### ****Task 3:****

****Data Driven framework****

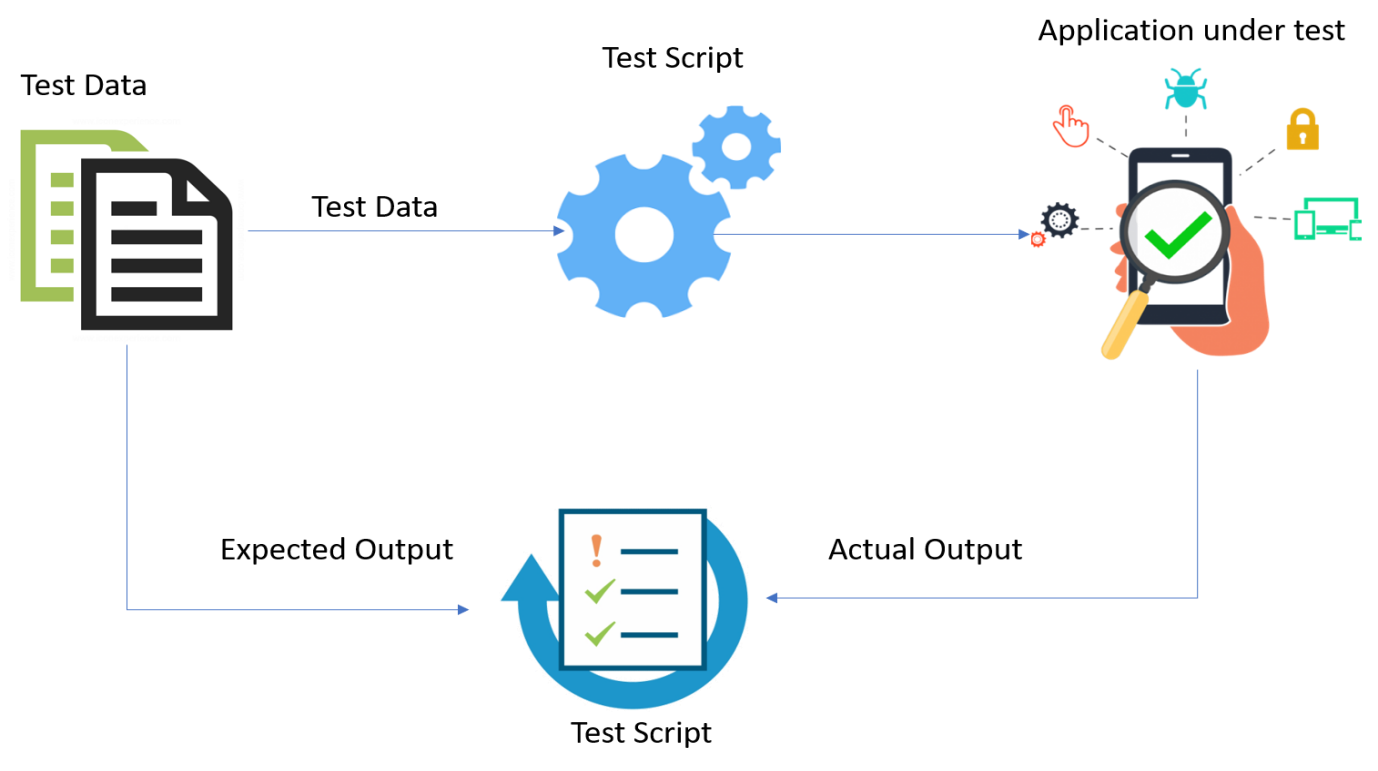
### ****Why Data Driven Tests important?****

There are two main benefits to this:

* They reduce the cost of adding new tests and changing them when your business rules change. This is done by creating parameters for different scenarios, using data sets that the same code can run against.
* They help to identify what data is most important for tested behavior. By separating first-class scenario data into parameters, it becomes clear what matters most to the test. This makes it easy to remember how something works when developers need to change it.

### ****What is Data Driven Testing Framework in Selenium?****

Data Driven framework is used to drive test cases and suites from an external data feed. The data feed can be data sheets like xls, xlsx, and csv files.



A Data Driven Framework in Selenium is a technique of separating the “data set” from the actual “test case” (code). Since the test case is separated from the data set, one can easily modify the test case of a particular functionality without making changes to the code.

For example, if one has to modify the code for login functionality, they can modify just the login functionality instead of having to modify any other feature dependent on the same code.

