***Dry Run***

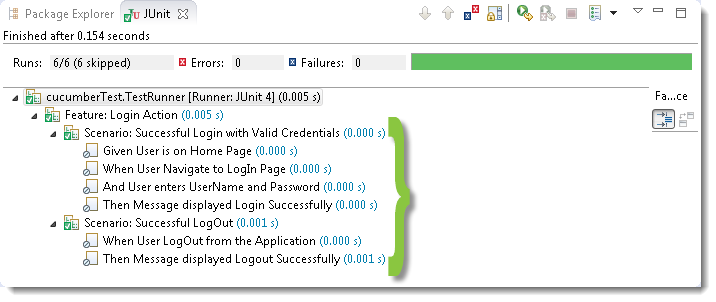
***dryRun****option can either set as****true****or****false****. If it is set as true, it means that Cucumber will only check that every Step mentioned in the Feature File has corresponding code written in Step Definition file or not. So in case any of the functions are missed in the Step Definition for any Step in Feature File, it will give us the message.*

*For practice just add the code ‘****dryRun = true‘*** *in****TestRunner****class:*

***TestRunner Class***

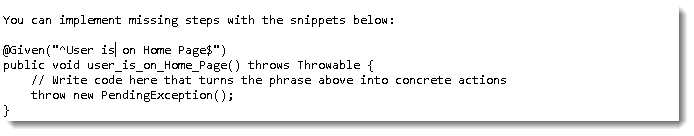
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  ,dryRun = true  )    public class TestRunner {    } |

*Now give it a run by Right Click* on **TestRunner** class and Click **Run As***>***JUnit Test.***Cucumber*will run the script and the result will be shown in the left-hand side*project explorer window* in *JUnit* tab.



*Take a look at the time duration at the end of the every*Steps*, it is (***0.000s***). It means none of the*Step*is executed but still,*Cucumber*has made sure that every Step has the corresponding method available in the*Step Definition*file.*

*Give it a try, remove the ‘***@Given(“^User is on Home Page$”)***‘ statement from the***Test\_Steps***class and run the***TestRunner** *class again. You would get the following message:*



***Monochrome***

This option can either set as ***true***or ***false***. If it is set as *true*, it means that the *console output* for the *Cucumber test*are much more readable. And if it is set as *false*, then the *console output* is not as readable as it should be. For practice just add the code ‘***monochrome = true***‘ in ***TestRunner*** class:

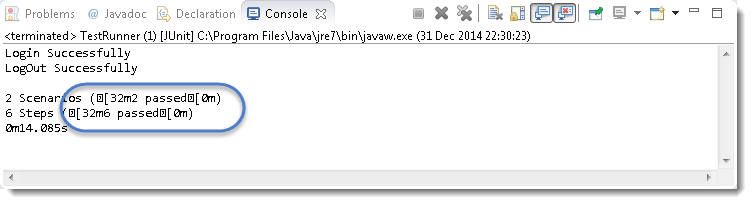
***TestRunner Class***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  ,monochrome = true  )    public class TestRunner {    } |

*Now give it a run by Right Click* on ***TestRunner*** class and Click ***Run As****>****JUnit Test.****Cucumber*will run the script and Console Output will display like this:



This time change the value from *true* to *false* and run the ***TestRunner*** class again. This time the *Console Output* will look like this:



