12/01/23

FRAME WORK

BDD (Behavioural 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 test cases into automation test scripts, we can only understand that return.

→ 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, product owner, Stake holders & management team are not able to understand the our written test script code.

→ By seeing the about 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 & Easy maintainability with new Enhancements we must follow Behavioral Driven Development Process (BDD).

→ To Achieve BDD process we have multiple options in industry like cucumber, JBehave, specflow -- etc. Here we selected cucumber Jars to successed BDD approach & process.

CUCUMBER :-

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

→ To know more about cucumber, to download .jar files to clarity our cucumber doubts we have official cucumber site like.

"https://www.io.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 understandable Language.

→ To achieve 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 (glueCode) got created then we can copy those glue code to generate step Definition.

→ Inside the step Definition's 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 JUM:Java from official site.

↓

Copy dependency of cucumber Java

↓

post 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 to cucumber plugin in Eclipse till 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 file by adding cucumber plugin.

Click on Help in Eclipse ~~market~~ main menu

↓

click on Eclipse market place

↓

Type cucumber word in findbar & searchbar

↓

click on Go button

↓

click on Install at cucumber plugin for Eclipse.

↓

Select checkbox of cucumber

↓

clion on Next

↓

Accept terms & Conditions

↓

click on next

↓

If untrusted kind of popup come select checkbox of trusted

↓

click on Install

↓

once cucumber plugin got installed click on restart of Eclipse

↓

And observe cucumber plugin got successfully added to the Eclipse & Not.

Configuration to 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 & Not.

Create feature file for nopCommerce - LoginTestCase:

Feature: Validate Nopcommerce Login page

Scenario: Nopcommerce Login Functionality.

Given Launch nopCommerce Login page

When enter valid username and password
When click on LoginButton
Then verify Login should success.

Note:-
→ once we create .feature file, for every step of the feature file we must create glue code to develop step Definition 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 gluecode in console pane. once gluecode got generated in console pane we can copy that code and past inside step definition class. In step Definition according to the methods we can add the selenium code with Java.

Example 1:-

```
public class nopCommerce_stepDefinitions {
    WebDriver driver;
@Given ("^Launch nopCommerce Login page$")
public void Launch_nopCommerce_Login_page () {
WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.get ("https://admin-demo.nopCommerce.com/Login");
```

```
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(Duration.
                    ofSeconds(30));
}
@When("I enter valid username and password $")
public void enter_valid_username_and_password()
                    throws Exception {
  Thread.sleep(7000);
driver.findElement(By.id("username")).clear();
driver.findElement(By.id("username")).sendKeys
                    ("admin@yourstore.com");
driver.findElement(By.id("password")).clear();
driver.findElement(By.id("password")).sendKeys
                    ("admin");
}
@When("click on login button")
public void click_on_login_button() throws Exception
                                        {
driver.findElement(By.xpath("//button[@type='submi
                    t']")).click();
  Thread.sleep(7000);
 }
@Then("Verify login should success")
public void Verify_login_should_success(){
  System.out.println("login should success");
}}
```

Way 2:-

→ To generate the glue code & step definition we have two ways means directly Executing the feature file and generating the glue code in console and passing inside the step definition file.

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

Example:-

Way 2 - feature

```
Feature: login with hard coded values
Scenario: Verify login can success or not
Given Launch nopcommerce login page
When enter Email and password
When click on login button
Then Verify login should success
```

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

(i) For which keyword statement we should write the glue code, then that keyword will become annotation for the method.

Syntax:-

Given Launch nopcommerce login page

@Given()
public void Launch_nopcommerce_Loginpage();

}

2) 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 Login page

@Given("Launch_nopcommerce Login page")
public void Launch_nopcommerce_Login_page(){

}

3) If we would like to match our statements with our glue code method with out any issue then we must declare the statements in annotation with ^ & $ symbol.

→ here ^ & $ symbols are regular Expressions, the ^ symbol will work as matching of the first character (&) first word.

→ here $ symbol will work as matching of the Last character (&). Last word.

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

Syntax:

Given Launch nopcommerce Login page

@Given("^Launch nopcommerce Login page$")

public void launch nopCommerce login page(){

}

→ In above 3 syntax's the last syntax means using Regular Expression is recommended and powerful syntax.

Example:—

Write stepDefinitions for way2 feature file using Regular Expression.

@Given("^Launch nopcommerce Loginpage.$")

public void launch_nopcommerce_login_page(){

}

@When("^enter valid Email and password$")

public void enter_valid_Email_and_password(){

}

@When("^click on Login button$")

public void click_on_Loginbutton(){

}

@Then("\n Verify login should success $")
public void Verify_loginShould_success(){

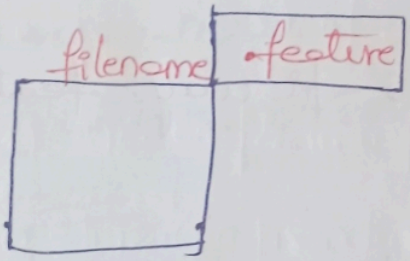
}

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 scenario's, it means in one feature file under

one feature we can have one & more than one scenario,

syntax:-

filename.feature

```
Feature: featurename
Scenario: scenarioname
```

→ In one .feature file we can delare at multiple scenario outlines. it means in one .feature file under feature tag we can declare one & more Scenario outlines. for scenario outline we must follow below syntax:

Syntax:-

filename.feature

```
Feature: FeatureName
Scenariooutline: SName
example
```

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 Example keyword in cucumber,

→ In Example keyword Every data seperated with '|' pipe symbol.

