**Apache POI API:**

Data driven testing from Excel

Maven Dependencies Setup:

Poi-ooxml and poi

**Strategy to Access Excel Data:**

Create object for XSSWorkbook class

Get Access to sheet

Get Access to all rows of Sheet

Access to specific row from all rows

Get Access to all cells of Row

Access the Data from Excel into Arrays

we will completely focus on data driving testing from Excel.

So if you have a data in Excel, how to pull those data from Excel worksheets into our Java test cases? - So by end of this section, our objective is to write a neat data-driven test cases

from connecting Excel to our test cases.

what are the different methods used to pull the data from Excel ?

Apache POI API.

This is the API used to connect from your Excel to your Java test case.

So it's an open source. You know that everything in Java is open source,

so Apache guys came out with one library for this connection between Excel to your Java test, and that library name is POI API.

Interview que :  how can you drive the data from Excel into your test? - using an Apache POI API.

So in this section, we are going to see

what is this POI API?

What are different methods available here ?

and how to connect and write a neat test on that, okay?

 I'll be using Maven project here to get all the POI API JARs, I will just define as dependencies in my project so that all the related POI API JARs will be in my machine if I simply pass Maven dependencies, instead of downloading the JARs and placing it in our project build path.

so before we start with Excel, let's have this POI API set up in our machine.

you have seen how to create a Maven project from your Command Prompt, right?

It's like, mvn archetype:generate, which will actually generate the Maven project for you

with a quick start template So same thing, you can actually do it from Eclipse only.

how to create a project from Eclipse itself ? So it's very simple, right click - New, and Maven Project – check the simple project – next - group ID is Framework here,

because we are seeing specific Java components required,

and for artifact, you can say as ExcelDriven - This is specific to your project,

So click on Finish, - and that will complete your ExcelDriven folder.

So right now, we just need only dependencies. We can still run a test as run as a Java application, but you have to get this POI API JARs into your machine.

(Poi-ooxml and poi)so these two are heart for working with this POI API.

so these two dependencies, let's get, and place it in pom.xml, so that automatically, all those related JARs will be pulled by Maven first.

Google - POI OOXML dependencies – copy the depen and paste it in pom.xml

Mvn repo – poi – copy the depe and paste it in pomxml ( before paste the depen in the pom.xml write the parent <dependencies> </dependencies>

These two dependencies are sufficient for us to work with Excel.

You need not download all these JARs specifically from internet.

You just got these two dependencies, which are required, and Maven does the work for you.

we are all set to write a code to take our data from Excel.

So I will also open an Excel, and do some basic stuff there.

we neatly document in our Excel, and we ask you to pull all these details from here

to our test case as an input.

That would be mobile test, web test case, or API test, whatever it is, for every test case, you should have some data like this, and you place it in excel.

So now, our duty is to call this into our Java test. So when I say Java test, as I said, it could be your Selenium test case, it could be your Appiam Mobile test case, or it could be your API test cases using REST Assured. So all these are comes under Java umbrella,

200. Strategy to Access Excel Data

XSSFWorkbook - This is a class which is present in this POI API. So with this class we are going to initiate an object to take a control over this Excel. So now to get a control to this entire Excel sheet

so first you need to create an object for this class ( XSSFWorkbook class ) because this class have all the methods which are required to pull the data from this Excel. So without creating an object for this class you cannot do anything.

Let's create an object. And once we have an object, then we'll get an access to this sheet.

Assume that there are multiple sheets.

In real time there could be a number of sheets, from this n number of sheets in this workbook

we will get access to the sheet which we require. This is step number two. Out of all sheets I'll go and access this sheet number one.

Once we get access to the sheet, a sheet is nothing but collection of rows, This is a sheet, and all these are rows. You have n number of rows. So from the sheet we will go and access

all the rows of that sheet. That's this one step.

Once you have access to all the rows, if you have some 100 rows,

if you are interested in the Purchase test case, we need row number 3, We'll only get a specific row from all the rows which we are interested I may pull this. So once I get access to specific row from all the rows, I will get access to all the cells of that row.

So once I get access to this row I will get access to each and every cell of this row.

And once if I get access to that cell I can pull the data, and you see that get access to all the cells and access the data from Excel once you stand in that specific cell.

( Excel -> sheet, Sheet -> rows, Rows -> cells ) --- That's how we will proceed.

Src/test/java – new – class – dataDriven – check the public – finish

go and create an object for this class.

If you want to create object for any class, this is how you define in Java.

XSSFWorkbook workbook = new XSSFWorkbook(fis);

How would this workbook know that this is the file it has to access?

Save and pass that location into this method argument. If you can pass a file path of this Excel into this, then only workbook will come to know where exactly it is present.