One can easily increase the number of test parameters by adding more username and password fields to the excel file (or other sources).

Now let’s understand how to create a test case using a Data Driven Framework.

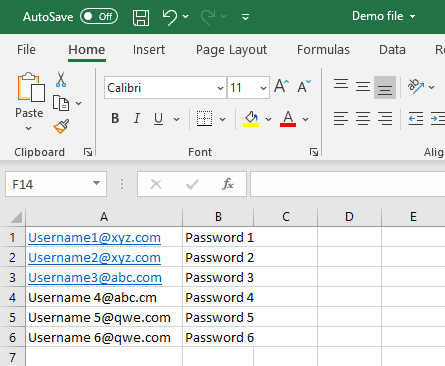
### ****Data Driven Testing Example****

This example will demonstrate how to read the data from excel files and perform data driven testing using [Selenium](https://www.browserstack.com/selenium" \o "What is selenium). WebDriver does not directly support data reading of excel files. Therefore, one needs to use a plugin such as ****Apache POI**** for reading/writing on any Microsoft office document.

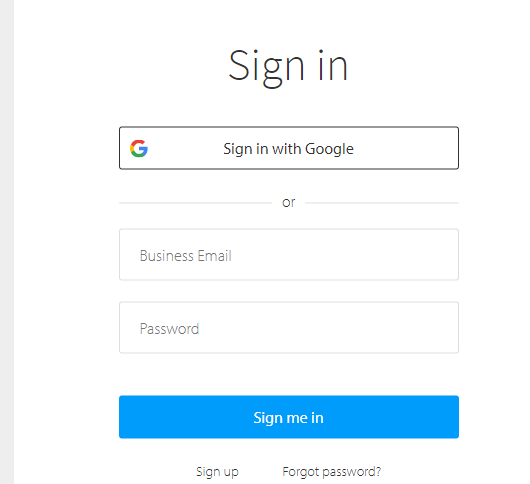
Download the zip file or tar file as per requirement and place them along with the set of Selenium JARs and configure your build path.

Now let’s understand how to write the first test case. An excel file to read the data from the sheet. The user has entered different combinations of username and password in the sheet.

The task here is to enter all the combinations of username and passwords into the login field in order to test the functionality. Let’s see how to do that.



Here, the target is to enter all these combinations of username and password into the Browswertack sign in page as shown below.



Let’s write a code snippet to read the data files.

****Step 1****: Go to the Eclipse IDE and create a project. Add all the dependencies for TestNG, Selenium and Apache POI.

****Step 2****: Create a class file to write the functionality.

Import org.openqa.selenium.By;

import org.testng.Assert;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ExcelExample{

@Test(dataProvider="testdata")

public void demoClass(String username, String password) throws InterruptedException

{

System.setProperty("webdriver.chrome.driver", "Path of Chrome Driver");

Webdriver driver = new ChromeDriver();

driver.get("<a href="https://www.browserstack.com/users/sign\_in</a>");

driver.findElement(By.name("user[login]")).sendKeys(username);

driver.findElement(By.name("user[password]")).sendKeys(password);

driver.findElement(By.name("commit")).click();

Thread.sleep(5000);

Assert.assertTrue(driver.getTitle().matches("BrowserStack Login | Sign Into The Best Mobile & Browser Testing Tool"), "Invalid credentials");

System.out.println("Login successful");

}

@AfterMethod

void ProgramTermination() {

driver.quit();

}

@DataProvider(name="testdata")

public Object[][] testDataExample(){

ReadExcelFile configuration = new ReadExcelFile("Path\_of\_Your\_Excel\_File");

int rows = configuration.getRowCount(0);

Object[][]signin\_credentials = new Object[rows][2];

for(int i=0;i<rows;i++)

{

signin\_credentials[i][0] = config.getData(0, i, 0);

signin\_credentials[i][1] = config.getData(0, i, 1);

}

return signin\_credentials;

}

}

In the above code, there is a “TestDataExample() method” in which the user has created an object instance of another class named “ReadExcelFile”. The user has mentioned the path to the excel file. The user has further defined a for loop to retrieve the text from the excel workbook. But to fetch the data from the excel file, one needs to write a class file for the same.

import java.io.File;

import java.io.FileInputStream;

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

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

public class ReadExcelFile{

XSSFWorkbook work\_book;

XSSFSheet sheet;

public ReadExcelFile(String excelfilePath) {

try {

File s = new File(excelfilePath);FileInputStream stream = new FileInputStream(s);

work\_book = new XSSFWorkbook(stream);

