What is an Automation Framework?

An automation testing framework is a *set of guidelines or rules* used for creating and designing test cases. The guidelines include *coding standards, object repositories, test-data handling methods, processes to store test results,* or any other information on how to access external resources.

A tester can always write tests without a framework, it is not a mandatory step but using an organized framework provides additional benefits like increased *code re-usage, higher portability, reduced script maintenance cost, and higher code readability.* It also helps the team to write down the test script in a *standard format.* Using an automation testing framework, efficient design and development of automated test scripts enable and it ensures reliable analysis of issues or bugs for the system or application under test. The below section lists a few of the important benefits which justifies the need for an automation testing framework:

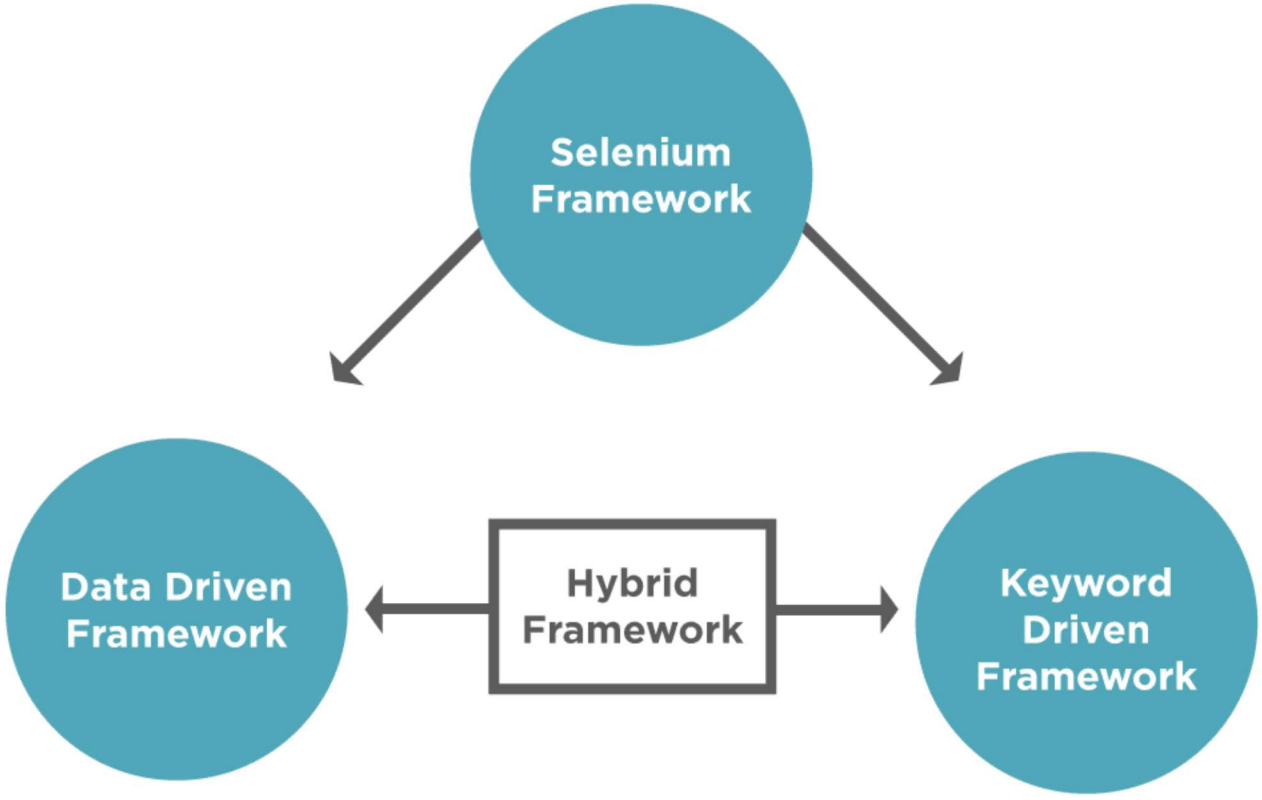
Why do we need an Automation testing framework?

It is important to use a framework for automated testing as it improves the automation testing team's efficiency and test development speed. Some of the benefits of using the automation framework are as below:

Standard Format for all the tests
Improved test efficiency
Lower script maintenance costs
Maximum test coverage
Reusability of code
Efficient Test Data Management

What are the different types of automation frameworks in Selenium?