But this XSSFWorkbook accepts a file input stream argument.

So file input stream is a class in Java where it will create an object which have a power to read any file.

 how do you create FileInputStream argument?

Make sure you are extension is xlsx.

FileInputStream fis = new FileInputStream(“c://users//Rahul//Documents//demodata.xlsx”);

So this (fis)object will have permission to read this file.

So this is how you have to actually configure to make your workbook where exactly this file is located and where is, how do you access it. This access permission to that Excel is given by this fis.

Now for example, you have multiple sheets here. Sheet1(testdata), Sheet2(sample), Sheet3(demo), but you are interested in Sheet1 only. But how do you tell to your workbook to go and read only Sheet1?

 for example, if I get a result like there are 10 sheets present then I will loop for each and every sheet and compare whether the sheet name is what I am willing to check. Once it's matched, then I will access to that sheet.

 get the sheet count

int sheets=workbook.getNumberOfSheets();

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

{

If(workbook.getSheetName(i).equalsIgnoreCase(“testdata”))

{

XSSFSHEET sheet = workbook.getSheetAt(i);

}

}

201. Getting rows and its cells from Sheet

get an access to all the rows of sheet.

Req : identify test cases column. By scanning the entire row. First row.

So our code has to scan entire row and identify for which column this test case is present.

That is our step number one.

So once we identified the column, then we have to scan that complete column to identify this purchase test case.

Once you get this purchase row, now you are actual duty is to scan that entire purchase row and take this data.

Pull all the data Of that row and feed it into test.

first we need to reach this purchase test case

So first you have to scan your first header row to understand where exactly this test case column is present, which we need to do in this step one.

Once you have that test case column and you have to scan that entire column to identify your actual test case, which is purchase and you are doing it here in the second step, once you get that purchase, you see that.

Now you have to grab all the data of that purchase test case and you feed it into your test case. So for any automation using driving the data from Excel, these are the three steps which you commonly need to do.

// Identify TestCases column by scanning the entire 1st row

Iterator<Row> rows = sheet.iteratir(); // sheet is collection of rows

Row firstrow = rows.next();

Iterator<cell> ce=firstrow.cellIterator(); //row is collection of cells

While(ce.hasNext()) // hasnext – Boolean value

{

Cell value =ce.next();

If(value.getStringCellValue().equalsIgnoreCasr(“TestCases”))

{

// desired column

}

Sheet is nothing but collection of rows and to traverse each and every row I'm using iterator concept.

Once you get desired row Now row is nothing but collection of cells to traverse to each and every cell.

Again, I'm using iterator and I have moved to that desired.

202. Retrieving Data from Excel based on condition

we got a desired column value here, but how? We reached that particular point, but we have not yet grabbed the index of that column. So basically, we need to know the column index.

K(new variable)

203. Practise Exercise- Excel Driven testing -1

I'll get row count of this sheet. I'll get all the rows of this sheet, Once I get all the rows,

and now I iterate through each and every row.

I'll only search for column A. And again, I'll move to the next row until I get that Purchase.

I will iterate to all my rows by only scanning column A.

hasNext()  - if next row is present, then go inside the block

204. Practise Exercise- Excel Driven testing -2

I'll create one ArrayList. ArrayList, which accepts only String. ( we are write it on the top )

when loop is iterating for each and every cell, the data is getting stored in an ArrayList.

So all the data of that test case is now properly stored in this ArrayList.

I will actually remove all this code and I'll write it in a method.

So, getData is the method which will get the information of test case.

So basically all the data we are storing in an array, ArrayList in “a” object.

Create new class ( testSample )

In testSample - I need to access the method present in this dataDriven class

So first for that I have to create object to access any methods present in that class

If you want to get some data from Excel, just use this getData Utility and define the data there and that's it. And you will have the data here. So, once the data is in your test case,

now it's up to you.

Instead of hard coding in your actual test, if you want to drive it from Excel, do it this.

205. Practise Exercise- Excel Driven testing -3

I change "Add Profile" of any of the value to integer if run it fails bcze every where we write getstringcellvalue() in loop .

You need to write one wrapper code to check if it is numeric or string based upon that you have to extract.

So that's how you deal with different data type formats in the Excel.

