[**https://github.com/pavanoltraining**](https://github.com/pavanoltraining)

**https://github.com/PramodDutta**

For practice: https://www.nopcommerce.com/en/features

=================== Cucumber By Testing world ===================

1. how to run cucumber scenarios in parrallel ?

A. we can run cucumber scenarios in parrallel by overriding the dataProvider parrallel mode to true inside the testRunner class.

Cucumber doc: https://cucumber.io/docs/guides/parallel-execution/?lang=java

File extension would be - **.feature**

The language used in Cucumber is **Gerkin**.

Common keywords used in **Gerkin**…..

**Feature:**

**Scenario:**

**Given, When, Then, And, But(Steps)**

**Background:**

**Scenario Outline:**

**Example:**

**“”” “”” document String**

**| data table**

**@ tags**

Keyword: **Feature** – defines End to End Scenario or application feature.

Syntax –

**Features**: name of the Feature.

Simple description (could be multiple line)

Keyword: **Scenario** – defines as a single test case of the feature.

Can have multiple Scenario(test case) under one feature.

Syntax –

**Scenario**: name of the Scenario

Steps of the test cases . . . . . .

Keyword: **Given** – defines initial point (connection DB, start Browser, put URL etc). can be multiple time.

Keyword: **When** – used to write actions we want to perform.

Keyword: **Then** – used for validation, to match actual result with expected result.

Keyword: **And** – used with When, Then for adding more action for When & Then keyword.

Keyword: **Background** – used once at the start instead reapiting in every Scenario.

**Step arguments:** used to pass data to the step.

syntax – When user enter uid “data” //so “data” is step arguments.

Keyword: **Scenario Outline** – alternative of Scenario keyword, When we want to execute same test case with multiple sets of data.

syntax – When user enter uid “<username>” ( used parameter instead raw data.)

When user enter password “<password>”

Keyword: **Example** – used for data table

Syntax –

**Example:**

**|username | password|**

**| user1 | pass1 |**

**|user2 | pass2 |**

Keyword: **#** used for comment

Keyword: **“””** **“””** (doc string) used to write larger text in text area….

Sysntax –

**And** user click on compose button in email page

**And** user enter data to “[abc@gmail.com](mailto:abc@gmail.com)” enter body text

“””

Text line 1

Text line 2

So on …

“””

Feature files examples:

------------------------------------------

Feature: Login Functionality

As a user

I want to be able to log in to my account

So that I can access my profile and settings

Scenario: Valid Login

Given I am on the login page

When I enter valid credentials

And I click on the login button

Then I should be logged in to my account

Scenario: Invalid Login

Given I am on the login page

When I enter invalid credentials

And I click on the login button

Then I should see an error message

------------------------------------------------------

Scenario Outline: Check login functionality

Given I am on the login page

When I enter "<username>" and "<password>"

Then I should be logged in successfully

Examples:

| username | password |

| Anshita | 123456 |

| Alice | qwerty |

-----------------------------------------------------------

explanation of above Scenario:

Scenario: Check login functionality for multiple users

Given I am on the login page

When I enter "Anshita" and "123456"

Then I should be logged in successfully

When I enter "Alice" and "qwerty"

Then I should be logged in successfully

-----------------------------------------------------------

Keyword: **@ tags** used to control the execution of our test Scenario.

We can do one or more Tag over the Scenario, Scenario Outline, or Feature.

The Tag will be Executed which one is called….

Syntax –

**@** End to End regresstion test

**Feature**: name of the feature

**@** Smoke test

**Scenario**: name of the Scenario

**@** Sanity test

**Scenario**: name the Scenario

**@** Smoke test, @Sanity test

**Scenario**: name of the Scenario

**@** Smoke test

**Scenario** **Outline**: name of the Scenario Outline

**Step definition** – what ever steps/actions(actual code) is written for the steps in Feature file is called step definition.

**Complete syntax** . –

**Scenario Outline**: verify login function

**Given** user open browser and enter url and navigate to login page.

**When** user enter uid “<username>”

**And** user enter password “<password>”

**And** user click on login button

**Then** user shuld be logged in

**Examples:  
 | username | password |  
 | user1 | pass1 |  
 | user2 | pass2 |  
 | user3 | pass3 |**

**Complete syntax for Background. –**

**Feature**: name of the feature

**Background**:   
 **Given** user open browser and enter url and navigate to login page.

**Scenario**: verify login function

**When** user enter uid “admin”  
 **And** user enter password “demo1”

**And** user click on login button

**Then** user should be logged in

Complete syntax . –

**Scenario Outline**: verify login function

**Given** user open browser and enter url and navigate to login page.

**When** user enter uid “<username>”  
**And** user enter password “<password>”

**And** user click on login button

**Then** user should be logged in

**Examples:  
 | username | password |  
 | user1 | pass1 |  
 | user2 | pass2 |  
 | user3 | pass3 |**

**End to End BDD framework: Selenium - Java – cucumber**

**# first we need to create .feature file (BDD test cases) then implement with the selenium code and create test runner class to execute test.**

Step 1:   
(a) Create Maven Project:   
(b) Add dependencies: : Cucumber-java, Cucumber-Junit, Junit, Extent Reports, ExtentReports adapter,  
Cucumber-extentReports, Selenium-java.  
(c) Download plugins: Cucumber Eclipse Plugin, Natural, maven-compiler, maven-surefire plugin.

Step 2: Design Project Structure  
(a) Create Folder Structure: feature, reports, locators, config, screenshots.

(b) Package Structure:  
**step\_definition:** actual code for step will be here. (com.automation.step\_definiton)  
**base** : Browser related code. (com.automation.base)  
**Page**: POM (com.automation.page)  
**utilities:** read config files, read locators. (com.automation.utility)  
**runner:** Execution starting point. (com.automation.runner)  
**assertions:** Comparison (com.automation.assertions)  
**actions:** code for performing user activties. (com.automation.action)

Step 3: Design Feature files //(right click in feature file > pretty format. (for formatting))

Write test cases in Gerkin language  
Scenario 1: Login & logout  
Scenario 2: Registration

right click on feature file folder>new>file>name(Login\_logout.feature)>finish.  
open feature file and start writing BDD test case.

