check if any step is not defined in Step Definitions file

plugin = {"pretty", "html:target/cucumber-html-report"},

tags = {}

)

**public** **class** TestRun{

}

Jenkins:

It is continuous integration software.

The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project. Jenkins is used to build and test your product continuously, so developers can continuously integrate changes into the build.

Jenkins supports jdk (8, 11) versions, it won’t support jdk (12)

TestNG.xml 🡨---- prom.xml 🡨-----Jenkins

Prom.xml:

In prom.xml, under plugin section i will conflict testNG.xml.

Mavan.compiler-plugin(3.8.0)

Mavan.surefire.plugin(3.0.0)

We can execute test by prom.xml

Prom.xml(Rigt click)---🡪 Run as( maven install)

Project(Right click)--🡪 maven-🡪update

Download Jenkins(Generic java package(.war))

Command promt---cd jenkin path----dir(to se Jenkins.war file)

Java –jar Jenkins.war ( to install Jenkins)

Copy Jenkins path and go to c drive Jenkins---copy Jenkins path---secrets---intial Admin password-----Copy jenkins password

Chrome----http://localhost:8080---signin

Manage jenkins----Build plugins----install---maven plugins And testNG plugins

Command promt---cd jenkin path----dir(to se Jenkins.war file)

Java –jar Jenkins.war ( to install Jenkins)

New item---OrangeHRM----Maven project----ok

OrangHRM---configure----Description----Build-----Root pom (prom.xml Loaction)----Goals @ Options( install)----Post build actions(public TestNG results)---Add maven---Maven----save

OrangHRM----Build now----console output

FAQS:

1. What are different locators in selenium?

2. What is the difference between driver.findElement Vs. driver.findElements?

A. driver.findElement means locating single web element.

driver.findElement(By.*name*("txtUsername")).sendKeys(UserName);

driver.findElements means locating group of web elements.

List<WebElement> links = driver.findElements(By.*tagName*("a"));

System.***out***.println ("Total links are:"+links.size ());

int count= links.size();

3. How do find no. of hyperlinks in a web page?

4. How do find no. of iFrames in a web page?

5. How to get test data from TestNG dataprovider?

6. How to get test data from TEstNG.xml parameter?

7. How to get test data from Excel file (.xlsx) using Apache POI?

8. How to get test data from Excel file (.xls) using Apache POI?

9. How to switch or handle driver among the frames?

10. How to switch or handle driver among the multiple windows?

11. How to handle Alerts?

12. Implicit wait?

13. Explicit wait?

14. Fluent wait?

15. List Box?

16. What is the difference between driver.quit() and driver.close()?

17. What is the difference between driver.get() and driver.navigate()?

A. driver.get();----driver will wait until all the web elements are loading in the page, can’t move to next instruction?

driver.navigate(); ----driver will wait immediate navigation and will be ready to execute next instruction.

18. How to perform browser forward,back and refresh?

A. driver.navigate().forward();

driver.navigate().back();

driver.navigate().refresh();

19. Explain about isDisplayed(), isEnabled(), isSelected()?

A.To verify a web element is displayed or not

Boolean str= driver.findElement(By.xpath(“”)).isDisplayed();

To verify button is enabled or not

Boolean str1= driver.findElement(By.xpath(“”)).isEnabled();

To verify check box is selected or not

Boolean str2= driver.findElement(By.xpath(“”)).isSelected();

20. How to do scroll?

21. How to handle web element using JavaScript?

22. Action classes in selenium?

23. How to get text from web element?

A. String vaueIneed= driver.findElement(By.xpath(“”).getText();

24. WebTable Example?

25. How to get screen shot in selenium?

**import** org.openqa.selenium.TakesScreenshot;

**import** org.openqa.selenium.OutputType;

A.Convert webdriver object to TakeScreenshot

TakeScreenshot scrshot=((TakeScreenshot)driver);

Call getScreenshotAs method to creste image file

File ScrFile=ScrShot.getScreenshotAs(OutputType.FILE);

Move image file to new destination

File DestFile=new File(fileWithPath);

Copy file at destination

FileUtiles.copyFile(SrcFile, DestFile);