BDD (Behavioral Driven Development

→ As of today we are experts in automation by using SW (Selenium webDriver), POM and TestNG and Java.

→ So whatever the code we have written to convert manual testcases into Automation test scripts, we can only understand that written code.

→ If any non technical Person try to understand the our automation test scripts he may not get clarity on the test scripts.

→ It means like scrum master, PO, stakeholders and management team are not able to understand the our written test script code

→ By seeing the above point we can believe we are missing the readability of our test script at management level

→ To acheive that readability, more reusability if each test step and easy maintainability with new enhancements we must follow Behavioral Driven Development Process (BDD).

→ To acheive BDD process we have multiple option in industry like cucumber, JBehave, specflow... etc. Here we selected cucumber Jars to succeed BDD approach or process.

Cucumber :-

→ Cucumber is a framework, work as a tool and comes in .jar file. The cucumber developed by using Ruby language, but most familiar with Java.

→ To know more about cucumber, to download .jar files to clarify our cucumber doubts we have official cucumber site like "https://www.to.cucumber.com"

Cucumber Points :- (Rules)

→ we are using the cucumber to acheive more readability at management level so we must write our test scripts in very unstandable language.

→ To acheive above point we must write English statements by using given keywords.

→ The Gherkin keywords will develop glue code and with glue code we develop step definition.

→ Here while developing glue code cucumber will take help of gherkin language keywords. It means every keywords of steps will generate one, one reusable methods.

→ once those reusable methods (glue code) got created then we can copy those glue code to generate step definition.

→ Inside the step definitions we can add our Selenium code.

Cucumber configuration at Project level :-

To add cucumber flavour to our project we should follow below navigation.

Go to maven Repository site

↓

Search for cucumber java keyword

↓

Click on cucumber JVM : Java from Official site

↓

Copy dependency of cucumber-Java

↓

Past in pom.xml file under dependencies

↓

Repeat same navigation for cucumber-Junit

↓

Click on save all options at Eclipse

↓

Cross check all above dependencies came in maven dependency folder

↓

Update your Project

↓

Cross check there should not be any error in our project

Cucumber plugin Configuration at Eclipse level..

→ Navigation for cucumber plugin in Eclipse -1//

now we add cucumber flavour at Project level only. But to highlight cucumber keywords in Eclipse we must tell to Eclipse we are creating cucumber feature files by adding cucumber plug in.

Click on help in Eclipse main menu

↓

Click on Eclipse market place

↓

Type cucumber word in findbar or searchbar

↓

Click on Go button

↓

Click on install at cucumber Plugin for Eclipse

↓

Select checkbox of cucumber

↓

Click on Next

↓

If untrusted kind of Popup came select checkbox of trusted

↓

Click on Install

↓

Once cucumber plugin got installed click on restart of Eclipse ↓

And observe cucumber plugins got ~~installed~~ successfully added to the Eclipse or not.

Configuration for converting the Project into the cucumber framework :-

Right click on the Project

↓

click on configure

↓

click on convert into cucumber

↓

observe project got converted into cucumber framework or not.

Create feature file for ~~Demo~~ Nopcommerce logintestas

Feature : NOPcommerce login Page

Scenario Outline : Login functionality

Given launch the NOP commerce URL

When enter valid username and password

When click on login button

Then verify login success or not

→ once we create .feature file, for every step of the feature file we must create glue code to develop stepDefinition file.

→ To generate gluecode we have two ways.

Way 1 :- Right click on the feature file and click on run as cucumber feature file. Once we run the feature file, if there is no gluecode for the steps then cucumber will generate glue code in console pan. Once glue code got generated in console pane we can copy that code and post inside stepDefinition class.

→ In stepDefinition according to the methods we can add the Selenium code with java.

Example :

Public class NOP

Way:-2

→ To generate the glue code or stepdefinition we have two ways means directly executing the feature file and generating the glue code in console and Passing inside the stepdefinitions file.

→ If we donot want to depends on the cucumber to generate glue code then we can write our own gluecode by following glue code syntax.