When testing an application using ***Selenium WebDriver,*** there are three main types of frameworks that we can use to create automated tests for any web application:



Data Driven Test Framework.
Keyword Driven Test Framework.
Hybrid Test Framework.

Each of these frameworks has its own architecture and different benefits and disadvantages. When building out a test plan, it's important to choose the framework that is right for you.

***Data Driven Testing Framework** is used to separate the test script from the test data. You can test the same script with multiple sets of data. We will discuss this framework in detail in the following topics.*
Keyword Driven Testing Framework** is an extension of the Data Driven framework. It allows storing a set of code called 'keywords' in a separate code file, externally from the test script. We can reuse these keywords across multiple test scripts. For details refer -**Keyword Driven Framework
Hybrid Driven Framework** is a combination of both the **Data-Driven** and **Keyword-Driven** framework. Here, the keywords, as well as the test data, are external. We maintain Keywords in a separate file and test data in excel file or CSV files or database. For details refer - **Hybrid Framework.

Here, in this article, Let us take a deep dive into the ***Data Driven Test Framework.***

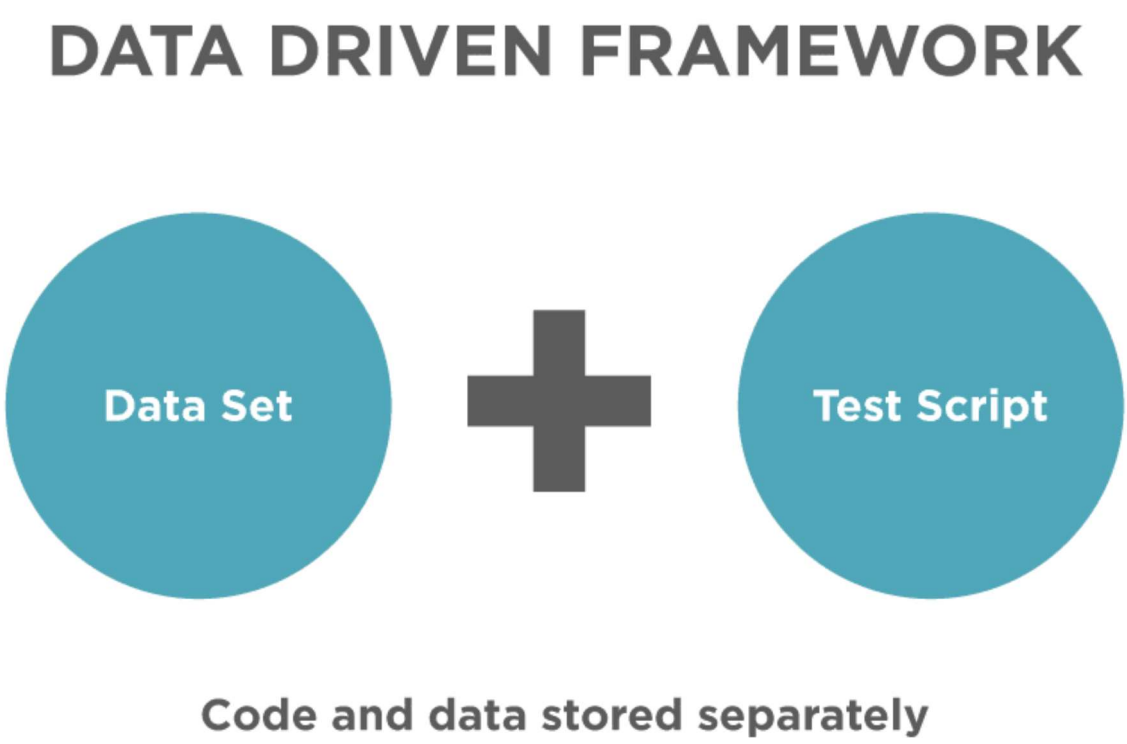
What is Data Driven Framework?

Generally, when we test an application manually, we run the same scenario for multiple test data. Additionally, we keep the same test data in some files like *Excel Files, Text Files, CSV Files, or any database.* The same is true for automation also, where we would like to run the same test scenario

for multiple test data. Let's say you have written an automation script to fill the *student registration form* on the ***ToolsQA Demo site.*** There can be many students for registration, the only thing that is differentiating in the code is input values *(Name, Address, Phone, Gender, etc..).* Will you write a separate script for every student to register? Is there a way, we can reuse the code and only change the student data?