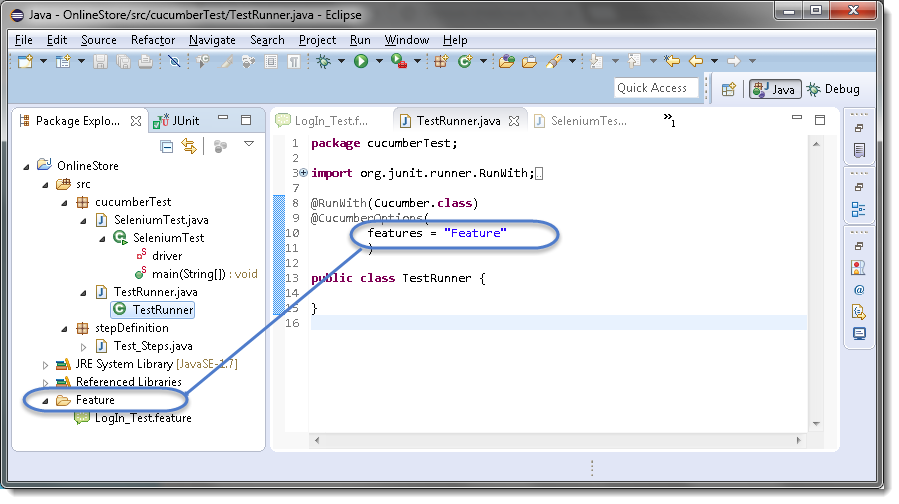
### *****Features*****

***Features Options***helps Cucumber to locate the Feature file in the project folder structure. You must have notices that we have been specifying the Feature Option in the ***TestRunner*** class since the first chapter. All we need to do is to specify the folder path and Cucumber will automatically find all the ‘***.features***‘ extension files in the folder. It can be specified like:

***features = “Feature“***

Or if the Feature file is in the deep folder structure

***features = “src/test/features“***



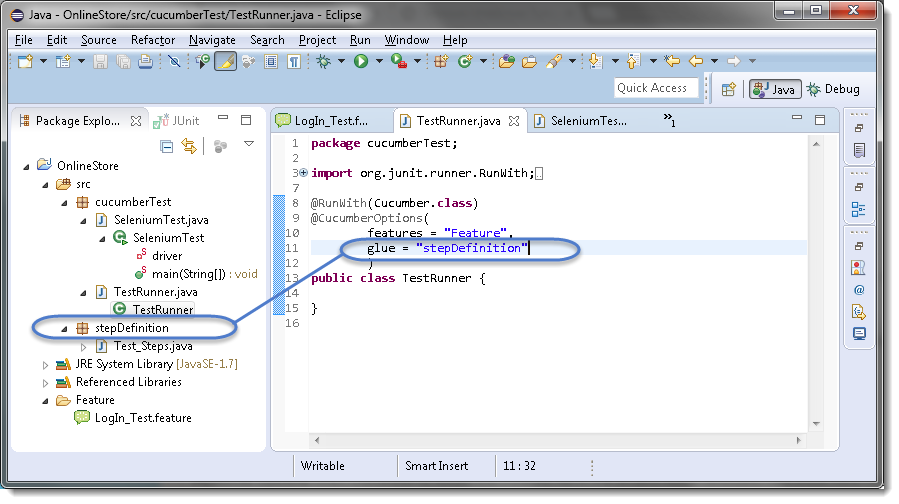
### *****Glue*****

It is almost the same think as Features Option but the only difference is that it helps Cucumber to locate the ***Step Definition file.*** Whenever Cucumber encounters a Step, it looks for a Step Definition inside all the files present in the folder mentioned in **Glue Option**. It can be specified like:

***glue = “stepDefinition“***

Or if the Step Definition file is in the deep folder structure

***glue = “src/test/stepDeinition“***



### *****Format*****

**Format Option** is used to specify different formatting options for the output reports. Various options that can be used as for-matters are:

***Pretty:***Prints the Gherkin source with additional colors and stack traces for errors. Use below code:

***format = {“pretty“}***

***HTML:***This will generate a HTML report at the location mentioned in the for-matter itself. Use below code:

***format = {“ pretty”, “html:Folder\_Name“}***

***JSON:***This report contains all the information from the gherkin source in JSON Format. This report is meant to be post-processed into another visual format by 3rd party tools such as Cucumber Jenkins. Use the below code:

***format = {“json:Folder\_Name/cucumber.json“}***

***JUnit:*** This report generates XML files just like Apache Ant’s JUnit report task. This XML format is understood by most Continuous Integration servers, who will use it to generate visual reports. use the below code:

***format = { “junit:Folder\_Name/cucumber.xml“}***

Data Driven testing :