Syntax:-

filename.feature

```
Feature: featurename
Scenario outline: Sonome
Given Launch the login page
When enter "<username>" and
          "<password"
-Examples:
|username |password|
|admin@yourstore.com|admin|
|nandini@store.com|nandini123|
```

→ In one feature file under feature tag we can write Combination of scenario and scenario outline with Examples keyword.

feature: FeatureName

scenario outline: SoName

Given Lanch the Login page

Given enter "<username>" and "<passwords>"

Examples:

| username | password |

| admin@yourstore.com | admin |

| nandini@store.com | nandini |

Scenario: ScenarioName

Given _____

When _____

Then _____

b) under scenario (&) Scenario outline with Example keyword we must write our test steps by using below keywords.

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

1) Given:—

→ this Given is a keyword in Gherkin Language to work with test statement.

→ It means to write the test step that to the test step precondition 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.

Example/syntax:-

> When enter valid username and password

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 then But keyword.

Syntax:-

```
When enter un and pws
But click on submit link
```

Data table in cucumber :-

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

→ if we are using the data table then we can get the data from data table in two ways.

1) asList.
2) asMaps.

Syntax 1 :- asList level :

```
List<List<String>> = dataTable.asLists();
```

rows — columns — cells data type

Syntax 2 :- asMap Level

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

rows — Columns & Value of Column — Column data type — Column's Value data type.

Example for datatable way 1 :-

Scenario: Nop Commerce login with single data
Given Launch the Valid URL in supported browser
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 successfully functionality.

Code:—

```
@When("^enter values in email field and
                    password field $")
public void enter_values_in_email_field_and_
        password_field(DataTable dataVal){
List<List<String>> realData = dataVal.asLists();
   String Val1 = realData.get(0).get(0);
   String Val2 = realData.get(0).get(1);
System.out.println(Val1 + "   " + Val2);
```

Example for data table way 2:—

Scenario: Nop Commerce login with single data
Given Launch the valid URL in supported browser
When enter values in email field and password field

UserName	password
hello@store.com	hello123$

And click on the login button to do login
Then verify login successfully functionality.

Ex:—

```
@When("^enter values in email field and password
                    field $")
```

```
Public void enter_values_in_email_field_and_
                password_field(DataTable dataVal){
List<Map<String, String>> realData = dataVal.
                                    asMaps();
String Val1 = realData.get(0).get("userName");
String Val2 = realData.get(0).get("password");
System.out.println(Val1+"  "+Val2);
}
```

Background keyword:-

→ Whenever we have similar duplicate statements or multiple in scenario and scenario outlines then we can write generic statements to those duplicate statements which are utilize to all scenario's and scenario outline. It means if we have repeated statements in multiple scenario's then to those all scenario outline and we can remove those repeated statements and we can keep inside the Background.

→ Background keyword will work as Generic keyword to all scenario and scenario outlines.

Syntax:-

Feature: Feature file level hard coded values

```
Background: This is a Background
```

Given Launch nopcommerce login page
When enter valid "admin@yourstore.com" and

"admin"

When click on Loginbutton

Scenario: Verify MyProfo should present

Then Verify MyProfo should present

Hook's in Cucumber:-

→ Hook's in Cucumber means if we have like any preconditions and post-conditions functionality of scenario (or) Scenario outline with Example keyword, then we should use below.

Hook's:-

(1) @Before

(2) @After

(3) @BeforeStep

(4) @AfterStep

(1) @Before:-

→ If we would like to give the pre-condition to all Scenario's 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 outline with Example keyword without Caring those Scenario's are exist in any feature file.

Syntax:-

```
@Before
public void setup(){
System.out.println("setup");
}
```

@After:-

→ If we would like to give post condition for all scenario's (or) scenario outline's with Examples keyword then we should use the @After annotation.

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

Syntax:-

```
@After
public void teardown(){
System.out.println("close all");
}
```

3) @BeforeStep:-

→ If we would like to give the pre-condition for each & every test step of scenario or scenario outlines with Example keyword, then we must

use the @Beforestep annotation.

→ This @Beforestep annotation doesn't care our teststep exist in which scenario & scenario outline with Example keyword.

syntax:-

```
@Beforestep
public void Beforestep() {
System.out.println("this is Beforestep");
}
```

4) @Afterstep:-

→ If we would like to give the post-condition for every teststep of scenario (&) scenario outline with Examples keyword then we must use the @Afterstep annotation.