Yes, this is where the *Data Driven Framework* comes into play and makes the test scripts work properly for different sets of test data. It saves time to write additional code. It's like write once and run many times mechanism as you can run the same *Selenium script* multiple times.

In simple words, we use the *Data Driven Framework* when we have to execute the same script with *multiple sets of test data,* whose storage is at a different place and not present inside the test script. Any changes done to the data will not impact the code of the test.



What are the benefits of using the Data Driven Testing Framework?

Following are a few of the major benefits which a QA can reap when he/she develops the automation framework using the *Data-Driven* technique:

Test cases can be modified without much changes to code.

It allows testing the application with multiple sets of data values, especially during regression testing.
It helps us to separate the logic of the test cases/scripts from the test data.

One of the most commonly used, data sources for the test is ***Microsoft Excel Sheets.*** We can maintain the data in excel sheets and use them in the test script. Let's see how we can create a Data Driven UI automation framework by reading the test data from Excel files.

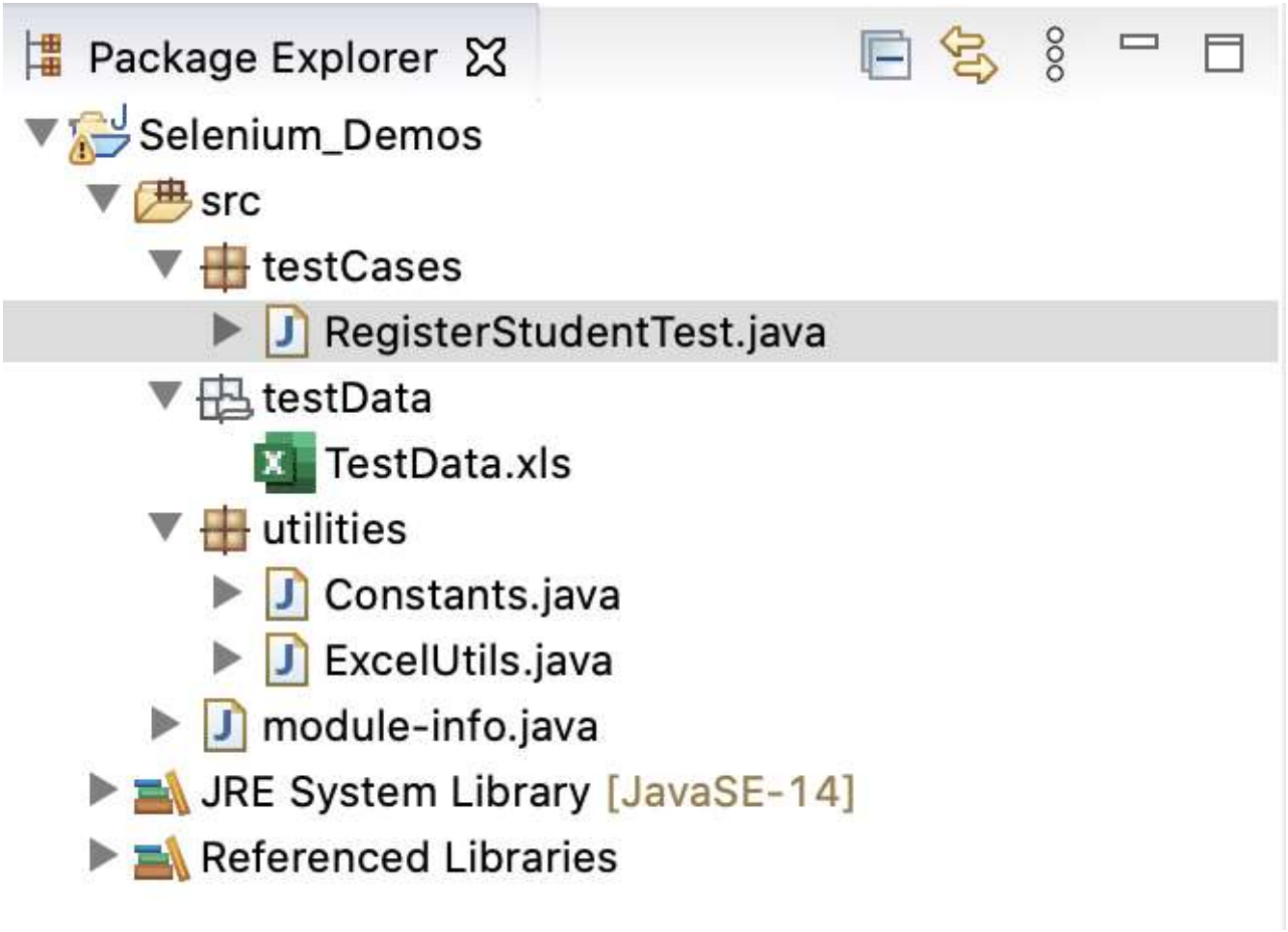
How to create a Data Driven Framework in Selenium using Apache POI?