1. Scenario Outline and Example
2. Data table with single input
3. Data table with multiple input

## Data-Driven Testing in Cucumber

***LogIn\_Test.fetaure***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Feature: Login Action    Scenario: Successful Login with Valid Credentials  Given User is on Home Page  When User Navigate to LogIn Page  And User enters UserName and Password  Then Message displayed Login Successfully    Scenario: Successful LogOut  When User LogOut from the Application  Then Message displayed LogOut Successfully |

***Test\_Steps.java***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | package stepDefinition;    import java.util.concurrent.TimeUnit;    import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;    import cucumber.api.java.en.Given;  import cucumber.api.java.en.Then;  import cucumber.api.java.en.When;    public class Test\_Steps {  public static WebDriver driver;  @Given("^User is on Home Page$")  public void user\_is\_on\_Home\_Page() throws Throwable {  driver = new FirefoxDriver();      driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);      driver.get("http://www.store.demoqa.com");  }    @When("^User Navigate to LogIn Page$")  public void user\_Navigate\_to\_LogIn\_Page() throws Throwable {  driver.findElement(By.xpath(".//\*[@id='account']/a")).click();  }    @When("^User enters UserName and Password$")  public void user\_enters\_UserName\_and\_Password() throws Throwable {  driver.findElement(By.id("log")).sendKeys("testuser\_1");      driver.findElement(By.id("pwd")).sendKeys("Test@123");      driver.findElement(By.id("login")).click();  }    @Then("^Message displayed Login Successfully$")  public void message\_displayed\_Login\_Successfully() throws Throwable {  System.out.println("Login Successfully");  }    @When("^User LogOut from the Application$")  public void user\_LogOut\_from\_the\_Application() throws Throwable {  driver.findElement (By.xpath(".//\*[@id='account\_logout']/a")).click();  }    @Then("^Message displayed LogOut Successfully$")  public void message\_displayed\_LogOut\_Successfully() throws Throwable {  System.out.println("LogOut Successfully");  }  } |

***TestRunner.java***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  )    public class TestRunner {    } |

### Parameterizing without Example Keyword

Now the task is to ***Parameterizing the UserName and Password***. Which is quite logical, why would anybody want to hardcode the UserName & Password of the application. As there is a high probability of changing both.

1) Go to the ***Feature File*** and change the statement where passing Username & Password as per below:

***And User enters “testuser\_1” and “Test@123“***

In the above statement, we have passed Username & Password from the Feature File which will feed in to Step Definition of the above statement automatically. Cucumber will do the trick for us. After the above changes, the code will look like this:

***LogIn\_Test.feature***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Feature: Login Action    Scenario: Successful Login with Valid Credentials  Given User is on Home Page  When User Navigate to LogIn Page  And User enters "testuser\_1" and "Test@123"  Then Message displayed Login Successfully    Scenario: Successful LogOut  When User LogOut from the Application  Then Message displayed LogOut Successfully |

2) Changes in the *Step Definition* file is also required to make it understand the *Parameterization of the feature file*. So, it is required to update the *Test Step* in the *Step Definition* file which is linked with the above-changed *Feature* file statement. Use the below code:

***@When(“^User enters \”(.\*)\” and \”(.\*)\”$”)***

The same can be achieved by using the below code as well:

***@When(“^User enters \”([^\”]\*)\” and \”([^\”]\*)\”$”)***

With the help of the above statements, *Cucumber* will understand that the associated*Test\_Step* is expecting some parameters.

3) Same parameters should also go into the associated *Test\_Step*. As the Test step is nothing but a simple Java method, syntax to accept the parameter in the Java method is like this:

***public void user\_enters\_UserName\_and\_Password(String username, String password) throws Throwable {***

***}***

4) Now the last step is to feed the parameters in the actual core statements of *Selenium WebDriver*. Use the below code:

***driver.findElement(By.id(“log“)).sendKeys(username);***  
***driver.findElement(By.id(“pwd“)).sendKeys(password);***  
***driver.findElement(By.id(“login“)).click();***

After making the above changes, the method will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @When("^User enters \"(.\*)\" and \"(.\*)\"$")  public void user\_enters\_UserName\_and\_Password(String username, String password) throws Throwable {  driver.findElement(By.id("log")).sendKeys(username);      driver.findElement(By.id("pwd")).sendKeys(password);      driver.findElement(By.id("login")).click();  } |

5) Run the test by *Right Click* on ***TestRunner class*** and Click ***Run As  > JUnit Test*** Application. You would notice that the *Cucumber* will open the Website in the browser and enter *username & password*which is passed from the *Feature File*.

