AUTOMATION FRAMEWORK: WALKTHROUGH

INTRODUCTION

Automation is the process of following a predetermined sequence of operations with little or no human labor, using specialized tools that perform and control most of the processes. It eases by performing most of the laborious and repetitive manual tasks with an initial investment of efforts, which indeed is worth it.
For automating web applications, we can make use of **Selenium** as it is open source and not platform specific.

PRE-REQUISITES

- Operating System : Windows/Mac/Linux/Solaris.
- IDE : Eclipse, NetBeans, Katalon Studio etc. (Note:Here we will use Eclipse IDE).
- Browser: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Opera
- Programming Language : Java (Java must be installed in the system)Download Link for Windows
- Visit Selenium's Official Website and download Selenium WebDriver for the browser that you will be making use of.
- Ensure that your browser driver is executable.(Go to project folder>Drivers>Right click on driver>Permissions>Check the check box:Allow executing file as a program.)Refer ScreenClip

PROJECT STRUCTURE

File Structure:
- Drivers
    - chromedriver
    - geckodriver
- logs
    - LogsOfExecuteLeadTest(To find logs of ExecuteLeadTest.java)
    - LogsOfExecuteLeadTestSingleTestCaseRunner(To find logs of ExecuteLeadTestSingleTestCaseRunner.java)

- Reports
    - ReportsOfExecuteLeadTest(To find reports of ExecuteLeadTest.java)
    - ReportsOfExecuteLeadTestSingleTestCaseRunner(To find reports of ExecuteLeadTestSingleTestCaseRunner.java)
    - ReportsOfPaginateUntilElementFound(To find reports of PaginateUntilElementFound.java)

- Repositories
    - Objectrepository.properties

- TestCases
    - TestCases.xlsx

CONCEPT

In this framework, a lot of tasks have been eased for the end users' comfort and developed in a way to reduce further coding efforts.
Firstly, there will be a POM(Page Object Model) file which will be added to the project so that the user need not worry about downloading the Jar files manually and add it to the project time and again. With the help of this POM file, all the required jars for this project will automatically be downloaded and if in future, the user wishes to update the jar version,the only change that needs to be incorporated in the POM file will be the version number of the jar files after which the old jars will be automatically removed by replacing it with the the new ones.

Secondly, we will be creating an object repositories properties file in which we will be storing all the Page Objects i.e., the XPaths (XML Path Language/XPath is a query language for selecting nodes from an XML document) so that we need not have a clumsy Java code.

Thirdly, we will be creating a keywords java class where we can have all the required methods i.e., the operations that we perform on the web pages. For instance : click, navigate, refresh, quit etc.

Finally, we will be creating a lead java class where we will call the keywords java file to perform the methods on the website which will be written as a sequence of operations in the excel sheet that we will be integrating with this lead java class.

In sum, what a user must do mostly is to write the steps in the sheet and mention the keyword, required data(if any) and object name if he/she needs to perform an operation.

Example:

Excel File

TestCaseId (Optional)	Description (Optional)	Keyword	Data	ObjectName	Runmode
TC_01	Open the browser	openbrowser	no value	no value	yes
TC_02	Click on the element	click	no value	elexpath	yes
TC_03	Enter data	input	abcd	textboxxpath	yes
TC_04	Close the browser	quit	No value	no value	no

Sheet Name : Sheet-A

ObjectRepositories.properties file
Here,
->We will be storing all the xpaths as object names
Ex:
username=//*[@id="user_username"]
password=//*[@id="user_password"]
update=//input[@id='submit_button']

Note:

Most of the XPaths have been framed in a way that they are dynamic enough to be reused in other places. However there are a few limitations because some elements technically do not have unique attributes of themselves or at times, some elements need to be located with their exact texts' that keeps changing like Student Id, Employee Id, Course Names/Subject Names etc.

Workaround

In order to overcome the aforementioned limitation, the user has to visit the objectrepository.properties file and just change the Student Id/Employee Id/Course Names/Subject Names etc in the dynamic Xpaths or store it with unique object names for every such element which is still a very good workaround. For example:

ClickOnStudent=//td[position()='1' and text()='275']//parent::tr[@class='tr-body']//a
Here, the user has to replace '275' with some other student's ID.
Or
ClickOnStudent275=//td[position()='1' and text()='275']//parent::tr[@class='tr-body']//a
These can be called Customized Xpaths.

Keywords.java file

Here,
->We will be reading the objectrepositories.properties file
->Create methods like
click: to click on an element
navigate: to navigate to a page
Refresh: to refresh the page
->**Set the path of the selenium webdriver for the respective browser you need to use.**
Syntax:
System.setProperty("webdriver.chrome.driver", "Path of the driver");

Ex:
System.setProperty("webdriver.chrome.driver", "/home/systemname/chromedriver");
Once this is done, the chrome browser will be ready for use.

The Path will be taken dynamically for the user's convenience by using current project directory.