Feature: registered and non registerd user come on application and perform activites

Scenario: registered user comes to application and login & logout  
Given user is on login page  
When user enters username  
And user enters password  
And user clicks on submit button  
Then user should be logged in  
When user clicks on logout button  
Then user should be logged out

Scenario: non registered user comes on application and register and login  
Given user on registration page clicks on register  
When user enters username  
And user enters email address  
And user enters password  
And user enters confirm password  
And user enters date of birth  
And user enters phone number  
And user enters address   
And user click on Home address type  
And user select Gender   
And user select Country  
And user select State  
And user select City  
And user enter zip code  
And user click on agree check box  
And user click on sign up button  
Then user account should be created

Step 4: Setup Runner file for Execution

Create Runner file with different options below.

(a) features – for defining where your features file is placed.

(b) glue – for defining the packages where our implementation code written.

(c) dryrun – to check missing methods which is not coded yet for feature steps

(d) monochrome – removing junk characters

(e) plugin – for default cucumber report

Eclipse project

>package(runner)>create a class(test\_runner) NB: runner class must have the word ‘test’.  
> add annotation @Runwith(Cucumber.class)  
> @Cucumberoptions(features=”featureFile”, monochrome=true)  
> execute once then copy the structure from console.  
> go step\_definitions(package)>create a class(StepDefinition)>paste the structure methods>remove comments.  
> add dryrun & glue to

@Cucumberoptions(features=”featueFile”,

monochrome=true,

dryrun=true, //dryrun is used for checking if any feature method missing.

glue=”com.automation.step\_definition”)

> go Test\_runner class>execute again (to see if any feature method missing)  
> go stepDefiniton class>implement all methods  
> go Test\_runner class>make dryrun=false>execute

> for reporting: add Extent config to project (create a config file in reports folder)  
> right click on reports folder>new>file>name(extent-config.xml)

@Runwith(Cucumber.class)  
@Cucumberoptoins(

{features = ”featureFile”,  
 monochrome = true,  
 dryrun = true/false,  
 glue = ”com.automation.step\_definition”

}  
 plugin = {

“pretty”,

"html:target/cucumber-report/report.html",

"json:target/cucumber-json/report.json",

“pretty:target/cucumber-pretty.txt",

"junit:target/cucumber-results.xml",  
 “com.vimalselvam.cucumber.listener.ExtentCucumberFormetter”

}

plugin = true

: cucumber will generate a online report url.

NB. But First Remove unnecessary texts from url.

)

Note# Runner class name must have the word ‘ Test ‘. Only then maven will execute it.

public class TestRunner{

@BeforeClass  
public static void setReportConfiguration()  
{  
ExtentProperties property = ExtentProperties.INSTANCE();  
property.setPropertyPath(“./Reports/reports.html”);

}

@AfterClass  
public static void setConfiguration()  
{

Reporter.loadXMLConfig(“./  
}

}   
> now execute>run as Eclipse Cucumber > copy structures from console

🡸🡸🡸🡸 **Cucumber BDD framework by Raghav** 🡺 🡺🡺🡺

**To keep packages and folders organize :**

**1. Create a main folder (Features) > create a sub folder(demoFeature) > create a feature file(demo.feature) under sub folder.**

**=write in features = “main/sub folder name”**

**2. Create a package main (StepDefinitons) > create a sub Package(demoStepDefinition) :**

**= right click on package > new > package > name(main.sub) > finish > create StepDefinitionClass(Steps) under sub Package.**

**= in glue {“main/subPackageName”}**

**We need it because in cucumber we can not specify class name.**

**We can only specify package.**

**3. if any feature doesn’t run then delete the feature from the directory ( C: drive).**

Step 1 - Create a maven project in eclipse.

Step 2 - Add dependency (Cucumber Java | Cucumber JUnit | JUnit | Selenium Java

Step 3 - Download **cucumber eclipse plugin** from Eclipse Marketplace

Step 4 - Create a folder **Features** under src/test/resources or under project.

Step 5 - Under **Features** folder create a new feature file **login.feature.**

Step 6 - Create feature file and add contents Feature, Scenario, Steps, Scenario Outline, Example, Tags, Comments …. **One feature focuses on only one functonality.**

Step 7 - Try to run the **feature file**. Then copy **step definations** from console.

Step 8 - Add Step Definitions / Glue Code under src/test/java package (manually or copy/paste).

i. Create a package ii. create a class iii. paste **step definations** in the class.

Implement **step definitions** methods with **selenium code**.

Step 9 - Create a Runner class

import org.junit.runner.RunWith;

import io.cucumber.junit.Cucumber;

import io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.class)

@CucumberOptions(

features=" Features/feature",

glue={"StepDefinitionsPack/subStepPack"}

)

**public** **class** TestRunner {

}

// monochrome = true; for formatted reports.

monochrome = true

plugin = { "pretty", "html:target/reports"}

plugin = { "pretty", "json:target/reports/cucumber.json"}

plugin = { "pretty", "junit:target/reports/cucumber.xml"}

tags="@smoketest"

Step 10 - Add Cucumber Options for generating reports

HTML | JSON | JUNIT | XML

Step 4 - Add Examples section

Step 5 - Add data for the parameters in the Examples section

Feature: Domino's login page with multiple data

Scenario Outline: Login with valid data

Given User on login Page

And user clicks on sign on button

When enter valid username “<userName>”

And enter valid password “<passWord>”

And click on keep signed in button

Then user should be logged in

Examples:

| userName | passWord |

| user\_name1 | passWord1 |

| user\_name2 | passWord2 |

| user\_name3 | passWord3 |

Step 6 - Update step definition to get values from feature file Examples section.

**public** **class** DataDrivenSteps {

WebDriver driver = **null**;

@Given("User on login Page")

**public** **void** user\_on\_login\_page() {

WebDriverManager.*chromedriver*().setup();

driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.***SECONDS***);

driver.get("https://www.dominos.com/");

}

@And("user clicks on sign on button")

**public** **void** user\_clicks\_on\_sign\_on\_button() {

driver.findElement(By.*xpath*("//button[normalize-space()='Sign in & earn rewards']")).click();

}

@When("^user enters valid\_uid (.\*)$")

**public** **void** user\_enters\_valid\_user\_name(String userName) {

driver.findElement(By.*xpath*("//input[@id='Email']")).sendKeys(userName);

}

@And("^user enters valid\_pass (.\*)$")

**public** **void** user\_enters\_valid\_pass\_word(String passWord) {

driver.findElement(By.*xpath*("//input[@id='Password']")).sendKeys(passWord);

}

@And("user clicks on keep signed in button")

**public** **void** user\_clicks\_on\_keep\_signed\_in\_button() {

driver.findElement(By.*xpath*("//button[normalize-space()='Keep me signed in']")).click();