## Data-Driven Testing Using Examples Keyword

1) Enter the***Example Data*** just below the *LogIn* Scenario of the *Feature* File.

***Examples:***  
***| username  | password  |***  
***| testuser\_1 | Test@153 |***  
***| testuser\_2 | Test@153 |***

***Note****: The table must have a header row corresponding to the variables in the Scenario Outline steps.*

The Examples section is a table where each argument variable represents a column in the table, separated by “***|***”. Each line below the header represents an individual run of the test case with the respective data. As a result, if there are 3 lines below the header in the Examples table, the script will run 3 times with its respective data.

2) Need to update the Statement in the *feature* file, which tells *Cucumber* to enter *username & Password*.

***And User enters <username> and <password>***

*Cucumber* understands the above statement syntax and looks for the ***Examples***Keyword in the test to read the *Test Data*.

***The complete code will look like this***:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Feature: Login Action    Scenario Outline: Successful Login with Valid Credentials  Given User is on Home Page  When User Navigate to LogIn Page  And User enters "<username>" and "<password>"  Then Message displayed Login Successfully  Examples:      | username   | password |      | testuser\_1 | Test@153 |      | testuser\_2 | Test@153 | |

3) There are no changes in ***TestRunner***class.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  )    public class TestRunner {    } |

4) There are no changes in ***Test\_Steps*** file from the previous chapter.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | package stepDefinition;    import java.util.concurrent.TimeUnit;    import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;    import cucumber.api.java.en.Given;  import cucumber.api.java.en.Then;  import cucumber.api.java.en.When;    public class Test\_Steps {  public static WebDriver driver;  @Given("^User is on Home Page$")  public void user\_is\_on\_Home\_Page() throws Throwable {  driver = new FirefoxDriver();      driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);      driver.get("http://www.store.demoqa.com");  }    @When("^User Navigate to LogIn Page$")  public void user\_Navigate\_to\_LogIn\_Page() throws Throwable {  driver.findElement(By.xpath(".//\*[@id='account']/a")).click();  }    @When("^User enters \"(.\*)\" and \"(.\*)\"$")  public void user\_enters\_UserName\_and\_Password(String username, String password) throws Throwable {  driver.findElement(By.id("log")).sendKeys(username);      driver.findElement(By.id("pwd")).sendKeys(password);      //driver.findElement(By.id("login")).click();  }    @Then("^Message displayed Login Successfully$")  public void message\_displayed\_Login\_Successfully() throws Throwable {  System.out.println("Login Successfully");  }    @When("^User LogOut from the Application$")  public void user\_LogOut\_from\_the\_Application() throws Throwable {  driver.findElement (By.xpath(".//\*[@id='account\_logout']/a")).click();  }    @Then("^Message displayed LogOut Successfully$")  public void message\_displayed\_LogOut\_Successfully() throws Throwable {  System.out.println("LogOut Successfully");  }    } |

5) Run the test by *Right Click* on ***TestRunner class*** and Click ***Run As  > JUnit Test*** Application.

# Data Tables in Cucumber

***Data Tables in Cucumber*** are quite interesting and can be used in many ways. DataTables are also used to handle large amounts of data. They are quite powerful but not the most intuitive as you either need to deal with a ***list of maps*** or a ***map of lists***.

## Difference between Scenario Outline & Data Table

***Scenario Outline:***

* This uses Example keyword to define the test data for the Scenario
* This works for the whole test
* Cucumber automatically run the complete test the number of times equal to the number of data in the Test Set

***Test Data:***

* No keyword is used to define the test data
* This works only for the single step, below which it is defined
* A separate code needs to understand the test data and then it can be run single or multiple times but again just for the single step, not for the complete test

As I said above, the Data Tables can be used in many ways because it has provided many different methods to use. Let’s just go through a few most popular methods. I will choose a simple scenario to illustrate the working of the Data Table but we will make effective use of this when we will do **Cucumber Framework** in the next series of this ***Cucumber Tutorial***.