We have learned in the previous article ***"Read & Write Data from Excel in Selenium"*** how to read and write data in *Excel files* using *Apache POI* and then pass the same data sets as test data to the Selenium tests. But in that script, all the actions of *reading data from an Excel file, writing data to the Excel file, passing the data to the Selenium actions* were happening in the main method of the class. That format is acceptable if we are just writing one or two test cases. However, when we have to develop an automation framework that will have multiple test scenarios, then it should properly organize and should have a defined folder hierarchy.

A basic thumb rule for the data driven testing framework would be to segregate the test data from the test scripts. Also, the actions to read/write the data from files should segregate and be available as utilities.

Follow the steps as mentioned below to create a basic Data Driven framework, which will be used to automate the ***"Student Registration Form".***

*Create three **New Packages** in your Project for testCases, testData, and utilities.*
*Under the **testData** package, put your Excel Sheet that has test data. Using this, we separate the test data from the **testCases.***
*Under the **utilities, create a New Class** and name it **"ExcelUtils"**. It will contain all functions related to Excel used for reading and writing.*
*Under the **utilities** package, create another class **"Constants"**. It will contain the constant values across the framework like testdata file path, URL of the application, etc.*
*Under the **testCases** package, we will create the test files that contain the Selenium code for interacting with web elements. (For Example, RegisterStudentTest.java)*



After performing the above steps, the folder structure will look like:

Let's understand the details of each of these classes:

. ***ExcelUtils Class -*** It is a utility class that will contain all the methods related to Excel Sheet read and write operations along with initializing the Workbook. You can then reuse the methods in different test cases, by creating an object of Excel Utils Class. The code for this class will be as below -

```
package utilities;

import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class ExcelUtils {
   private static HSSFWorkbook workbook;
    private static HSSFSheet sheet;
    private static HSSFRow row;
    private static HSSFCell cell;

   public void setExcelFile(String excelFilePath,String sheetName) throws IOException {
```

```
        //Create an object of File class to open xls file
        File file =    new File(excelFilePath);

        //Create an object of FileInputStream class to read excel file
        FileInputStream inputStream = new FileInputStream(file);

        //creating workbook instance that refers to .xls file
        workbook=new HSSFWorkbook(inputStream);

        //creating a Sheet object
         sheet=workbook.getSheet(sheetName);

    }

     public String getCellData(int rowNumber,int cellNumber){
        //getting the cell value from rowNumber and cell Number
         cell =sheet.getRow(rowNumber).getCell(cellNumber);

         //returning the cell value as string
         return cell.getStringCellValue();
     }

     public int getRowCountInSheet(){
        int rowcount = sheet.getLastRowNum()-sheet.getFirstRowNum();
        return rowcount;
     }

     public void setCellValue(int rowNum,int cellNum,String cellValue,String excelFilePath
         //creating a new cell in row and setting value to it
         sheet.getRow(rowNum).createCell(cellNum).setCellValue(cellValue);

         FileOutputStream outputStream = new FileOutputStream(excelFilePath);
         workbook.write(outputStream);
     }
}
```

The above code contains different methods like ***setExcelFile*** to initialize the *Excel Workbook,* ***getCellValue*** to retrieve the value present in a particular cell in the file, ***setCellValue*** to set some value into a newly created cell. In a similar way, you can create different methods related to excel operations in this class.