}

@Then("user should be logged in")

**public** **void** user\_should\_be\_logged\_in() {

System.***out***.println("user logged in successfully");

//driver.close();

}

}

Step 7 - Run and verify

Step 10 - Add Cucumber Options for generating reports

HTML | JSON | JUNIT | XML

Step 11 - Run and verify results

What is POM

Design pattern to create Object Repository

A class is created for each page to identify web elements of that page

Also contains methods to do action on the objects Separates test objects and test scripts

Step 1 - Create a class for each page

Step 2 - Create locators of all objects to be used in that page

Step 3 - Create methods or actions to be performed on the objects

Step 4 - Refer in the test scripts

Step 5 - Run and validate

**public** **class** LoginPage {

WebDriver driver = **null**;

**public** LoginPage(WebDriver driver) {

**this**.driver = driver;

}

By signinButton = By.*xpath*("//button[text()='Sign in]");

By userName = By.*xpath*("//input[@id='Email']");

By passWord = By.*xpath*("//input[@id='Password']");

By submitButton = By.*xpath*("//button[normalize-space()='Keep me signed in']");

**public** **void** click\_signin\_Button() {

driver.findElement(signinButton).click();

System.***out***.println("clicked signin button");

}

**public** **void** typeUserName(String userName) {

driver.findElement(userName).sendKeys(userName);

System.***out***.println("entered userName");

}

**public** **void** typePassword(String passWord) {

driver.findElement(passWord).sendKeys (passWord);

System.***out***.println("entered password");

}

**public** **void** click\_submit\_Button() {

driver.findElement(submitButton).click();

System.***out***.println("clicked submit button");

}

}

---------------------------------------------------------------------------------------------------------

**public** **class** LoginPageTestWithCucumber {

WebDriver driver = **null**;

LoginPage login = new LoginPage(driver);

@Given("User on login Page")

**public** **void** user\_on\_login\_page() {

WebDriverManager.*chromedriver*().setup();

driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.***SECONDS***);

System.***out***.println("browser opened");

driver.get("https://www.dominos.com/");

System.***out***.println("domino loginpage opened");

}

@And("user clicks on sign on button")

**public** **void** user\_clicks\_on\_sign\_on\_button() {

login. click\_signin\_Button();

}

@When("^user enters valid\_uid (.\*)$")

**public** **void** user\_enters\_valid\_user\_name(String userName) {

login. typeUserName(userName);

}

@And("^user enters valid\_pass (.\*)$")

**public** **void** user\_enters\_valid\_pass\_word(String passWord) {

login. typePassword(passWord);

}

@And("user clicks on keep signed in button")

**public** **void** user\_clicks\_on\_keep\_signed\_in\_button() {

login. **public** **void** click\_submit\_Button();

}

@Then("user should be logged in")

**public** **void** user\_should\_be\_logged\_in() {

System.***out***.println("user logged in successfully");

//driver.close();

}

}

**Demo How to implement Page Factory Model**

Step 1 - Create a class for each page

Step 2 - Create locators of all objects to be used in that page using @FindBy

Step 3 - Create methods or actions to be performed on the objects

Step 4 - Create constructor to get driver instance and initialize Elements using method initElements

public LoginPage\_PF(WebDriver driver) {

this.driver = driver;

PageFactory.initElements(driver, LoginPage\_PF.class);

}

Step 5 - Update Test Scripts to refer methods from PageFactory class

Step 6 - Run and validate

**public** **class** LoginPageFactory {

WebDriver driver = **null**;

**public** LoginPageFactory(WebDriver driver) {

**this**.driver = driver;

PageFactory.*initElements*(driver, **this**);

}

@FindBy(xpath = "//button[text()='Sign in]")

WebElement signinButton;

@FindBy(xpath = "//input[@id='Email']")

WebElement userName;

@FindBy(xpath = "//input[@id='Password']")

WebElement password;

@FindBy(xpath = "//button[normalize-space()='Keep me signed in']")

WebElement submitButton;

**public** **void** clickSignButton() {

signinButton.click();

}

**public** **void** typeUserName() {

userName.sendKeys("agdds");

}

**public** **void** typePassword() {

password.sendKeys("asdgga");

}

**public** **void** clickSubmitButton() {

submitButton.click();

}

}

-----------------------------------------------------------

**public** **class** LoginPageFactoryTestWithCucumber {

WebDriver driver = **null**;

LoginPageFactory login = new LoginPage(driver);

@Given("User on login Page")

**public** **void** user\_on\_login\_page() {

WebDriverManager.*chromedriver*().setup();

driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.***SECONDS***);

System.***out***.println("browser opened");

driver.get("https://www.dominos.com/");

System.***out***.println("domino loginpage opened");

}

@And("user clicks on sign on button")

**public** **void** user\_clicks\_on\_sign\_on\_button() {

login. click\_signin\_Button();

}

@When("^user enters valid\_uid (.\*)$")

**public** **void** user\_enters\_valid\_user\_name(String userName) {

login. typeUserName(String userName);

}

@And("^user enters valid\_pass (.\*)$")

**public** **void** user\_enters\_valid\_pass\_word(String passWord) {

login. typePassword(String passWord);

}

@And("user clicks on keep signed in button")

**public** **void** user\_clicks\_on\_keep\_signed\_in\_button() {

login. **public** **void** click\_submit\_Button();

}

@Then("user should be logged in")

**public** **void** user\_should\_be\_logged\_in() {

System.***out***.println("user logged in successfully");

//driver.close();

}

}

Today we will learn:

**1 - What are Tags in cucumber**

2 - How to use Tags (Demo)

3 - Useful Tips What are Tags in Cucumber

Features and Scenarios can be marked with Tags

Tags use @ symbol with some text e.g. @SmokeTest

In the test runner we can run specific tags

A feature or scenario can have multiple tags

@smoke @regression @important

Feature: Verify login

Can run with single OR multiple Tags

Can run with a combination of tags or using AND, OR conditions

Can skip scenarios having specific Tag

Step 1 - Create a new or use an existing Feature File

Step 2 - Mark the feature and scenarios with Tags : @TagName

Step 3 - Create new or use an existing TestRunner class

Step 4 - Add the tags in CucumberOptions section

@All

Feature: test Run With Tags

@Smoke

Scenario: Test sample 1

Given

When

Then

@Regression

Scenario: Test sample 2

Given

When

Then

@SmokeTest @RegressionTest

Scenario: Test sample 3

Given

When

Then

@Smoke

Scenario: Test sample 4

Given

When

Then

**package** runner;

**import** org.junit.runner.RunWith;

**import** io.cucumber.junit.Cucumber;

**import** io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.**class**)

@CucumberOptions(features = "featureFiles",

glue = {"stepDefinition"},

// used above 'feature' keyword to run all scenarios.

