Behavior Driven Development

BDD :

- Behavior-Driven Development (BDD) is the software development process
- BDD is all about collaboration between the teams.
- Generates the documents that can be understood by all teams and stakeholders.

What is CUCUMBER :

- Cucumber is a tool that supports BDD.
- It understands the documents of the requirements and turns it into automated tests.

Traditional Development Process :

- Business owner tells Business Analyst (or) Product owner about his requirements.
- Product owner writes the requirements document. This should share to
  a) Developer converts requirements into code.
  b) Tester converts requirements into test cases.
- Test execution, bug fixing, retesting & regression.

Drawbacks :

Dev and QA team will not be involved in requirements discussion, user story creation and examples creation.

BDD PROCESS :

- Business owner tells Business Analyst (or) Product owner about his requirements.
- PO, DEV and QA meet to discuss the requirements and explore with example to define the system behavior.
- Examples are documented using specifications language like Gherkin that can be used for development and automation.
- Test execution, bug fixing, retesting, regression & release.

What is Gherkin ?

Gherkin is like a special language that helps you write down instructions in plain English for testing software.

Gherkin keywords

Feature :

- The purpose of the Feature keyword is to provide a high-level description of a software feature, and to group related scenarios.
- The first primary keyword in a Gherkin document must always be Feature, followed by a : and a short text that describes the feature.

Eg : **Feature: Guess the word**

Rule :

- It provides additional information for a feature.
- The purpose of the Rule keyword is to represent one business rule that should be implemented.

Eg : **Feature: Highlander**

Rule: There can be only One

Steps

- Each step starts with Given, When, Then, And, or But.

Given

- Given steps are used to describe the initial context of the system - the scene of the scenario.

Examples:

- Mickey and Minnie have started a game
- I am logged in
- Joe has a balance of £42

When

- When steps are used to describe an event, or an action. This can be a person interacting with the system, or it can be an event triggered by another system

Examples:

- Guess a word
- Invite a friend
- Withdraw money

Then

- Then steps are used to describe an expected outcome, or result

Examples:

- See that the guessed word was wrong
- Receive an invitation
- Card should be swallowed

And, But

- If you have successive Given's or Then's, you could write:

Example: Multiple Givens
Given one thing
Given another thing
Given yet another thing
When I open my eyes
Then I should see something
Then I shouldn't see something else

- Or, you could make the example more fluidly structured by replacing the successive Given's or Then's with And's and But's:

Example: Multiple Givens
Given one thing
And another thing
And yet another thing
When I open my eyes
Then I should see something
But I shouldn't see something else

- **Gherkin also supports using an asterisk (*) in place of any of the normal step keywords. This can be helpful when you have some steps that are effectively a list of things**

```
Scenario: All done
  Given I am out shopping
  * I have eggs
  * I have milk
  * I have butter
  When I check my list
  Then I don't need anything
```

Scenario:

- Definition: A Scenario is a single test case that describes a specific situation or feature to be tested.
- Purpose: It helps verify that a part of the software works as expected.

```
Scenario: Successful login
  Given the user is on the login page
  When the user enters valid credentials
  Then the user is redirected to the dashboard
```

Scenario Outline:

- Definition: A Scenario Outline is used to run the same Scenario multiple times with different sets of data.
- Purpose: It helps test the same feature with various inputs efficiently.
- Scenario outlines allow us to more concisely express these scenarios through the use of a template with < >-delimited parameters:

```
Scenario Outline: Login with multiple users
  Given the user is on the login page
  When the user enters "<username>" and "<password>"
  Then the user is redirected to the dashboard

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

What are Step Definitions ?

- They link Gherkin instructions (written in plain English) to the actual code
- When Gherkin gives an instruction, step definitions tell the code what action to take.

Feature File:

Definition: A feature file is a plain text file containing one or more scenarios written using Gherkin language.

Purpose: It describes the functionality of the software feature in a way that is easy for both non-technical stakeholders and developers to understand.

Structure: Each feature file starts with the Feature keyword, followed by a brief description of the feature. It then contains one or more Scenario or Scenario Outline sections.

Feature: User login

Scenario: Successful login with valid credentials

Given the user is on the login page

When the user enters valid username and password

Then the user is redirected to the dashboard

Scenario Outline: Unsuccessful login with invalid credentials

Given the user is on the login page

When the user enters "<username>" and "<password>"

Then an error message is displayed

Examples:

| username | password |

| invalid1 | wrongpass |

| invalid2 | wrongpass |

Maven project creation and cucumber installation :

- Create new maven project
- Add maven dependencies

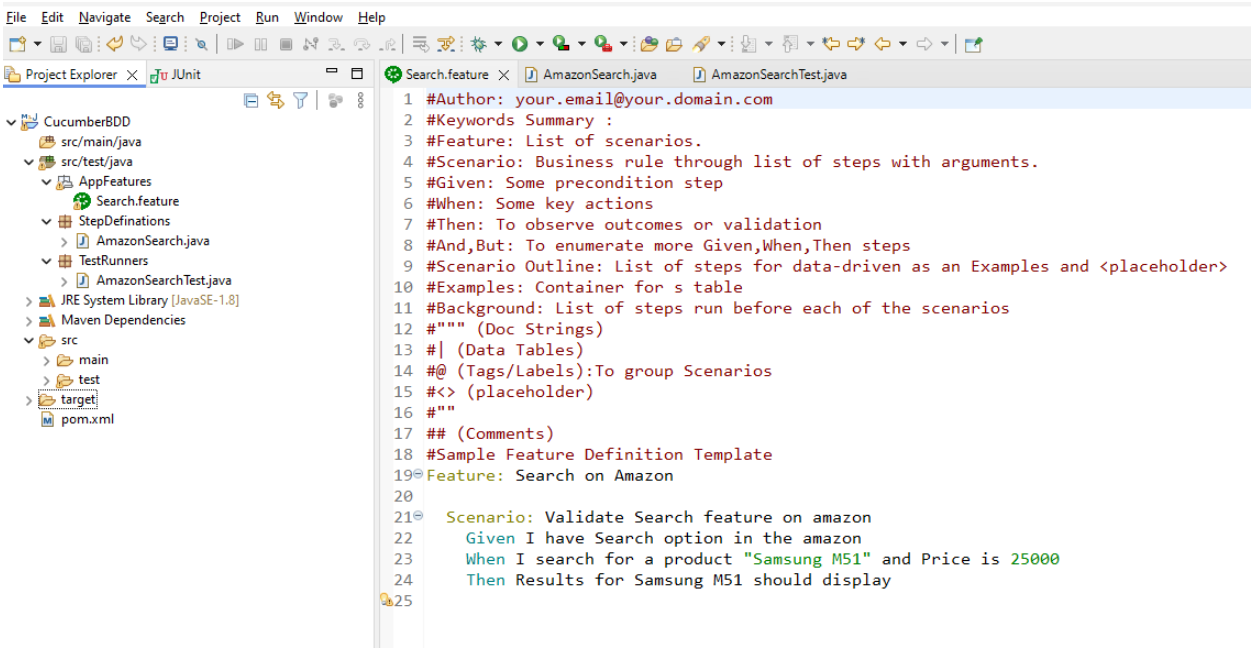
```
<dependency> // <!-- https://mvnrepository.com/artifact/junit/junit -->
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
</dependency>
```

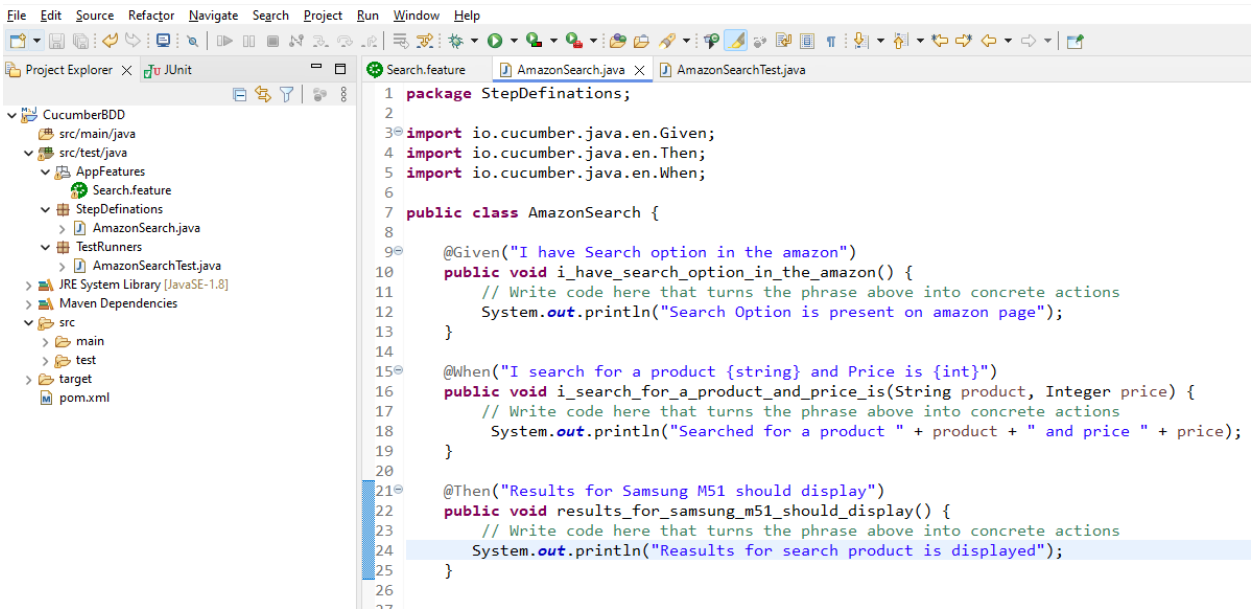
```
<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-java -->
<dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-java</artifactId>
        <version>7.15.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
<dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>7.16.1</version>
        <scope>test</scope>
</dependency>
```

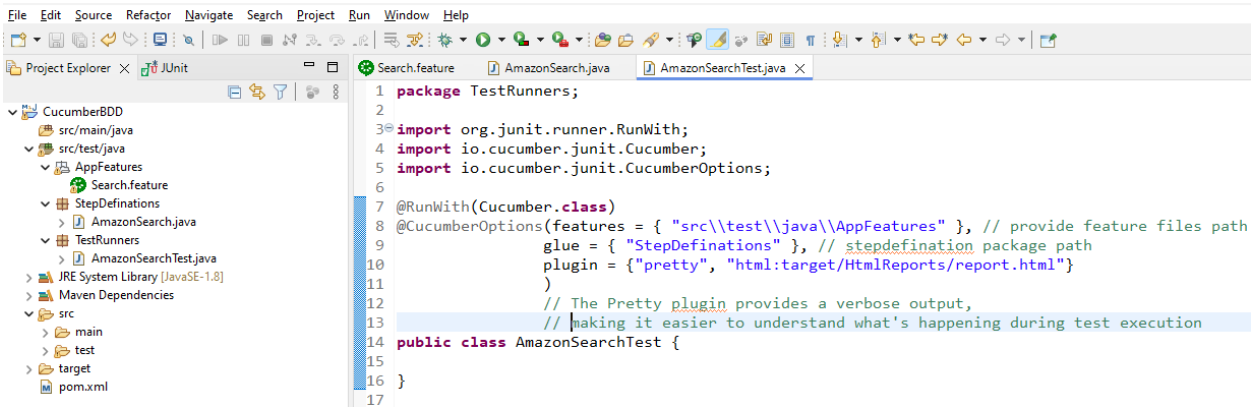
- Download cucumber plugin from eclipse marketplace
- Create a folder for “Features” under src/test/resources
- Under features folder create new feature file with **.feature** extension
- In feature file add contents → Feature, Scenario, Steps, Scenario Outline, Examples, Tags, Comments
- Try to Run Feature file
- Add step Definition/Glue code under src/test/java package

Automation Test Case Creation Process :

1) Create feature file → 2) Create Step Definition class for feature scenarios →3) create runner class

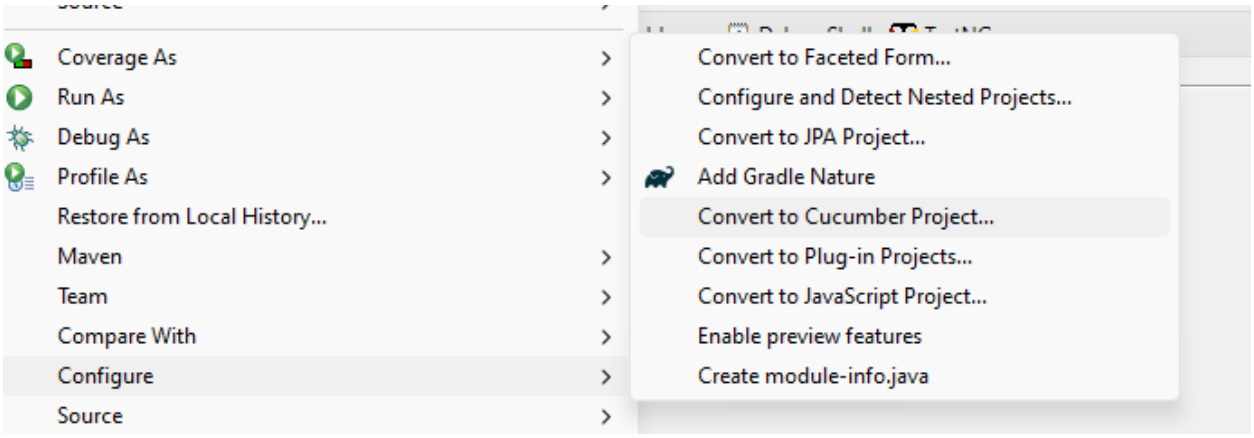




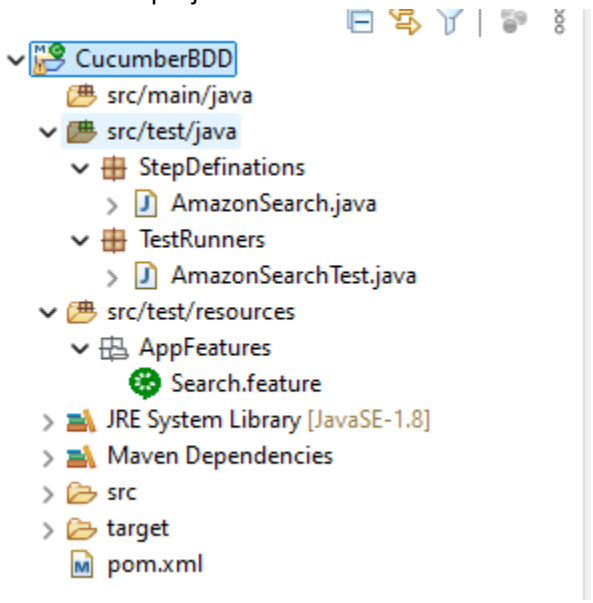


Map feature file steps with step definition methods in step definition class :

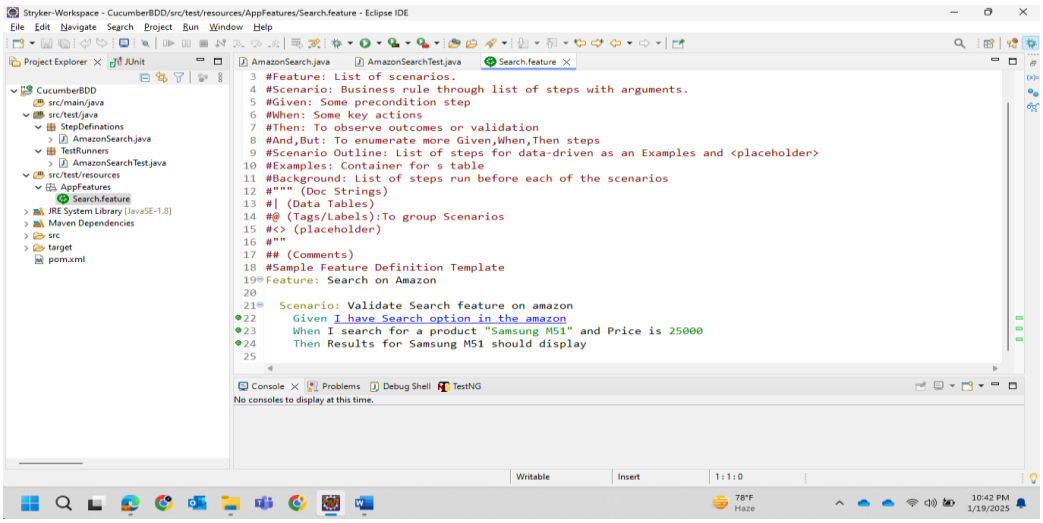
- Right click on project -> configure -> Convert to Cucumber Project

