Cucumber

Feature File

It is plain English Language

Everyone should know what we are going to do (or)what we have done by reading this file (Non coders as well).

We are using Gherkin keywords over here

Features - I always giving User stories over here.

Scenario – We are passing only one set of data

Scenario Outline – Executing same scenario with different set of datas

Here we use Examples keyword mandatory and will pass the value using "pipe" symbol

Given- I will pass the condition(Pre condition)

When – I will pass the action, that means we are passing credentials or data in to the field.

And – it is used to connect the two actions

Then – It is used for validation purpose , for example , we can say , I have completed one scenario after that

It will navigated to another page or another field.

But - But keyword is used to add invalid test cases

Background – here we can maintain all the common functionalities

Test Runner Class

Test Runner class , yeah…..we are using link between feature file and Stepdefinition class.

Here we are using 2 Annotations that are

@RunWith (Cucumber.class)

@CucumberOptions

Features – yes , Here we pass path of the feature file

Glue – Mm mm…Here we pass the package name of the step definition class

DryRun – Yes , It is a Boolean……and ……..it will show the missing snippets in the console after running test runner class.

Here Execution will not be done….only showing missing snippets.

Strict – Yes…..it is also same as DryRun,……but here……Execution will be done…….After execution only it will show missing snippets in the console…….That is the main difference between DryRun and Strict.

Monochrome – it will convert the console As human readable form.

Plugin – Yes….Plugin we are using for report generation

And in the plugin,,,we are using "**pretty**", …….pretty is nothing but ..in console …….we will be getting….what scenarios have been executed…or……what scenarios have been skipped.

We can also use “**Usage**”…….but it is like key and value….like json report.

Hooks Class

Here we have 2 Annotations….

This allows us to manage the code workflow better… and helps to reduce code redundancy……..

Package -----> Cucumber.api.java

@**Before** – To set up the webdriver

@**After** – To clear up the webdriver…..and……here…..if any scenarios have been failed…… we can also add

Takes Screenshot interface over here…..

In **TakesScreenshot** we are using “**embed**” method.

Program:

```
Public class HooksClass extends BaseClass(){

@Before

public void beforeScenario(){

get.driver();

launchUrl(“”);

}

@After

Public void afterScenario(Scenario s) {

if(s.failed) {
```

```
TakesScreenshot ts=( TakesScreenshot)driver;
byte[] screenshotAs=ts.getScreenshotAs(OutputType.BYTES);
String name=s.getName();
s.embed(ScreenshotAs,name+".png");
}
driver.close();
}}
```

Report Generation

We use JVM Report

It is a Graphical Representation

Dependency------> Cucumber reporting 5.3.1

Need to create a folder in project

Then we need to create a class (Reporting class)

Program:

```
public class ReportingClass {
public static void JVMReportGeneration(String json){
File file = new File("Folder path");
Configuration con=new Configuration(file,"project name");
Con.addClassification("platform","WIN-10");
Con.addClassification("Browser","Chrome");
Con.addClassification("SprintNo","32");
List<String> li=new ArrayList<String>();
```

li.add(json);

ReportBuilder builder=new ReportBuilder(li,con);

builder.generateReports();

}

Explanations:

First need to create class

We will create method and pass one string

Then we will pass path of the folder using class **File**

Then we will configure file and project using **Configuration**

Then we use the method "**addClassification**", here we pass the platform, browser, sprint number etc.

Then we put in to the **list**

Then we create object using **ReportBuilder**, here we pass the list and reference name of the configuration.

Then we go with **generateReport** method.

Then go to TestRunnerClass and inside the class need to give the annotation called @**AfterClass** then create a method.

After that reportingclass.method.

Report TestRunnerClass

Public class TestRunnerClass{

Public static void afterClass(){

ReportingClass.
JVMReportGeneration("……//projctname.json")

}

}

Tags

Tags are always declared in feature level and Scenario &Scenario outline

We can Ignore the tests using tags or we can club different tags and run.

In feature file

@Smoke Test

@Regression Test

@Sanity Test

If we declare the @SmokeTest then go to CucumberOptions

Case1:

tags={@SmokeTest} then it will execute only this tag

Case2:

tags={~@SmokeTest} then it will except this tag.

~ ----> Negation

And

tags={“@SmokeTest”,”@SanityTest”}

OR

Tags={“@SmokeTest,@SanityTest”}