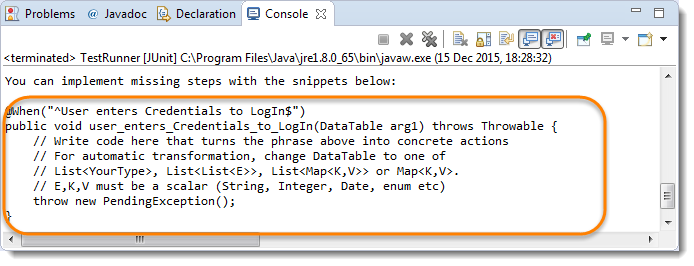
## Data Tables in Cucumber

In this example, we will pass the test data using the data table and handle it using***Raw()*** method.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Scenario: Successful Login with Valid Credentials  Given User is on Home Page  When User Navigate to LogIn Page  And User enters Credentials to LogIn      | testuser\_1 | Test@153 |  Then Message displayed Login Successfully |

The complete scenario is same as what we have done earlier. But the only difference is in this, we are not passing parameters in the step line and even we are not using Examples test data. We declared the data under the step only. So we are using Tables as arguments to Steps.

If you run the above scenario without implementing the step, you would get the following error in the Eclipse console window.



Copy the above hint in the Step Definition file and complete the implementation.

***The implementation of the above step will be like this:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | The implementation of the above step will belike this:  @When("^User enters Credentials to LogIn$")  public void user\_enters\_testuser\_\_and\_Test(DataTable usercredentials) throws Throwable {    //Write the code to handle Data Table  List<List<String>> data = usercredentials.raw();    //This is to get the first data of the set (First Row + First Column)  driver.findElement(By.id("log")).sendKeys(data.get(0).get(0));    //This is to get the first data of the set (First Row + Second Column)      driver.findElement(By.id("pwd")).sendKeys(data.get(0).get(1));        driver.findElement(By.id("login")).click();  } |

***The complete test Implementation***

***Test Runner Class***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  )    public class TestRunner {    } |

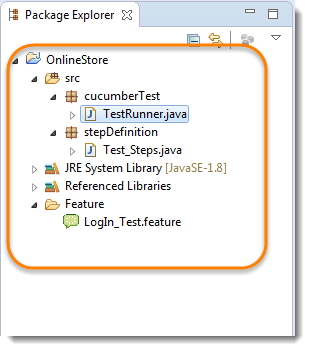
***Feature File***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | Feature: Login Action    Scenario: Successful Login with Valid Credentials  Given User is on Home Page  When User Navigate to LogIn Page  And User enters Credentials to LogIn      | testuser\_1 | Test@153 |  Then Message displayed Login Successfully    Scenario: Successful LogOut  When User LogOut from the Application  Then Message displayed LogOut Successfully |

***Step Definition***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | package stepDefinition;    import java.util.List;  import java.util.concurrent.TimeUnit;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  import cucumber.api.DataTable;  import cucumber.api.java.en.Given;  import cucumber.api.java.en.Then;  import cucumber.api.java.en.When;    public class Test\_Steps {  public static WebDriver driver;  @Given("^User is on Home Page$")  public void user\_is\_on\_Home\_Page() throws Throwable {  driver = new FirefoxDriver();      driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);      driver.get("http://www.store.demoqa.com");  }    @When("^User Navigate to LogIn Page$")  public void user\_Navigate\_to\_LogIn\_Page() throws Throwable {  driver.findElement(By.xpath(".//\*[@id='account']/a")).click();  }    @When("^User enters Credentials to LogIn$")  public void user\_enters\_testuser\_\_and\_Test(DataTable usercredentials) throws Throwable {  List<List<String>> data = usercredentials.raw();  driver.findElement(By.id("log")).sendKeys(data.get(0).get(0));      driver.findElement(By.id("pwd")).sendKeys(data.get(0).get(1));      driver.findElement(By.id("login")).click();  }    @Then("^Message displayed Login Successfully$")  public void message\_displayed\_Login\_Successfully() throws Throwable {  System.out.println("Login Successfully");  }    @When("^User LogOut from the Application$")  public void user\_LogOut\_from\_the\_Application() throws Throwable {  driver.findElement (By.xpath(".//\*[@id='account\_logout']/a")).click();  }    @Then("^Message displayed LogOut Successfully$")  public void message\_displayed\_LogOut\_Successfully() throws Throwable {  System.out.println("LogOut Successfully");  }  } |