Example :-

Feature : login with hardcoded values

Scenario : verify login can success or not

Given launch nopcommerce login page
When enter valid username & Password
When click on login button
Then verify login success.

→ In above feature file we have four statements for each statement we must write glue code method by following below rules.

→ For each keyword statement we should write the glue code, then that keyword will become annotation for the method.

Syntax :-

Given launch nopcommerce login page

```
@Given( )
    P.V. nop_com_login_Page( ) {
    }
```

→ Our provided method if we would like to match with our corresponding statement then that statement we must declare in @Given annotation in String format.

syntax:-

Given launch nopcommerce loginpage

@Given("launch_nopcommerce_loginpage") #

Pv launch_nopcommerce-loginpage(){

}

→ If we would like to match our statement with our glue code method without any issue then we must declare the statement in annotation with ^ & $ symbol.

Here ^ & $ symbols are regular expressions, The ^ symbol will work as matching of the first character or first word.

→ It means by using ^ & $ symbols we are doing exact match of statements and glue code.

syntax:-

Given launch nopcommuce login page

@Given("^Launch nopcommerce_login page $")

Public void launch_nopcommerce_login Page(){

}

@When("^Enter valid username & Password $")

Public void Enter_username_Password(){

}

```
@When("^click on loginButton$")
Public void click_on_loginButton(){
=====
}
@Then("^verify login should success$")
Public void verify_login_should_success(){
=====
}
```

Feature file Creation rules:-

→ To create feature file of the cucumber we must give .feature extension to our feature file.

→ If we donot gives .feature extension to our file then it cannot become a feature file.

Syntax :-

→ In our .feature file only one time feature keyword is allowed

→ It means in one .feature file we can only write one feature.

Syntax :-

filename.feature

```
feature: featureName
```

→ In our feature file we can declare multiple scenarios. It means in one feature file under one feature we can have one or more than one.

Syntax :-

```
Feature : feature name
Scenario : Scenario name
```

filename.feature

→ In one .feature file we can declare at multiple Scenario outlines. It means in one .feature file under feature tag we can declare one or more Scenario outlines. for Scenario Outline we must follow below Syntax.

Syntax :-

filename.feature

```
Feature : Feature name
Scenario : Sname
examples :
```

Note :- If we are writting the Scenario Outline then we must write the examples keyword without Examples keyword if we are trying to write the Scenario Outline then we will get error.

→ To achieve Data Driven Testing we can use Scenario outline with Examples keyword in Cucumber.

→ In Examples keyword every data seperated with '|' Pipe Symbol.

Syntax :-

Feature : feature name

Scenario Outline : S.O. name

Given launch the login page

When enter "<username>" and "<Password>"

Examples :

| username | Password |

| admin@yourstore.com | admin |

| admin@store.com | admin |

→ In one .feature file under feature tag we can write combination of Scenario and Scenario Outline with Examples keyword.

```
Feature: FeatureName
ScenarioOutline: S.O.Name
Given launch the login page
Given enter "<username>" and "<Password>"
Examples :
        |username | Password |
        |admin@yourstore.com|admin|
        |admin@store.com | admin |
Scenario: scenario name
  Given ____
  When ____
  Then ____
```

→ under Scenario or Scenario Outline with Examples keyword we must write our test steps by using below keyword.

1. Given
2. When
3. Then
4. And
5. But

Given :- This Given is a keyword in Gherkin langua to work with test statement.

→ It means to write the test step that to the test step pre condition of test steps then that statement we must declared with Given keyword

Syntax / Example :-

```
Given launch Nopcommerce login Page
```

When :-

→ once precondition steps got executed then if we would like to perform some actions then those action.

→ Statements we must declared by using When keyword.

→ It means whenever we want to do some actions like click, entering data, Double click ... etc, then those kind of action statement we can declared by using When keyword.

Then :- Whenever we would like to do verification based on the actions then those verification statements we can declared by using Then keyword statements.