tags = "@All"

// will run all Smoke scenarios also combined with regression.

tags = "@Smoke"

//all tags scenarios will run doesn’t matter any one passed or not.

tags = "@Smoke or @Regression"

// will run combined tag scenarios only

tags = "@SmokeTest and @RegressionTest"

// will run all smoke except regression

tags = "@SmokeTest and not @RegressionTest"

// will run all un-tag scenarios

tags = "not @SmokeTest and not @RegressionTest"

// will run all scenarios except regression

tags = "not @RegressionTest"

tags = "@important or @Smoke and @Regression"

)

**public** **class** TagRaunner {

}

**: Tips- Tags can be placed above the following Gherkin elements**

1. Feature

2. Scenario

3. Scenario Outline

4. Examples

**-not possible to place tags above ‘Background’ or steps(Given,When,And,Then & But)**

Useful Tips –

Tags Inheritance

Tags are inherited by child elements.

Tags that are placed above a Feature will be inherited by Scenario, Scenario Outline, or Examples. Tags that are placed above a Scenario Outline will be inherited by Examples

Useful Tips –

Execution with Tags You can create and keep ready multiple TestRunner classes with different combination of Tags

OR

Create commands with tags combination as required to be run from the command line mvn test -cucumber.filter.tags="@smoke and @fast"

**Hooks:**

What are HOOKS ? // @Before, @After

Blocks of code that runs before OR after each scenario

Hooks in Cucumber are like Listeners in TestNG

Conditional Hooks – hooks are associated with tags for conditional execution

If we specify tag name with hook then will run only for tag.

// @Before(“smoke”) now only smoke tag scenario will run.

Why to use HOOKS - To manage the setup and teardown

To avoid re-writing the common setup or teardown actions

Allow better management of code workflow

When to use HOOKS

Whenever you have some common setup and teardown actions to be executed before each scenario

How to use HOOKS

Step 1 - Create a new or use an existing Feature File

Step 2 - Create the steps for the scenario in the feature file

Step 3 - Create setup and teardown methods and mark with annotation.

: Hooks can be created in stepDefiniton class. or

: Can be created in separate Hooks class then add the Hooks class package name in glue.

: glue = {“stepDefinitonPackage”, “hooksClassPackage”}

@Before : runs before each scenario

@After : runs after each scenario

@BeforeSteps : runs before each step

@AfterSteps : runs after each step

Step 4 - Create a TestRunner class

Step 5 - Run the TestRunner class and check execution

We can use Tags with Hooks - Conditional Hooks

Hooks can be conditionally selected for execution based on the tags of the scenario

To run a particular hook only for certain scenarios, you can associate a Before or After hook with a tag expression

Tags can be used with

@BeforeSteps

@AfterSteps

@After(value="@smoke", order=2) // order works like priority.

Single tag = {"@smoke"}

Multiple tags Tags with AND OR conditions

tags = {"@smoke or @regression"}

tags = {"@smoke and @regression"}

tags = {"@smoke and not @regression"}

Skip or Ignore Tags

tags = {"(@smoke or @regression) and not @important"}

Ordering Hooks

We can use multiple Before and After hooks and also assign order of execution

@Before(order=0)

@Before(order=1)

Background

Whatever happens in hooks is invisible to people who only read the features

Only use hooks for low-level logic such as starting a browser or deleting data from a database.

You should consider using a background as a more explicit alternative,

especially if the setup should be readable by non-technical people

--------------------------------------------------------------------------------------

Feature: Hooks of my feature

Scenario: Hooks of my scenario

Given I want to write a steps for Hooks

When I complete action hooks

Then I validate the outcomes fjor hooks

---------------------------------------------------------

**package** runner.hooksRunner;

@RunWith(Cucumber.**class**)

@CucumberOptions(features = "src/test/resources/Features/HooksFeature/Hooks.feature",

glue = "stepDefinitions/stepDefinition")

**public** **class** TestRunner {

}

**package** stepDefinitions.stepDefinition;

**public** **class** HooksSteps {

//@Before(order=1) // order will work like priority.

//@Before("@smoke") or //@Before(value="@smoke", order=1)

@Before // this Hook will run before every scenario.

**public** **void** startBrowser() {

System.***out***.println("Browser started");

}

//@After(order=1) // at after annotation order will work reversely.

@After // this Hook will run after every scenario.

**public** **void** quitBrowser() {

System.***out***.println("driver quited/closed the browser");

}

@BeforeStep // this Hook will run before every step.

**public** **void** before\_step() {

System.***out***.println("I come before every step");

}

@AfterStep // this Hook will run after every step.

**public** **void** after\_step() {

System.***out***.println("I come after every step");

}

@Given("I want to write a steps for Hooks")

**public** **void** i\_want\_to\_write\_a\_steps\_for\_hooks() {

System.***out***.println("I am in Given");

}

@When("I complete action hooks")

**public** **void** i\_complete\_action\_hooks() {

System.***out***.println("I am in When");

}

@Then("I validate the outcomes for hooks")

**public** **void** i\_validate\_the\_outcomes\_for\_hooks() {

System.***out***.println("I am in Then");

}

}

====================================================================================

What is Background?

A Step or a group of steps that are common to all the scenarios in a feature Is defined once in the feature Runs before every scenario of the feature

Why use Background?

To avoid repeating the common steps in every scenario For better readability & maintenance Unlike hooks, background is visible to the readers of the feature file

When to use Background?

Whenever there are common repeating steps in a feature

When you want the common steps to be visible to the readers

DEMO

How to use Background

Step 1 - Create a new or use an existing Feature File

Step 2 - Create the steps for the scenario in the feature file

Step 3 - Create a TestRunner class

Step 4 - Run the test runner file

Step 5 - Create a background section and add common steps

Step 6 - Run the test runner file and check execution of the background

…………………………………

Feature: Login to Account

Background: will run before every scenario

Given user open brower and go to url

Scenario: login with valid data

#Given user open brower and go to url

When user enter valid userName and Password

Then user clicked on loginButton

Scenario: login with invalid data

#Given user open brower and go to url

When user enter invalid userName and Password

Then user clicked on logButton

…………………………………………….