***Project Explorer***



Run the test by *Right Click* on ***TestRunner class*** and Click ***Run As  > JUnit Test*** Application. you will notice that Cucumber will automatically figure out, what to provide in the Username and Password field.

## Maps in Data Tables

*Maps in Data Tables* can be used if different ways. ***Headers***can also be defined for the *data tables.* A same step can be executed multiple times with different set of test data using ***Maps***.

***Maps in Data Tables with Header***

In the previous chapter of [***Data Tables in Cucumber***](http://toolsqa.com/cucumber/data-tables-in-cucumber/)***,*** we pass *Username & Password* without Header, due to which the test was not much readable. What if there will be many columns. The basic funda of BDD test is to make the Test in Business readable format, so that business users can understand it easily. Setting Header in Test data is not a difficult task in Cucumber. take a look at a below Scenario.

***Feature File Scenario***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Scenario: Successful Login with Valid Credentials  Given User is on Home Page  When User Navigate to LogIn Page  And User enters Credentials to LogIn  | Username   | Password |      | testuser\_1 | Test@153 |  Then Message displayed Login Successfully |

***The implementation of the above step will be like this:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @When("^User enters Credentials to LogIn$")  public void user\_enters\_testuser\_and\_Test(DataTable usercredentials) throws Throwable {    //Write the code to handle Data Table  List<Map<String,String>> data = usercredentials.asMaps(String.class,String.class);  driver.findElement(By.id("log")).sendKeys(data.get(0).get("Username"));      driver.findElement(By.id("pwd")).sendKeys(data.get(0).get("Password"));      driver.findElement(By.id("login")).click();             } |

***Maps in Data Tables with Multiple Test Data***

In this test we will pass *Username and Password* two times to the test step. So our test should enter *Username & Password*once, click on *LogIn* button and repeat the same steps again.

***Feature File Scenario***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Scenario: Successful Login with Valid Credentials  Given User is on Home Page  When User Navigate to LogIn Page  And User enters Credentials to LogIn  | Username   | Password |      | testuser\_1 | Test@153 |      | testuser\_2 | Test@154 |  Then Message displayed Login Successfully |

***The implementation of the above step will be like this:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @When("^User enters Credentials to LogIn$")  public void user\_enters\_testuser\_and\_Test(DataTable usercredentials) throws Throwable {    //Write the code to handle Data Table  for (Map<String, String> data : usercredentials.asMaps(String.class, String.class)) {  driver.findElement(By.id("log")).sendKeys(data.get("Username"));      driver.findElement(By.id("pwd")).sendKeys(data.get("Password"));      driver.findElement(By.id("login")).click();  }    } |

***Map Data Tables to Class Objects***

Luckily there are easier ways to access your data than *DataTable*. For instance you can create a *Class-Object* and have Cucumber map the data in a table to a list of these.

***Feature File Scenario***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Scenario: Successful Login with Valid Credentials  Given User is on Home Page  When User Navigate to LogIn Page  And User enters Credentials to LogIn  | Username   | Password |      | testuser\_1 | Test@153 |      | testuser\_2 | Test@154 |  Then Message displayed Login Successfully |

***The implementation of the above step will be like this:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @When("^User enters Credentials to LogIn$")  public void user\_enters\_testuser\_and\_Test(List<Credentials>  usercredentials) throws Throwable {    //Write the code to handle Data Table  for (Credentials credentials : usercredentials) {  driver.findElement(By.id("log")).sendKeys(credentials.getUsername());      driver.findElement(By.id("pwd")).sendKeys(credentials.getPassword());      driver.findElement(By.id("login")).click();  }  } |

***Class Credentials***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | package stepDefinition;    public class Credentials {  private String username;  private String password;    public String getUsername() {          return username;      }  public String getPassword() {          return password;      }  } |

In Cucumber All the methods identified by Given keyword.