Syntax :- `Then verify login should success`

And :- Whenever we have repeated statements with same functionality then we can use And keyword

syntax :-
```
Scenario : Sample Scenario
    Given launch any page
    When enter any page
    And click on submit link
    Then verify next page got loaded
```

But :- Whenever we have conditional based statements then those statements we can keep inside the But keyword

Syntax :-
```
When enter un and pws
But click on submitbtn
```

Data Table in Cucumber

→ Whenever we would like to pass the data for particular statement level we can use Datatable concept.

→ If we are using the datatable then we can get the data from dataTable in two ways.

1. asLists()
2. asMaps()

Syntax 1 :- asLists level

```
List<List<String>> data = dataval.asLists();
```

List<List<String>> = Datatable.asLists();

- List → Rows
- List → Columns
- String → Celldata type

Syntax 2 :-

```
List<Map<String, String>> = Datatable.asMaps();
```

- List → Rows
- String → Column datatype
- String → Column value datatype

Example for datatable way1 :-

Scenario: nopcommerce login with single data

Given launch the Valid URL

When enter values in email field and Password field

|hello@store.com|hello123$|

And click on the login button to do login

Then verify login success

```
@When("^enter values in email field and Password field$")
Public void enter_values_email_Password(DataTable
                                        dataval){
List<List<String>> realData = dataval.asLists();
    String val1 = realData.get(0).get(0);
    String val2 = realData.get(0).get(1);
      S.o.pln(val1+" "+val2);
```

Example for datatable way2 :-

scenario: nop commerce login with single data

Given launch the valid URL

When enter values in email field and Password field

|username|Password|

|hello@store.com|hello123$|

And click on the login button

Then verify login success

```
@When("^enter values in email field and password field$")
Public Void enter_email_password(DataTable dataVal){
List<Map<String, String>> realData = dataVal.asMaps();
    String Val1 = realData.get(0).get("username");
    String val2 = realData.get(0).get("Password");
        S.O.pln(val1 + " " + val2);
```

Background Keyword :-

→Whenever we have similar duplicate statements on multiple in scenario and scenario Outline, then we can write generic statements for those duplicate statements which are utilize for all scenario's and scenario Outline.

→It means if we have repeated statements in multiple scenarios then for those all scenarioOutlin and we can remove those repeated statements and we can keep inside the Background.

→Background keyword will work as generic keyword for all scenario and scenario outline

Syntax :- Featurefile level hard coded values

```
Background: This is a background
    Given launch nopcommerce nopcommerce
    When enter valid "admin" and "admin"
    When click on login button
Scenario: verify Dashboard should Present
    Then verify dashboard Present
```

Hook's in Cucumber :-

→ Hooks in cucumber means if we have like any Preconditions and Post conditions functinlity of Scenario or Scenario Outline with Examples keyword, then we should use below.

Hooks :-

1. @Before
2. @After
3. @ Beforestep
4. @ Afterstep

@Before :-

→ If we would like to give the Precondition for all Scenarios and Scenario Outline with Example keyword then we must use the @Before annotation

→ This @Before annotation method will execute before triggering the Scenario or Scenario Outlin with Examples keyword without casing those Scenario's are exist in any feature file.

Syntax :-

```
@Before
Public Void setup(){
    S.o.Pln ("Setup");
}
```

@After :-

→ If we would like to give post condition for all Scenarios or Scenario Outline with Examples keyword then we should use the @After annotation.

→ This @After annotation method will execute after executing the Scenario or Scenario Outline with Examples keyword without caring those Scenario are exist in any feature file.

Syntax :-

```
@After
Public void tearDown(){
S.O.pln("close all");
}
```

@Beforestep :-

→ If we would like to give the Pre condition for each and every test step of Scenario or Scenario outline with Examples keyword, then we must use the @Beforestep annotation.

→ This @Beforestep annotation doesnot care on test step exist in which Scenario or Scenario Outline with Examples keyword.

Syntax :-

```
@Beforestep
P. v. Beforestep(){
S.O.pln("This is Before Step");
}
```

@After step :-

→ If we would like to give the post condition for every test step of scenario or Scenario Outline with Examples keyword then we must use the @Afterstep annotation.