**package** stepDefinitions.loginStep;

**import** io.cucumber.java.en.\*;

**public** **class** Steps {

@Given("user open brower and go to url")

**public** **void** user\_open\_brower\_and\_go\_to\_url() {

System.***out***.println("I am in Given");

}

@When("user enter valid userName and Password")

**public** **void** user\_enter\_valid\_user\_name\_and\_password() {

System.***out***.println("I am in when");

}

@Then("user clicked on loginButton")

**public** **void** user\_clicked\_on\_login\_button() {

System.***out***.println("I am in then");

}

@When("user enter invalid userName and Password")

**public** **void** user\_enter\_invalid\_user\_name\_and\_password() {

System.***out***.println("I am in when2");

}

@Then("user clicked on logButton")

**public** **void** user\_clicked\_on\_log\_button() {

System.***out***.println("I am in then2");

}

}

………………………………………

**package** runner;

**import** org.junit.runner.RunWith;

**import** io.cucumber.junit.Cucumber;

**import** io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = "Features/Login.feature",

glue = "stepDefinitions/loginStep",

monochrome=**true**

)

**public** **class** LoginTestRunner {

}

………………………………………………………………….

**Cucummber with Extent Report**

**url: https://mvnrepository.com/artifact/tech.grasshopper/extentreports-cucumber7-adapter**

Step 1. Add depencies :

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

<junit.version>4.13.1</junit.version>

<cucumber.version>6.9.0</cucumber.version>

<maven.compiler.version>3.8.1</maven.compiler.version>

<maven.surefire.version>2.22.2</maven.surefire.version>

</properties>

<dependencies>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-java</artifactId>

<version>${cucumber.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-junit</artifactId>

<version>${cucumber.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>tech.grasshopper</groupId>

<artifactId>extentreports-cucumber6-adapter</artifactId>

<version>2.8.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-testng</artifactId>

<version>${cucumber.version}</version>

<scope>test</scope>

</dependency>

</dependecies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>${maven.compiler.version}</version>

<configuration>

<encoding>UTF-8</encoding>

<source>${java.version}</source>

<target>${java.version}</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>${maven.surefire.version}</version>

<configuration>

<includes>

<include>\*\*/ParallelRun.java</include>

</includes>

<parallel>methods</parallel>

<threadCount>4</threadCount>

<useUnlimitedThreads>false</useUnlimitedThreads>

</configuration>

</plugin>

</plugins>

</build>

</project>

Step 2. Create 3 files:

(i) cucumber.properties. inside file [cucumber.publish.enabled = true]

(ii) extent.properties.

(iii) extent-config.xml

(ii) extent.properties.

extent.reporter.spark.start=true

extent.reporter.spark.out=test-output/SparkReport/Spark.html

extent.reporter.spark.config=src/test/resources/extent-config.xml

extent.reporter.spark.out=test-output/SparkReport/

screenshot.dir=test-output/

screenshot.rel.path=../

extent.reporter.pdf.start=true

extent.reporter.pdf.out=test output/PdfReport/ExtentPdf.pdf

#basefolder.name=reports

#basefolder.datetimepattern=dd-MM-YY HH-mm-ss

extent.reporter.spark.vieworder=dashboard,test,category,exception,author,device,log

systeminfo.os=Mac

systeminfo.user=Naveen

systeminfo.build=1.1

systeminfo.AppName=AutomationPractice

(iii) extent-config.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<extentreports>

<configuration>

<!-- report theme -->

<!-- standard, dark -->

<theme>dark</theme>

<!-- document encoding -->

<!-- defaults to UTF-8 -->

<encoding>UTF-8</encoding>

<!-- protocol for script and stylesheets -->

<!-- defaults to https -->

<protocol>http</protocol>

<!-- title of the document -->

<documentTitle>Extent</documentTitle>

<!-- report name - displayed at top-nav -->

<reportName>Grasshopper Report</reportName>

<!-- location of charts in the test view -->

<!-- top, bottom -->

<testViewChartLocation>bottom</testViewChartLocation>

<!-- custom javascript -->

<scripts>

<![CDATA[

$(document).ready(function() {

});

]]>

</scripts>

<!-- custom styles -->

<styles>

<![CDATA[

]]>

</styles>

</configuration>

</extentreports>

Step 4. TestRunner class:

package testrunners;

@RunWith(Cucumber.class)

@CucumberOptions(

features = {"src/test/resources/AppFeatures"},

glue = {"stepdefinitions", "AppHooks"},

plugin = {"pretty", “usgage”,

"com.aventstack.extentreports.cucumber.adapter.ExtentCucumberAdapter:",

"timeline:test-output-thread/"}

)

public class MyTestRunner {

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Cucumber Data Table \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\* Data table feature \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Feature: Tata table test feature

I want to use this template for my feature file

Scenario: user information details

Given user on login page

When user enters information in details

| jawad | karim | 473847 |main street | famous city | bright state |

| donald | Trump | 558566 |2nd street | wonder city | super state |

| joe | Biden | 647474 |3rd street | Nolla city | carten state |

Then user updated info successfully

Scenario: user information details using Map function

Given user on login page MapFunction

When user enters information in details using map

| first name | last name | phone no |street name |city name |state name |

| jawadi | karimi | 473847 |4th street |famous city |bright state |

| donaldo | Trumpo | 558566 |5th street |wonder city |super state |

| joee | Bidendo | 647474 |6th street |Nolla city |carten state |

Then user updated info successfully using map

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Data table step definitions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**package** stepDefinitions.dataTable;

**import** java.util.List;

**import** java.util.Map;

**import** io.cucumber.datatable.DataTable;

**import** io.cucumber.java.en.\*;

**public** **class** Data\_table\_steps {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* data table steps using List of List method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Given("user on login page")

**public** **void** user\_on\_login\_page() {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*at given step\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

@When("user enters information in details")

**public** **void** user\_enters\_information\_in\_details(DataTable dataTable) {

List<List<String>> dataList = dataTable.asLists(String.**class**);

**for**(List<String> e : dataList) {

System.***out***.println("List of list data : "+e);

}

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*data list\*\*\*\*\*\*\*\*\*\*\*\* "+dataList);

System.***out***.println("===first row first column data : "+ dataList.get(0).get(0));

System.***out***.println("===first row 2nd column data : "+ dataList.get(0).get(1));

}

@Then("user updated info successfully")

**public** **void** user\_updated\_info\_successfully() {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*user updated his info\*\*\*\*\*\*\*\*\*\*\*\*");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* data table steps using List of Map method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Given("user on login page MapFunction")