}

catch(Exception e) {

System.out.println(e.getMessage());

}

}

public String getData(int sheetnumber, int row, int column){

sheet = work\_book.getSheetAt(sheetnumber);

String data = sheet.getRow(row).getCell(column).getStringCellValue();

return data;

}

public int getRowCount(int sheetIndex)

{

int row = work\_book.getSheetAt(sheetIndex).getLastRowNum();

row = row + 1;

return row;

}

In the code above, the user has used Apache POI libraries to fetch the data from the excel file. Next, it will point to the data present in the excel file and then enter the relevant username and password to the sign in page.

****Note****: The same thing can be done using a Data provider in TestNG. But to fetch the data from the Excel sheet, the user needs Apache POI jar files.

****Note****: Please enter one valid credential to test.

### ****Advantages of Data Driven Testing Framework****

1. Allows testing of the application with multiple sets of data values during regression testing
2. Separates the test case data from the executable test script
3. Allows reusing of Actions and Functions in different tests
4. Generates test data automatically. This is helpful when large volumes of random test data are necessary
5. Results in the creation of extensive code that is flexible and easy to maintain
6. Lets developers and testers separate the logic of their test cases/scripts from the test data
7. Allows execution of test cases several times which helps to reduce test cases and scripts
8. It does not let changes in test scripts affect the test data.

By incorporating data-driven testing using Selenium, testers can refine their test cases for more efficient execution. This shortens timelines, makes their lives easier and results in more thoroughly tested and better quality software.

## ****Hybrid framework****

Hybrid framework is a technique wherein we can make the best use of both Data Driven & Keyword Driven Selenium framework (s). Using the examples shown above in this blog, we can build a Hybrid framework by storing the methods to execute in an excel file (keyword driven approach) and passing these method names to the Java Reflection Class (data driven approach) instead of creating an ****If/Else**** loop in the “DriverScript” class.

Take a look at the modified “DriverScript” class in the below code snippet. Here, instead of using multiple If/ Else loops, data driven approach is used to read the method names from the excel file.

|  |  |
| --- | --- |
|  | **package** HybridFramework;  **import** java.lang.reflect.Method;  **public** **class** DriverScriptJava  {   //This is a class object, declared as 'public static'   //So that it can be used outside the scope of main[] method  **public** **static** Actions actionKeywords;  **public** **static** String sActions;   //This is reflection class object, declared as 'public static'   //So that it can be used outside the scope of main[] method  **public** **static** Method method[];  **public** **static** **void** main(String[] args) **throws** Exception   {   //Declaring the path of the Excel file with the name of the Excel file   String sPath = "C:UsersVardhanworkspaceSelenium Frameworks DemodataEngine.xlsx";   //Here we are passing the Excel path and SheetName to connect with the Excel file   //This method was created previously   ReadExcelData.setExcelFile(sPath, "Sheet1");   //Hard coded values are used for Excel row & columns for now   //Later on, we will use these hard coded value much more efficiently   //This is the loop for reading the values of the column (Action Keyword) row by row   //It means this loop will execute all the steps mentioned for the test case in Test Steps sheet  **for** (**int** iRow=1;iRow<=7;iRow++)   {   sActions = ReadExcelData.getCellData(iRow, 1);   //A new separate method is created with the name 'execute\_Actions'   //You will find this method below of the this test   //So this statement is doing nothing but calling that piece of code to execute   execute\_Actions();   }   }  //This method contains the code to perform some action  //As it is completely different set of logic, which revolves around the action only, it makes sense to keep it separate from the main driver script  //This is to execute test step (Action)  **private** **static** **void** execute\_Actions() **throws** Exception   {   //Here we are instantiating a new object of class 'Actions'   actionKeywords = **new** Actions();   //This will load all the methods of the class 'Actions' in it.   //It will be like array of method, use the break point here and do the watch   method = actionKeywords.getClass().getMethods();   //This is a loop which will run for the number of actions in the Action Keyword class   //method variable contain all the method and method.length returns the total number of methods  **for**(**int** i = 0;i<method.length;i++)   {    //This is now comparing the method name with the ActionKeyword value received from the excel  **if**(method[i].getName().equals(sActions))   { //In case of match found, it will execute the matched method    method[i].invoke(actionKeywords);     //Once any method is executed, this break statement will take the flow outside of for loop  **break**;   }   }   }  } |