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Iterator;

import org.apache.poi.ss.usermodel.Cell;

import org.apache.poi.ss.usermodel.CellType;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.util.NumberToTextConverter;

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

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

public class dataDriven {

//Identify Testcases coloum by scanning the entire 1st row

//once coloumn is identified then scan entire testcase coloum to identify purcjhase testcase row

//after you grab purchase testcase row = pull all the data of that row and feed into test

public ArrayList<String> getData(String testcaseName) throws IOException // method in class

{

//fileInputStream argument

ArrayList<String> a=new ArrayList<String>();

FileInputStream fis=new FileInputStream("C:\\Users\\Dell\\Documents\\demodata.xlsx");

XSSFWorkbook workbook=new XSSFWorkbook(fis);

int sheets=workbook.getNumberOfSheets();

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

{

if(workbook.getSheetName(i).equalsIgnoreCase("testdata"))

{

XSSFSheet sheet=workbook.getSheetAt(i);

//Identify Testcases coloum by scanning the entire 1st row

Iterator<Row> rows= sheet.iterator();// sheet is collection of rows

Row firstrow= rows.next(); // 1st row -- if you mention the step 2nd time the control will goes to 2nd column

Iterator<Cell> ce=firstrow.cellIterator();//row is collection of cells

int k=0;

int coloumn = 0;

while(ce.hasNext()) // if next cell is present

{

Cell value=ce.next();

if(value.getStringCellValue().equalsIgnoreCase("TestCases"))

{

coloumn=k; // k =4 iteration

}

k++;

}

System.out.println(coloumn);

////once coloumn is identified then scan entire testcase column to identify purchase testcases row

while(rows.hasNext())

{

Row r=rows.next();

if(r.getCell(coloumn).getStringCellValue().equalsIgnoreCase(testcaseName))

{

////after you grab purchase testcase row = pull all the data of that row and feed into test

Iterator<Cell> cv=r.cellIterator();

while(cv.hasNext())

{

Cell c= cv.next();

if(c.getCellType()==CellType.STRING)

{

a.add(c.getStringCellValue());

}

else{

a.add(NumberToTextConverter.toText(c.getNumericCellValue()));

}

}

}

}

}

}

return a;

}

public static void main(String[] args) throws IOException {

// TODO Auto-generated method stub

}

}

import java.io.IOException;

import java.util.ArrayList;

public class testSample {

public static void main(String[] args) throws IOException {

// TODO Auto-generated method stub

dataDriven d = new dataDriven();

ArrayList data = d.getData("Add Profile");

System.out.println(data.get(0));

System.out.println(data.get(1));

System.out.println(data.get(2));

System.out.println(data.get(3));

}

}

206. Importance of Data Provider and Excel Integration for better Data driven

so you might be heard about TestNG Data Provider and Excel driving data separately, but why do we need to integrate this both?

This section we are going to integrate two different data driven concepts for better and robust automation test building.

why do we need to integrate these both futures to build one more future ?

assume that if you want to drive the data using TestNG Data Provider you will create data provider like this in your test case.

So there are three different arrays here. This is first set of data, which we also call as first array because this is returning multi-dimensional array.

when you pass this data provider to this @Test with three different sets of data, now this test case will execute three times. For the first time, it'll execute with this one set, second time, this one set, and third with this data set. So you will see the test output, like, test run 3 pass 3, fail 0

TestNG will treat it as three separate test cases with three separate set data.

Disadvantage here is In your Java files, you are declaring the data. So, some teams may not like to have the data in your Java files.

Instead, they want to drive this data from external sources like xml, Excel, and js.

So, for that we have a new concept called excel driven where you can drive the data.

Ex : there are three test cases again – greeting, communicate, ID

If you want to run this get data with these three sets of data using Excel.

you can write some for loop inside it and say count the number of rows you have in the Excel

loop through the rows and complete that test each and every time with each and every data set. but ultimately output you will see as test run one only. if it is passed 1 fail zero.

Though it ran three different times that is inside the test only.

So this is the main method. Get data, inside this we are looping three times. So ultimately your report will be like one test only and if it's passed 1 zero failures.

Tomorrow, if second data test fail, it will show us whole test as a failure, but it'll not give details, like, first set data passed, Even one of the three test data fail. Whole test will report as failure because Excel do not have a knowledge of running this whole test three different times.

But whereas with the TestNG, if you have three different sets of data, this have an ability to show in report that test run three different times and it will give you report for each and every run. How what happened for first data, second data and third data. With Excel, you can ultimately achieve the same thing but overall report will be showed as only one test pass

because you are writing the code inside the method.

whereas data provider, you are writing the code you are connecting this data from outside the method. You see that, you are connecting this data provider on test outside level.

So now, our main goal is to remove this hard coding and dynamically get this data

into this multidimensional array from Excel. Pull this data from Excel and place it here.

If you simply past the Excel path that should read this whole content and it should send it here. That's what we are going to achieve.

By doing this, we are satisfying the condition

of eliminating hard coding the data in Java test files. And also we are utilizing the future of data provider by connecting the test and running it with three different times with three different sets and to get a report of test run three instead of one.