**public** **void** user\_on\_login\_page\_map\_function() {

System.***out***.println("\*\*\*\*\*\*\*\*\* Given for List of Map Function \*\*\*\*\*\*\*\*\*\*\*");

}

@When("user enters information in details using map")

**public** **void** user\_enters\_information\_in\_details\_using\_map(DataTable dataTable) {

List<Map<String, String>> dataMap = dataTable.asMaps(String.**class**, String.**class**);

System.***out***.println("data map : "+dataMap);

System.***out***.println("first name :"+ dataMap.get(0).get("first name"));

}

@Then("user updated info successfully using map")

**public** **void** user\_updated\_info\_successfully\_using\_map() {

System.***out***.println("\*\*\*\*\*\* information updated using map \*\*\*\*\*\*\*\* ");

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* parallel test Cucumber testNG \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\* parallel test in scenario level...**

**1) create a package named ‘parallel’ in src/test/java folder add all the step deifinition, Hooks, runner Classes then glue the package.**

**And feature will be in the src/test/resources add the feature package to cucumber option features.**

**2)** Add the **Surefire plugin configuration** to the build section to the POM.

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.0</version>

<configuration>

<parallel>methods</parallel>

<useUnlimitedThreads>true</useUnlimitedThreads>

</configuration>

</plugin>

**:change the parallel option for classes and method level.**

**<configuration>**

**<parallel>classesAndMethods</parallel>**

**<useUnlimitedThreads>true</useUnlimitedThreads>**

**</configuration>**

**: To set the thread count to a specific number** **instead of useUnlimitedThreads use the below setting.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* for scenario level \*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**<configuration>**

**<parallel>methods</parallel>**

**<threadCount>4</threadCount>**

**</configuration>**

**NB. treadCount 4 means 4 scenarios will run in parallel.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* For class level \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**<configuration>**

**<parallel>classes</parallel>**

**<threadCount>4</threadCount>**

**</configuration>**

**:** The thread count in the above setting is **4 threads per core**. If you want this to be **4 threads across all cores** set the**perCoreThreadCount** to **false**.

<configuration>

<parallel>methods</parallel>

<threadCount>4</threadCount>

<perCoreThreadCount>false</perCoreThreadCount>

</configuration>

**:** In case of **multiple runners** one can also set the parallel option to classesAndMethods or classes in addition to methods.

**<configuration>**

**<parallel>classesAndMethods</parallel>**

**<useUnlimitedThreads>true</useUnlimitedThreads>**

**</configuration>**

**: to run with Maven =================================**

**<configuration>**

**<includes> it will run all class that name has the word runner.**

**<include>\*\*/runner\*/<include>**

**</includes>**

**<parallel>methods</parallel>**

**<threadCount>4</threadCount>**

**</configuration>**

**3) add the following code to the runner class....**

**public class RunCucumberTest extends AbstractTestNGCucumberTests{**

**@Override**

**@DataProvider(parallel = true)**

**public Object[][] scenarios() {**

**return super.scenarios();**

**}**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* parallel cucumber test with .xml file \*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\* feature file**

@AllScenario

Feature: Validate login Function

@validID\_Password

Scenario Outline: validate login page with valid data

Given opent browser

And enter url

When enter valid userName "<validID>"

And enter valid password "<validPassWord>"

And click on login button

Then user should be logged in

Examples:

| validID | validPassWord |

| uid1 | pass1 |

@invalidId

Scenario Outline: validate login page with invalid ID

Given opent browser

And enter url

When enter invalid userName "<invalidID>"

And enter valid password "<validP>"

And click on login button

Then user should be logged in

Examples:

| invalidID | validP |

| uid2 | pass2 |

@invalIdPass

Scenario Outline: validate login page with invalid Pass

Given opent browser

And enter url

When enter valid userName "<validI>"

And enter invalid password "<invalidP>"

And click on login button

Then user should be logged in

Examples:

| validI | invalidP |

| uid3 | pass3 |

@invalidID\_Password

Scenario Outline: validate login page with invalid data

Given opent browser

And enter url

When enter invalid userName "<invalidI>"

And enter invalid password "<invalidP>"

And click on login button

Then user should be logged in

Examples:

| invalidI | invalidP |

| uid4 | pass4 |

**\*\*\* step definition**

**package** stepDefinitions.loginPageStepDefinitions;

**import** io.cucumber.java.After;

**import** io.cucumber.java.Before;

**import** io.cucumber.java.en.\*;

**public** **class** LoginPageSteps {

@Before

**public** **void** beforeScenario() {

System.***out***.println("before scenario");

}

@After

**public** **void** afterScenario() {

System.***out***.println("after scenario");

}

@Given("opent browser")

**public** **void** opent\_browser() {

System.***out***.println("user open browser");

}

@And("enter url")

**public** **void** enter\_url() {

System.***out***.println("user enter url");

}

@When("enter valid userName {string}")

**public** **void** enter\_valid\_user\_name(String string) {

System.***out***.println("user enter valid user name : "+string);

}

@When("enter valid password {string}")

**public** **void** enter\_valid\_password(String string) {

System.***out***.println("user enter valid pass word : "+string);

}

@And("click on login button")

**public** **void** click\_on\_login\_button() {

System.***out***.println("user click on login button");

}

@Then("user should be logged in")

**public** **void** user\_should\_be\_logged\_in() {

System.***out***.println("user should be logged in");

}

@When("enter invalid userName {string}")

**public** **void** enter\_invalid\_user\_name(String string) {

System.***out***.println("user enter invalid user name : "+string);

}

@And("enter invalid password {string}")

**public** **void** enter\_invalid\_password(String string) {

System.***out***.println("user enter invalid pass word : "+string);

}

}

**\*\*\* runner files ===================================**

**package** runners;

**import** io.cucumber.testng.AbstractTestNGCucumberTests;

**import** io.cucumber.testng.CucumberOptions;

@CucumberOptions(

features = "Features/loginValidData.feature",

glue = "stepDefinitions/loginPageStepDefinitions",

dryRun = **false**,

monochrome = **true**,

tags = "@validID\_Password",

plugin = "pretty"

)

**public** **class** LoginPageTestRunnerValidDATA **extends** AbstractTestNGCucumberTests{

}

**--------------------------------**

**package** runners;

**import** io.cucumber.testng.AbstractTestNGCucumberTests;

**import** io.cucumber.testng.CucumberOptions;

@CucumberOptions(

features = "Features/loginValidData.feature",

glue = "stepDefinitions/loginPageStepDefinitions",

dryRun = **false**,

monochrome = **true**,

tags = "@invalidId",

plugin = "pretty"