→ @Afterstep annotation doesn't care about our test step in exist in which scenario or Scenario Outline with Example keyword.

Syntax :-

```
@After step
P.V. afterstep(){
S.O.Pln("this is after step");
}
```

Example feature file :-

```
Feature : Sample feature file
Scenario : Sample Scenario
  Given launch nopcommerce login page
  When Enter un and Pws
  And click on submit button
  Then verify next page got loaded.
Scenario Outline : Sample Scenario
  Given launch NOPCommerce loginPage
  When enter Un and Pws
  And Click on submit btn
  Then verify login Success
Examples:
  |username|Password|
  |admin@yourstore.com|admin|
```

Example step definition:

```
Public class sampler {
@Before
Public void setup(){
 S.O.pln("setup");
 }
@After
Public void tearDown(){
  S.O.pln();
 S.O.pln("close all");
 }
@Beforestep
 Public void beforestep(){
   S.O.pln("Before step");
 }
@Afterstep
 Public void afterstep(){
   S.O.pln("after step");
}
@Given("launch any Page")
Public void launch_any_Page(){
   S.O.pln("launch Page");
@When("enter un and Pws")
 Public void enter_un_Pws(){
       S.O.pln("enter un and Pws");
 }
```

```
@And("click on submit link")
    Public void click_on_login button(){
        S.O.pln("clickon loginbtn");
    }
@Then("verify login success")
    Public void verify_login_success(){
        s.o.pln("verify login success");
    }
```

Note :- In cucumber mainly we have four types of Hook's like @Before, @After, @Beforestep, @Afterstep.

→ In above four hooks @Before and @After are recommanded for pre-conditions and post conditions

Tags in cucumber :-

→ Whenever we would like to divide Scenario or Scenario Outlines according to the groups, Then in cucumber we can use Tag's concept.

→ These tags we must declare in .feature file only. This tags we can declared either feature level or Scenario level or Scenario Outline level.

→ The Tag names we can declare by giving any name but giving meaningful names are recommended for better reusability.

Example :-

@ END To END

Feature : Sample feature file.

@Smoke

Scenario : Sample Scenario 2

Given launch any Page

When enter un and Pws

AND click on submit button

Then Verify login success

@Regression

Scenario Outline : Sample Scenario 2

Given launch nopcommerce Page

When enter un and Pws

And click on login btn

Then Verify login Success

Examples :

| Username | Password |

| admin@yourstore.com | admin |

Note :- To execute above tags we must write the runnerClass and we must declared by name to tags keyword under cucumber options

Example

```
@Run with (cucumber.class)
@CucumberOptions(
        features = "Path file location";
         glue = {"stepdefinitions"};
         tags = {"@smoke"}
        )
Public class TestRunner{

}
```

Tagged Hooks in cucumber :-

→ Whenever we have a situation if we would like to execute particular pre & post condition of hooks according to the tags then we can use tagged hooks.

→ If we have three Pre & Post hooks but first hook need to execute only for Smoke tag and second hook need to execute only for @Regression tag and third hook need to execute only @EndToEnd tag, then according to the tags we should declare hooks by giving tag names inside hooks.

Note:- Take the above feature files as an Example for tag declarations

Stepdefinition :-

```
Public class sample1 {
@Before ("@Smoke")
Public void setupsmoke(){
    s.o.pln ("setup for smoke");
      s.o.pln ();
  }
@Before("@Regression")
  Public void setupRegression (){
    s.o.pln ("setup for Regression");
      s.o.pln ();
  }
@Before ("@ENDTOEND")
  Public void tearDownsmoke(){
    s.o.pln();
    s.o.Pln("close all for smoke");
  }
@After ("@Regression")
  Public void tearDownRegression(){
    s.o.pln();
    s.o.pln ("close all for Regression");
  }
@After ("@ENDTOEND")
  Public void tearDownEndTOEnd(){
  }  s.o.Pln ("close all for EndTOEnd");
```