What is Test Plan?

Bug life cycle?

What is Test strategy?

Difference between delete, drop and truncate?

***CREATE :***

CREATE TABLE table\_name (column\_name column\_type);

EX:

CREATE TABLE student (name varchar (20), id int NOT NULL , age int, PRIMARY KEY(id))

***DROP TABLE:***

DROP TABLE table\_name;

EX:

DROP TABLE student

***ALTER TABLE:***

ALTER TABLE table\_name

ADD column\_name datatype;

EX:

ALTER TABLE employee ADD salary int

***INSERT :***

INSERT INTO table\_name ( field1, field2,...fieldN )

VALUES

( value1, value2,...valueN );

EX:

INSERT into student (name, id, age) values ('Don123', 2,21)

***SELECT:***

SELECT field1, field2,...fieldN

FROM table\_name1, table\_name2...

[WHERE Clause]

[OFFSET M ][LIMIT N]

EX:

SELECT \* FROM student

***UPDATE:***

UPDATE table\_name SET field1 = new-value1, field2 = new-value2

[WHERE Clause]

EX:

UPDATE student SET name='DON' where id=1

***DELETE :***

DELETE FROM table\_name [WHERE Clause]

EX:

DELETE FROM student WHERE id=2

***DISTINCT:***

SELECT DISTINCT column1, column2, ...

FROM table\_name;

EX:

SELECT DISTINCT id FROM student

WHERE

LIKE

Count, AVG, Sum, Min, Max

Between

In

Group By

Having

***ORDER BY:***

SELECT column1, column2, ...

FROM table\_name

ORDER BY column1, column2, ... ASC|DESC;

The ORDER BY keyword sorts the records in ascending order by default.

EX:

SELECT \* FROM student ORDER BY id DESC

***IN:***

The IN operator allows you to specify multiple values in a WHERE clause.

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (value1, value2, ...);

or:

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (SELECT STATEMENT);

EX:

SELECT \* FROM student WHERE id IN(1,2,3)

***SQL Aliases:***

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of the query.

SELECT column\_name AS alias\_name

FROM table\_name;

EX:

SELECT \* FROM student as s WHERE id IN(1,2,3)

## **SQL JOIN**

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

**Inner Join :**

Returns records that have matching values in both tables

SELECT column\_name(s)

FROM table1

INNER JOIN table2

ON table1.column\_name = table2.column\_name;

**Left Join :**

Returns all records from the left table, and the matched records from the right table

SELECT column\_name(s)

FROM table1

LEFT JOIN table2

ON table1.column\_name = table2.column\_name;

**Right Join :**

Returns all records from the right table, and the matched records from the left table

SELECT column\_name(s)

FROM table1

RIGHT JOIN table2

ON table1.column\_name = table2.column\_name;

**Full Outer Join :**

Returns all records when there is a match in either left or right table

SELECT column\_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column\_name = table2.column\_name

WHERE condition;

***GROUP BY:***

SELECT column\_name(s)

FROM table\_name

WHERE condition

GROUP BY column\_name(s)

ORDER BY column\_name(s);

EX:

SELECT COUNT(id), age FROM student GROUP BY age

***HAVING Clause:***

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SELECT column\_name(s)

FROM table\_name

WHERE condition

GROUP BY column\_name(s)

HAVING condition

ORDER BY column\_name(s);

The main difference between WHERE and HAVING clause comes when used together with GROUP BY clause, In that case WHERE is used to filter rows before grouping and HAVING is used to exclude records after grouping.

***Nth highest salary***

SELECT MAX(SALARY) from EMPLOYEE where SALARY NOT IN (SELECT DISTINCT TOP 1(SALARY) from EMPLOYEE order by SALARY desc)