)

**public** **class** LoginPageTestRunnerInValidID **extends** AbstractTestNGCucumberTests{

}

**--------------------------------**

**package** runners;

**import** io.cucumber.testng.AbstractTestNGCucumberTests;

**import** io.cucumber.testng.CucumberOptions;

@CucumberOptions(

features = "Features/loginValidData.feature",

glue = "stepDefinitions/loginPageStepDefinitions",

dryRun = **false**,

monochrome = **true**,

tags = "@invalidPass",

plugin = "pretty"

)

**public** **class** LoginPageTestRunnerInValidPassWord **extends** AbstractTestNGCucumberTests{

}

**--------------------------------**

**package** runners;

**import** io.cucumber.testng.AbstractTestNGCucumberTests;

**import** io.cucumber.testng.CucumberOptions;

@CucumberOptions(

features = "Features/loginValidData.feature",

glue = "stepDefinitions/loginPageStepDefinitions",

dryRun = **false**,

monochrome = **true**,

tags = "@invalidID\_Password",

plugin = "pretty"

)

**public** **class** LoginPageTestRunnerInValidDATA **extends** AbstractTestNGCucumberTests{

}

**\*\*\* xml files**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* parallel=*"false"*>

<test name=*"Test"*>

<classes>

<class name=*"runners.LoginPageTestRunnerValidDATA"*></class>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

----------------------------------------------------------

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* parallel=*"false"*>

<test name=*"Test"*>

<classes>

<class name=*"runners.LoginPageTestRunnerInValidID"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

------------------------------------------------------------

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* parallel=*"false"*>

<test name=*"Test"*>

<classes>

<class name=*"runners.LoginPageTestRunnerInValidPassWord"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

---------------------------------------------------------------

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* parallel=*"false"*>

<test name=*"wholeSuite"*>

<classes>

<class name=*"runners.LoginPageTestRunnerInValidDATA"*></class>

</classes>

</test>

</suite> <!-- Suite -->

--------------------------------------------------------------------

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* parallel=*"classes"*>

<test name=*"wholeSuite"*>

<classes>

<class name=*"runners.LoginPageTestRunnerValidDATA"*></class>

<class name=*"runners.LoginPageTestRunnerInValidID"*></class>

<class name=*"runners.LoginPageTestRunnerInValidPassWord"*></class>

<class name=*"runners.LoginPageTestRunnerInValidDATA"*></class>

</classes>

</test>

</suite> <!-- Suite -->

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* how to re-run failed test \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**package** runners;

**import** io.cucumber.testng.AbstractTestNGCucumberTests;

**import** io.cucumber.testng.CucumberOptions;

@CucumberOptions(

features = "Features/loginValidData.feature",

glue = "stepDefinitions/loginPageStepDefinitions",

dryRun = **false**,

monochrome = **true**,

tags = "@invalidID\_Password",

plugin = {

"pretty",

“rerun:target/failedrerun.txt”

}

)

**public** **class** LoginPageTestRunnerInValidDATA **extends** AbstractTestNGCucumberTests{

}

**\*\*\*\*\*\*Cucumber verified Code\*\*\*\*\*\***

**: dependency versions**\_ selenium-java(3.141.59), testng(6.14.3),

WebDriverManager(latest version), Cucumber-java(7.0.0), Cucumber-Junit(7.0.0),

ExtentReport\_adapter 7.

**: Cucumber data driven test**

**: feature file\_\_\_\_\_**

@tag

Feature: Login to Account

@tag2

Scenario Outline: login with multiple datas

Given User open browser and enter url

When User enter userName as "<uname>"

And User enter passWord as "<pass>"

Then clicked login button

Examples:

| uname | pass |

| mngr455792 | UsEzadU |

| mngrd55792 | UsEkadU |

| mngr455792 | UsEzadU |

**: Step Definiton class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**package** stepDefinitions.stepDfinition;

**public** **class** LoginSteps {

WebDriver driver;

String baseUrl = "https://demo.guru99.com/V4/index.php";

LoginPage login = **new** LoginPage(driver);

@Given("User open browser and enter url")

**public** **void** user\_open\_browser\_and\_enter\_url() {

WebDriverManager.*chromedriver*().setup();

driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(20, TimeUnit.***SECONDS***);

driver.get(baseUrl);

}

@When("User enter userName as {string}")

**public** **void** user\_enter\_user\_name\_as(String uName) {

driver.findElement(By.*xpath*("//input[@name='uid']")).sendKeys(uName);

}

@And("User enter passWord as {string}")

**public** **void** user\_enter\_pass\_word\_as(String pass) {

driver.findElement(By.*xpath*("//input[@name='password']")).sendKeys(pass);

}

@Then("clicked login button")

**public** **void** clicked\_login\_button() **throws** Exception {

driver.findElement(By.*xpath*("//input[@name='btnLogin']")).click();

//driver.close();

Thread.*sleep*(2000);

}

}

**: Test Runner Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**package** runners;

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = "Features/Login.feature",

glue = "stepDefinitions/stepDfinition",

dryRun = **false**

)

**public** **class** LoginTestRunner {

}

**Cucumber data driver test usin POM**

**: Page class….**

**package** pages;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.support.FindBy;

**import** org.openqa.selenium.support.PageFactory;

**public** **class** LoginPage {

WebDriver driver;

**public** LoginPage(WebDriver driver) {

**this**.driver = driver;

PageFactory.*initElements*(driver, **this**);

}

@FindBy(xpath = "//input[@name='uid']")

WebElement username;

@FindBy(xpath = "//input[@name='password']")

WebElement password;

@FindBy(xpath = "//input[@name='btnLogin']")

WebElement loginbutton;

**public** **void** enterUsername(String uid) {

username.sendKeys(uid);

}

**public** **void** enterPassword(String pass) {

password.sendKeys(pass);

}

**public** **void** clickLoginButton() {

loginbutton.click();

}

}

**: features . . .**

Feature: LoginPage with pom

Scenario Outline: Login to account with pom

Given User opens browser

And User open Url

When User enter userID as "<uname>"

And User enter PassWord as "<pass>"

Then User clicked on login button

Examples:

| uname | pass |

| mngr455792 | UsEzadU |

| mngrd55792 | UsEkadU |

| mngr455792 | UsEzadU |

**StepDefinitions. . . class**

**package** stepDefinitions.loginWithPOM\_definiton;

**public** **class** LoginWithPomSteps {

WebDriver driver;