→ @Afterstep annotation doesn't care about our teststep is exist in which scenario (&) scenario outline with Example keyword.

Syntax:-

```
@Afterstep
public void afterstep(){
System.out.println("This is afterstep");
}
```

Example feature file:-

```
Feature: Sample feature file
Scenario: Sample scenario1
  Given Launch any page
  When enter un and pws
  AND click on submitbutton
  Then verify next page got loaded.
Scenario outline: Sample Scenario 2
    Given Launch anypage
    When enter un and pws
    And click on submitlink
    Then verify next page loaded
  Examples:
        |username          | password |
        |admin@yourstore.com| admin    |
```

Example for stepDefinition:-

```
public class Sample1 {
@Before
 public void setup(){
 System.out.println("setup");
}
@After
 public void teardown(){
 System.out.println();
 System.out.println("closeall");
}
```

```
@BeforeStep
 public void beforestep(){
 System.out.println("Before step");
}
@Afterstep
 public void afterstep(){
 System.out.println("afterstep");
}
@Given("launch any page")
 public void launch_any_page(){
 System.out.println("launch_any_page");
}
@When("enter un and pws")
 public void enter_un_and_pws(){
 System.out.println("enter un_and_pws");
}
@AND("click on submitbutton")
 public void click_on_submit_button(){
 system.out.println("click on submit button");
}
@Then("verify next page got loaded")
 public void verify_nextpage_got_loaded(){
 System.out.println("verify next page got loaded");
}
```

Note:—

→ In cucumber mainly we have four types of Hooks

like @Before @After, @Beforestep, @Afterstep.
* In above four hook's @Before and @After are recommended for pre-conditions and post-conditions.

Tags in Cucumber:-

* In Cucumber mainly we have four types of Hooks
* 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 declared by giving any name but giving meaning full 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 next page got loaded

@Regression

Scenario outline: Sample Scenario 2
Given Launch any page
When enter un and pws
AND click on submit button
Then Verify next page got loaded

-Examples:

username	password
admin@yourstore.com	admin

Note:-
-> To execute above tags we must write the Runner class and we must declared by name to tag's keyword under cucumber option's.

Example/Syntax:-

```
@Run with(cucumber.class)
@cucumberoptions(
        features = "path file location",
        glue = {" step Definition's "},
        tags = {"@smoke"}
)
public class Test Runner{
}
```

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(or) 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 declaration.

StepDefinition:-

```
public class Sample1 {
@Before("@Smoke")
public void setupSmoke(){
 System.out.println("setup for Smoke");
 System.out.println();
}
@Before("@Regression")
public void setupRegression(){
System.out.println("setup for Regression");
 System.out.println();
}
```

```
@Before("@ENDTOEND")
public void setupENDTOEND(){
System.out.println("setup for ENDTOEND");
System.out.println();
}
@After("@smoke")
public void tearDownSmoke(){
System.out.println();
System.out.println("close all for smoke");
}
@After("@Regression")
public void tearDownRegression(){
System.out.println();
System.out.println("close all for Regression");
}
@After("@ENDTOEND")
public void tearDownEndTOEnd(){
System.out.println();
System.out.println("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 to executing our cucumber class in Junit format by using @Run with annotation.

→ Here we must use an @cucumberoption annotation to declare when, what and how to execute cucumber project. To achieve @cucumberoption 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 helps us executing cucumber class in Junit format with Junit reports.

Syntax:— @Runwith (Cucumber.class)

↓ Come from Junit

↓ Come from cucumber Jar.

2) cucumber options:—

→ This is the annotation which can ~~helps us executing~~ guide us to when, what, how to execute cucumber project.

Syntax:-

@cucumberoptions(_____)

↓ Come from cucumber junit

↓ Come from cucumber options arguments.

3) Feature:-

→ Here the one of the argument is cucumberoption annotation. By using this argument we can define (or) declared feature files by giving feature filespaths.

Syntax:-

```
@cucumberoptions(
        features = "path of featurefile",
)
```

4) Glue:-

→ this is one of the argument in @cucumber options annotation we can declare by using this argument we can declare stepdefinition location.

Syntax:-

```
@cucumber options (
    features: "path of the feature files"
        glue = {"stepdefinition'},
)
```

5) stepNotification:-

→ This is one of the argument in @cucumberoptions annotation.

→ By using this argument we can displaying 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 clumgy 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 this dryRun argument we can generate the glue code for feature files steps

in Console output. when if that feature steps don't contains step definitions (or) glue code.

→ By using this dryRun argument we can crosscheck for all test steps of the feature files. we have created stepDefinition's are not.

Syntax:-

```
dryRun = true
```

Strict:-

→ this is one of the arguments in @cucumberoption Annotation. By using this Strict argument we can cross check for all test steps of features file, we have created stepDefinitions or not.

→ This strict argument for any one of the step glue code (or) stepDefinition 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 .feature 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 Bat 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 shall use

Command → `Mvn test`

Note:-

→ To follow above 2 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 only.

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

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