ExecuteLeadTestSingleTestCaseRunner.java

Here,
->We will be calling the keywords.Java file
->We will be giving the path of the excel file in

FileInputStream file = new FileInputStream("filepath_of_ExcelFile_filename.xlsx");
The Path will be taken dynamically for the user's convenience by using current project directory.
So as to read the file

->Entering the sheet name in

XSSFSheet sheet = workbook.getSheet("Sheet-A");
So as to read the particular sheet in that excel file

Key Points

Thread.Sleep(milliseconds);
which will be given in the sheet as sleep in the keyword, amount of time in Data and no value in ObjectName must be given as per the user's internet speed. An explicit wait could have been implemented which will wait only for visibility of the element but it will increase the code and make it a little ambiguous for the user as he/she will have to specify time and again in the repositories, the xpath of that object. Workaround: To overcome that sleep has been

implemented which will simply wait for the amount of seconds as it will wait for a certain period of time until the next step's execution.

Ex:

Excel File

TestCaseId (Optional)	Description (Optional)	Keyword	Data	ObjectName	Runmode
TC_01	Sleep for some seconds	sleep	3000	no value	yes

The above step will make the code wait for 3000 milliseconds i.e., 3 seconds.

Select Box
selectbyvisibletextint, selectbyvalue, selectbyindex, selectbyvisibletextstring

selectbyvalue: selects the element by it's value.
Example: If there is a select box with values aone, ab2, ab3
Excel File

TestCaseId (Optional)	Description (Optional)	Keyword	Data	ObjectName	Runmode
TC_01	Select ab3	selectbyvalue	ab3	xpathofselectbox	yes

The above step will select ab3.

selectbyindex: selects the element by it's index.
Example: If there is a select box with values aone, ab2, ab3
Excel File

TestCaseId (Optional)	Description (Optional)	Keyword	Data	ObjectName	Runmode
TC_01	Select ab3	selectbyindex	2	xpathofselectbox	yes

The above step will select ab3.

selectbyvisibletextint: selects the element if the option is an integer value.

Example: If there is a select box with values 11,23, 245

Excel File

TestCaseId (Optional)	Description (Optional)	Keyword	Data	ObjectName	Runmode
TC_01	Select ab3	selectbyvisibletextint	23	xpathofselectbox	yes

The above step will select 23.

selectbyvisibletextstring: selects the element if it is string.

Example: If there is a select box with values apple, orange, mango

Excel File

TestCaseId (Optional)	Description (Optional)	Keyword	Data	ObjectName	Runmode
TC_01	Select ab3	selectbyvisibletextstring	mango	xpathofselectbox	yes

The above step will select mango.

The same goes for **inputint** when you have to enter integer values and **input** for other data input

ALTERNATIVE METHOD

=CONCATENATE() function in the excel sheet.

If you have to enter a phone number starting with +91 or want to selectbyvisibletextstring regardless of the data type, then you will have to use =CONCATENATE() function.

SYNTAX

=CONCATENATE(“First data”,”Second date”,”etc”) and press enter or save. Clicking on some other cell might concatenate the other selected cells.

EXAMPLE

=CONCATENATE(“for-ele”,” -31”,”any”)
Result of the cell=for-ele -31any

Location of csv file to be added in places where upload csv is needed
If it is Ubuntu : Just Copy the file and paste it in a notepad> Now you will get the exact path of the file>Paste this under Data in Excel sheet.
Also, the code will now take only the relative path of the file so you need not give the exact path. However, the sheets are already present in the project and you might just have to cross check the location just in case(for your reference as well).
Example:

Excel File

TestCaseId (Optional)	Description (Optional)	Keyword	Data	ObjectName	Runmode
TC_01	Select a file	selectfile	/home/foradian/TuesdayCurriculum.csv	No value	yes

After downloading the Files
Go to File>Open Projects From File System>Select the FINZY Folder>Click on Finish
Double Click on FINZY Project>src/test/java>com.FINZY.pageObjects

Run the code

Run PaginateUntilElementFound.java
Just Enter the search text/keywords wherever mentioned after the comments

Run multiple test cases(if any) at once

Run ExecuteLeadTest.java
Just add sheet name in this way:

```
String[] test_cases = new String[]{ "Finzy", “nextsheetname”};
```

Package bonus

-MultipleBrowserParallelRunner.java
-MultipleBrowserSequentialRunner.java

To run the tests in parallel across multiple browsers, open **TestNGMultipleBrowserParallelRunner.xml** and run it as a testng suite.
Similarly, to run the tests sequentially across multiple browsers, open **TestNGMultipleBrowserSequentialRunner.xml** and run it as a testng suite.

SEQUENCE and Working to run single test cases

Just Open the Excel File, double click on the sheet name and copy it> Open **ExecuteLeadTestSingleTestCaseRunner.java** and paste it inside the XSSFSheet command. A few examples are shown below:

1)sheetName

```
XSSFSheet sheet = workbook.getSheet("Finzy");
```

Similarly, you can paste the sheet names and run the script of your choice