LoginPage login;

@Given("User opens browser")

**public** **void** user\_opens\_browser() {

WebDriverManager.*chromedriver*().setup();

driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(10, TimeUnit.***SECONDS***);

login = **new** LoginPage(driver);

}

@And("User open Url")

**public** **void** user\_open\_url() {

driver.get("https://demo.guru99.com/V4/index.php");

}

@When("User enter userID as {string}")

**public** **void** user\_enter\_user\_id\_as(String un) {

login.enterUsername(un);

}

@And("User enter PassWord as {string}")

**public** **void** user\_enter\_pass\_word\_as(String pw) {

login.enterPassword(pw);

}

@Then("User clicked on login button")

**public** **void** user\_clicked\_on\_login\_button() {

login.clickLoginButton();

}

}

**: runner class . . . (class name must starts or ends with the word Test)**

**Note: must to have surefire plugin in pom.xml file**

**package** runners;

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = "Features/LoginWithPOM.feature",

glue = "stepDefinitions/loginWithPOM\_definiton",

dryRun = **false**)

**public** **class** TestLogin\_withPOM {

}

**: Cucumber with Extent Reports adapter 7**

Step 1: Add dependency Cucumber-java 7.0.0 and Cucumber-Junit 7.0.0

Step 2: Add tech grasshopper maven dependency (Extent Report Cucumber 7 Adapter)

Step 3: Create **extent.properties** file in src/test/resources/ ...

Step 4: Add the plugin plugin = {"com.aventstack.extentreports.cucumber.adapter.ExtentCucumberAdapter:"}

Step 4: Execute your code. Then refresh the project.. test-output folder will be there.

**: extent.properties file­­­­­­­­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

extent.reporter.spark.start=true

extent.reporter.spark.out=test-output/SparkReport/Spark.html

**: Test Runner class\_\_\_\_\_\_\_\_\_\_\_\_\_**

**package** runners;

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = "Features/Login2.feature",

glue = "stepDefinitions/loginStepsWithReport",

dryRun = **false**,

monochrome = **true**,

plugin = {"com.aventstack.extentreports.cucumber.adapter.ExtentCucumberAdapter:"})

**public** **class** LoginTestRunnerWithReport {

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_

**Regular Extent report with Cucumber and Hooks\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**: BaseClass . .** .

**package** pages;

**public** **class** BaseClass {

**public** WebDriver driver;

**public** **static** String randomestring() {

String generatedString1 = RandomStringUtils.*randomAlphabetic*(5);

**return** (generatedString1);

}

}

**: Page Class . . .**

**package** pages;

**public** **class** LoginPage {

WebDriver driver;

**public** LoginPage(WebDriver driver) {

**this**.driver = driver;

PageFactory.*initElements*(driver, **this**);

}

@FindBy(xpath = "//input[@name='uid']")

WebElement username;

@FindBy(xpath = "//input[@name='password']")

WebElement password;

@FindBy(xpath = "//input[@name='btnLogin']")

WebElement loginbutton;

**public** **void** enterUsername(String uid) {

username.sendKeys(uid);

}

**public** **void** enterPassword(String pass) {

password.sendKeys(pass);

}

**public** **void** clickLoginButton() {

loginbutton.click();

}

}

**: features . . .**

Feature: Login to Account and extent report

Scenario Outline: loginpage test with extent report

Given User1 open browser

And User1 enter url

When User1 enter userName as "<uname>"

And User1 enter passWord as "<pass>"

Then user1 clicked login button

Examples:

| uname | pass |

| mngr455792 | UsEzadU |

| mngrd55792 | UsEkadU |

| mngr455792 | UsEzadU |

**: StepDefinitions class . . .**

**package** stepDefinitions.loginWithRegularExtent;

**public** **class** RegularExtentSteps **extends** BaseClass{

ExtentReports extent;

ExtentSparkReporter spark;

ExtentTest logger;

LoginPage login;

String baseUrl;

@Before

**public** **void** setup() {

extent = **new** ExtentReports();

spark = **new** ExtentSparkReporter("./reports/bankReport.html");

extent.attachReporter(spark);

// probaly can be set in cunstructor !

}

@After

**public** **void** tearDown() {

extent.flush();

}

@Given("User1 open browser")

**public** **void** user1\_open\_browser() {

WebDriverManager.*chromedriver*().setup();

driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(10, TimeUnit.***SECONDS***);

login = **new** LoginPage(driver);

baseUrl = "https://demo.guru99.com/V4/index.php";

logger = extent.createTest("Loginpage test with extent reports");

logger.info("browser opened");

}

@And("User1 enter url")

**public** **void** user1\_enter\_url() {

driver.get("https://demo.guru99.com/V4/index.php");

logger.info("entered url");

}

@When("User1 enter userName as {string}")

**public** **void** user1\_enter\_user\_name\_as(String n) {

login.enterUsername(n);

logger.info("entered user name");

}

@And("User1 enter passWord as {string}")

**public** **void** user1\_enter\_pass\_word\_as(String p) {

login.enterPassword(p);

logger.info("entered password");

}

@Then("user1 clicked login button")

**public** **void** user1\_clicked\_login\_button() {

login.clickLoginButton();

logger.info("clicked login button");

System.***out***.println("random String : "+**super**.*randomestring*());

}}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* cucumber with excel data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

: create a excel file in a data folder under project

@TS\_signin\_03

Scenario Outline: User on login page and login with invalid and valid inputs from Excel "<Sheetname>" and <RowNumber>

Given The user is on signin page

When The user enter sheet "<Sheetname>" and <RowNumber>

Then click login button

Examples:

| Sheetname | RowNumber |

| Sheet1 | 0 |

| Sheet1 | 1 |

| Sheet1 | 2 |

| Sheet1 | 3 |

//Step definition..

// User read both invalid and valid data from excel

@When("The user enter sheet {string} and {int}")

**public** **void** the\_user\_enter\_sheet\_and(String sheetname, Integer rownumber)

**throws** InvalidFormatException, IOException {

ExcelReader reader = **new** ExcelReader();

List<Map<String, String>> testdata = reader.getData(Excelpath, sheetname);

username = testdata.get(rownumber).get("username");

password = testdata.get(rownumber).get("password");

message = testdata.get(rownumber).get("expectedmessage");

Loggerload.info("User Enter username as \" " + username + " \"and Password as \" " + password + "\" ");

**if** (username != **null** || password != **null**){

sign.doLogin(username, password);

}

}