Runner class in cucumber :-

→ If we want to execute multiple feature files according to the step definition then we must execute from customized Runner class
→ To write the runner class we need cucumber Junit dependency
→ Once we add the cucumber Junit dependencies to our pom.xml file and click on save then Junit and cucumber Junit jar files will loaded to our Project
→ Here Junit jar we need for executing our cucumber class in Junit format by using @Runwith annotation.
→ Here we must use as @cucumberOptions annotation to declare when, what and how to execute cucumber Project to achieve @cucumberOptions annotation
- we need cucumber junit dependencies or jar
→ In @cucumberOptions we have multiple arguments like features, glue, stepdefinitions, monochrome, Plugin, dryrun, strict, tags ... etc.

1. @RunWith:- This annotation which can help us executing cucumber class in junit format with junit dependency Reports.

Syntax

@RunWith(cucumber.class)

↓ come from Junit

↓ come from cucumber junit jar

2. Cucumber Options :- This is the annotation which can guide us to when, what, how to execute cucumber Project.

Syntax :- @CucumberOptions(____)

↓ Came from cucumber junit

↓ Came from cucumberoptions arguments.

. Feature :- Here the one of the argument in cucumberOption annotation. By using this argument we can define or declared feature files by giving feature file paths.

syntax :- @CucumberOptions (

features = "path of feature file".

)

. Glue :- This is one of the argument in @Cucumber options annotation. we can declare by using the argument we can declare stepdefinition location.

Syntax :- @CucumberOptions (

features = "Path of the feature file"

glue = {'stepdefinition'},

}

. StepNotifination :- This is one of the argument in @CucumberOptions annotation.

→ By using this argument we can display the step level reports in Junit reports.

Syntax :- `stepNotifications = true;`

6. Monochrome :- This is one of the argument in @CucumberOptions annotation. If we want to avoid special symbols and clumsy output in console then we must use monochrome argument.

Syntax :- `monochrome = true ;`

Plugin :- This is one of the argument in @CucumberOptions annotation. By using this argument we can generate inbuilt cucumber reports.

Syntax :- `Plugin = {'Pretty', "html:cucumber-report"};`

dryRun :- This is one of the argument in @cucumber options annotation. By using dryRun argument we can generate the glue code for feature files steps in console output. when if that feature steps doesn't contains stepdefinitions or glue code.

→ By using this dryRun argument we can Cross check for all test steps of the featurefile we have created stepdefinitions are not.

Syntax :- `dryRun = true`

strict :- This is one of the argument in @CucumberOptions annotation. By using this strict argument we can cross check for all test

steps of feature file, we have created stepdefinition or not.

→ This strict argument for any one of the step glue code or step definitions code is missing then it won't execute the cucumber project. It means when all stepdefinitions got created for all test steps of the feature file then only this strict argument will allowed to execute cucumber runner class.

Syntax :- `strict = true;`

tags :- This is one of the argument in @CucumberOptions annotation. By using this argument we can declared tags which are created in .featured file.

Syntax :- `tags = {"@smoke"}`

Example :-

```
import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
@RunWith(cucumber class)
@CucumberOptions(
        features = "Path of the .feature file",
        glue = {"stepdefinition"}

stepNotification = true;
```

```
monochrome = true,
Plugin = {"pretty," "html:cucumber-report"},
dryRun = true,
strict = true,
tags = {"@smoke"}
)
Public class TestRunner {

}
```

Batch execution :-

→ Whenever we would like to execute batch execution from Command prompt for maven Project then we should follow below two rules

Rule 1 :

In which location our maven project got created from that location we should open Command Prompt

Rule 2:

To execute entire maven Project from command Prompt we should use `mvn test` command.

Note :- To follow above two rules, before we should add maven-compiler-plugin in our pom.xml file.

→ To execute cucumber files from command prompt we should follow above two rules.

→ In cucumber execution if we would like to filter cucumber options then we should use below command.

```
mvn clean test -Dcucumber.filter.tags="@Smoke"
```