TestNG Data Provider + Excel Integration –

@DataProvider(name=”Authenticate”)

Public static Object[][] credentials[] {

return new Object[][] {{“user”,”pwd”}, {“user\_1”,”pwd@123”}, {“user\_2”,”pwd@99”}};

}

@Test(name=Autheticate)

Public void getData()

{

For()

{

//3 different

}

207. Understand Dataprovider and how it sends data in Multidimensional array

First is to start with DataProvider and then bring the Excel integration into it.

Right click ->new maven project -> next -> select maven-archetype-quickstart 1.1 ->next ->

Group id : dataDriven, Artifact id : excelDataProvider -> finish

Src/test/java – right click->new class(dataProvider)-> Finish

with the help of TestNG DataProvider we can send multiple sets of data to our tests. So each set of data will be represented as one array. so that your actual test reads that array from DataProvider and execute the tests.

Now, if you send five sets of data as five arrays from data provider to your tests, then your tests will run five times with five separate sets of data, which are nothing but five different arrays.

So these are three different arrays we are sending to data object. And then we will pass this data object to the test. So the test will run for first time with this complete data, whatever you are passing. For the second time, it will run with array of two, and third time it will run with third array.

And this data object is nothing but multi-dimensional object.

why I am giving written type as an object here?- so object is a super set of all the data types.

So that means your array can be integer array, Your array can be string array.

Object is a root. And this is a super class of all subtypes, like string, integer, long. For all these data types this is a super set. So when you declare object then that will accept any data type.

if you want to print all together, just give concatenation(+) operator to separate each and every variable which you want to print.

**package** dataDriven.excelDataProvider;

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

**import** org.testng.annotations.Test;

**public** **class** dataProvider {

@Test(dataProvider="driveTest") // name of which dataprovider ur listening to

**public** **void** testCaseData(String greeting,String communication,**int** id)

{

System.***out***.println(greeting+communication+id);

}

@DataProvider(name="driveTest") // any name

**public** Object[][] getData()

{

Object[][] data={{"hello","text",1},{"bye","message",143},{"solo","call",453}};

**return** data;

}

}

Output:

hellotext1

byemessage143

solocall453

PASSED: dataDriven.excelDataProvider.dataProvider.testCaseData("bye", "message", 143)

PASSED: dataDriven.excelDataProvider.dataProvider.testCaseData("hello", "text", 1)

PASSED: dataDriven.excelDataProvider.dataProvider.testCaseData("solo", "call", 453)

208. Get Excel dependencies and connect from java code to excel

give the Eclipse knowledge to this project.

get the Excel dependencies

google – apache poi api maven ->copy and paste the depen in pom.xml

mvn repo – apache poi ooxml -> copy and paste the depen in pom.xml

mvn repo – TestNG -> copy and paste the depen in pom.xml

So basically, you need to create outer loop and an inner loop. Outer loop will take care of this row, and the inner loop will take care of its column.

209. Live demo on integrating Excel to Dataprovider to parameterize data

It will go to outer loop only when complete inner loop is done.

 Excel may have integer strings or double number, any data type, right? But when you put it into testNG data provider, and when your method is catching it, you are clearly telling that it should be string only. So how do you do that format?

let's do one formatting here. So after you grab the data from Excel, before you store it into your multi-dimensional array, let's format that and make sure that it converted into string, no matter if it is integer or double.

So if you don't bring this data provider concept, though you, you can still loop here the Excel data, but you will end up having only one test. So you will clearly know which test pass, which test fail.

**package** dataDriven.excelDataProvider;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** org.apache.poi.ss.usermodel.DataFormatter;

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

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

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

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

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

**import** org.testng.annotations.Test;

**public** **class** dataProvider {

DataFormatter formatter = **new** DataFormatter();

@Test(dataProvider="driveTest")

**public** **void** testCaseData(String greeting,String communication,String id)

{

System.***out***.println(greeting+communication+id);

}

@DataProvider(name="driveTest")

**public** Object[][] getData() **throws** IOException

{

FileInputStream fis = **new** FileInputStream("C:\\Users\\Dell\\Documents\\excelDriven.xlsx");

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

XSSFSheet sheet = wb.getSheetAt(0);

**int** rowCount=sheet.getPhysicalNumberOfRows();

XSSFRow row=sheet.getRow(0);

**int** colCount=row.getLastCellNum(); // 4

Object data[][]=**new** Object[rowCount-1][colCount];

**for**(**int** i=0;i<rowCount-1;i++)

{

row=sheet.getRow(i+1); // get 1st row in excel

**for**(**int** j=0;j<colCount;j++)

{

XSSFCell cell=row.getCell(j);

//it is converting in to string of any datatype it stored in to multi dimensional array

data[i][j]=formatter.formatCellValue(cell);

}

}

**return** data;

}

}