. ***Constants Class-*** It is used to put constant values in a file so that the same can be reused across test cases. One more advantage of placing values in separate files is that since these values are common across various tests if there is any change in any of the values, you will just have to update in one place. *For example, if the file path is changed, then instead of updating all the test cases with the new value, you can just update it here in one file.* The structure and values present in this class are as below -

```
package utilities;

public class Constants {
    public static final String URL = "https://demoqa.com/automation-practice-form";
    public static final String Path_TestData = "E:\\Projects\\src\\testData\\";
```

```
    public static final String File_TestData = "TestData.xls";
}
```

. ***RegisterStudentTest-*** It is the test script for the student registration form, which we used to enter the first name, last name, mobile, email, gender, etc for a particular student. Since we have now separated the excel related methods in a separate file, the code of our test case also changes.

We will create an object of ***ExcelUtils*** class in this test file and also use ***Constants*** to refer to the path of the file.

The updated code now looks like **-**

```
package testCases;


import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import utilities.Constants;
import utilities.ExcelUtils;
import java.io.IOException;
import java.util.concurrent.TimeUnit;

public class RegisterStudentTest {

        //creating object of ExcelUtils class
    static ExcelUtils excelUtils = new ExcelUtils();

    //using the Constants class values for excel file path
    static String excelFilePath =Constants.Path_TestData+Constants.File_TestData;

    public static  void main(String args[]) throws IOException {
        //set the Chrome Driver path
        System.setProperty("webdriver.chrome.driver","E:\\Projects\\chromedriver.exe");

        //Creating an object of ChromeDriver
        WebDriver driver = new ChromeDriver();


        //launching the specified URL
        driver.get("https://demoqa.com/automation-practice-form");

        //Identify the WebElements for the student registration form
        WebElement firstName=driver.findElement(By.id("firstName"));
        WebElement lastName=driver.findElement(By.id("lastName"));
        WebElement email=driver.findElement(By.id("userEmail"));
        WebElement genderMale= driver.findElement(By.id("gender-radio-1"));
        WebElement mobile=driver.findElement(By.id("userNumber"));
        WebElement address=driver.findElement(By.id("currentAddress"));
        WebElement submitBtn=driver.findElement(By.id("submit"));

        //calling the ExcelUtils class method to initialise the workbook and sheet
        excelUtils.setExcelFile(excelFilePath,"STUDENT_DATA");
```

```
        //iterate over all the row to print the data present in each cell.
        for(int i=1;i<=excelUtils.getRowCountInSheet();i++)
        {
                //Enter the values read from Excel in firstname,lastname,mobile,email,add
                firstName.sendKeys(excelUtils.getCellData(i,0));
                lastName.sendKeys(excelUtils.getCellData(i,1));
                email.sendKeys(excelUtils.getCellData(i,2));
                mobile.sendKeys(excelUtils.getCellData(i,3));
                address.sendKeys(excelUtils.getCellData(i,4));

                //Click on the gender radio button using javascript
                JavascriptExecutor js = (JavascriptExecutor) driver;
                js.executeScript("arguments[0].click();", genderMale);

                //Click on submit button
                submitBtn.click();

                //Verify the confirmation message
            WebElement confirmationMessage = driver.findElement(By.xpath("//div[text()='T

            //check if confirmation message is displayed
            if (confirmationMessage.isDisplayed()) {
                // if the message is displayed , write PASS in the excel sheet using the
                excelUtils.setCellValue(i,6,"PASS",excelFilePath);
            } else {
                //if the message is not displayed , write FAIL in the excel sheet using t
                excelUtils.setCellValue(i,6,"FAIL",excelFilePath);
            }

            //close the confirmation popup
            WebElement closebtn=driver.findElement(By.id("closeLargeModal"));
            closebtn.click();

            //wait for page to come back to registration page after close button is click
            driver.manage().timeouts().implicitlyWait(2000,TimeUnit.SECONDS);
        }
        //closing the driver
        driver.quit();
    }
}
```

The above class will perform the actions on the student registration page. However, if you notice, methods of the ***ExcelUtils*** handle all the excel related code.

So, this is one of the ways you can use the data driven framework in Selenium. Additionally, you can utilize the advantage of running the same test across multiple sets of data.

Key Takeaways

Data-driven is a test automation framework that stores test data in a table or spread spreadsheet format.

Additionally, a Data-driven Testing framework helps to separate test data from Functional tests. Moreover, it allows testing applications with multiple sets of data values without rewriting the same code across different test scripts.
Also, we perform data-driven Testing in Selenium using Apache POI. It is a library that helps to read and write data in the Excel Sheet.