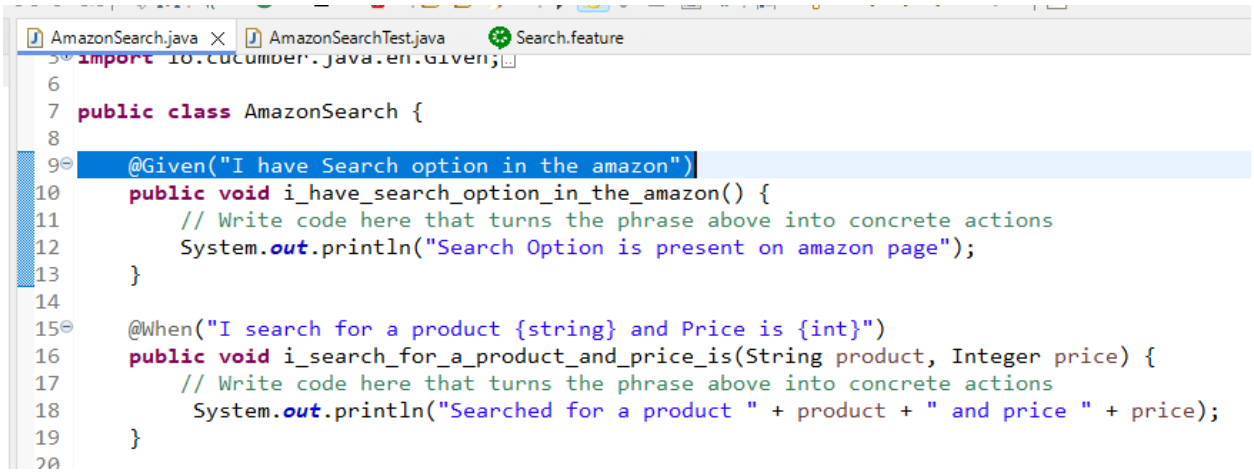


- Now maven project converted in to maven cucumber project

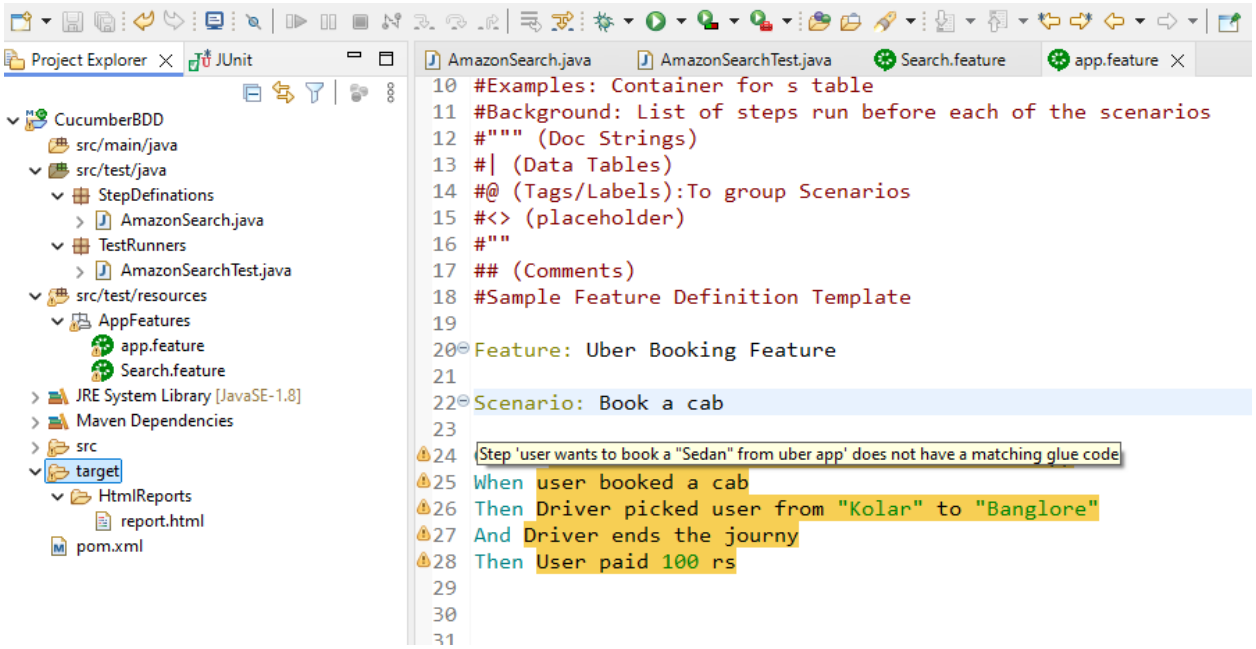


- Control + click to navigate to step definition methods of feature file scenarios.

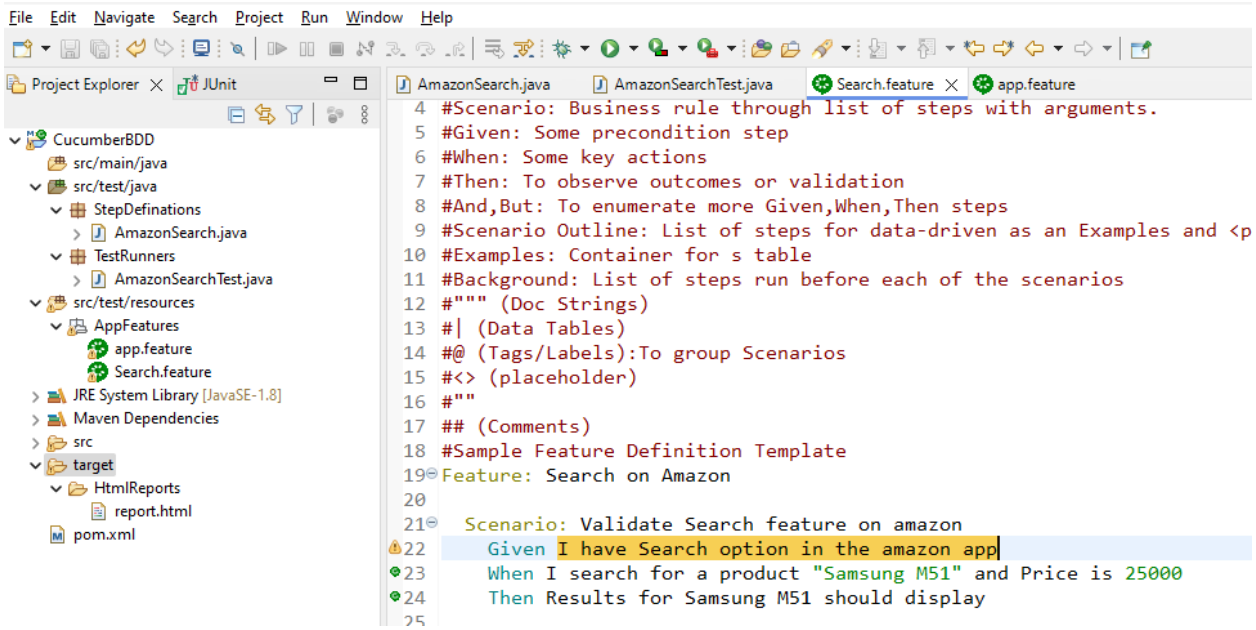




- The moment you created feature file with scenarios, cucumber plug in warns you to create step Definition or Glue code



- If you modify any scenario then also we will get the warning



Regular Expression In Cucumber :

- Regular Expressions → [0-8]+, (\\d+)

- Cucumber Expressions → {String}, {int}

Rules:

- Step def file will be generating cucumber expression by default.
- But we can also use Regular expression in step def file.
- We can mix both cucumber and regular expression in step def file.
- We can't mix both expression in a step def method.

Regular Expressions:

Quantifiers in Reg Exp : + , * , ? , {n}

- How many times a character needs to be occurred

([0-9]+) : 0 to 9 digits appear (once or more)

([0-9]{4}) : 0000 , 9999, 1212, 3456, 1234, 8888

([0-9]*) : zero or more

([0-9]?) : zero or once

[Predefined shorthand character classes - Java 9 Regular Expressions [Book]]

- Cucumber Expressions

UberBookingFeature.feature | BookUberCab.java

```
    @Given("user wants to book a {string} from uber app")
    public void user_wants_to_book_a_from_uber_app(String string) {

    }

    @When("user booked a cab")
    public void user_booked_a_cab() {
       System.out.println("User Booked a cab");
    }

    @Then("Driver picked user from {string} to {string}")
    public void driver_picked_user_from_to(String from, String to) {
        System.out.println("Driver picked user from :" + from + " to " + to);
    }

    @Then("Driver ends the journy")
    public void driver_ends_the_journy() {
       System.out.println("Driver ends the journy");
    }

    @Then("User paid {int} rs")
    public void user_paid_rs(Integer price) {
        System.out.println("User paid amount : "  + price + " rs");
    }

}
```

- Regular Expressions

If we need to use Regx in step definition we need to starts with **^** and ends with **$**

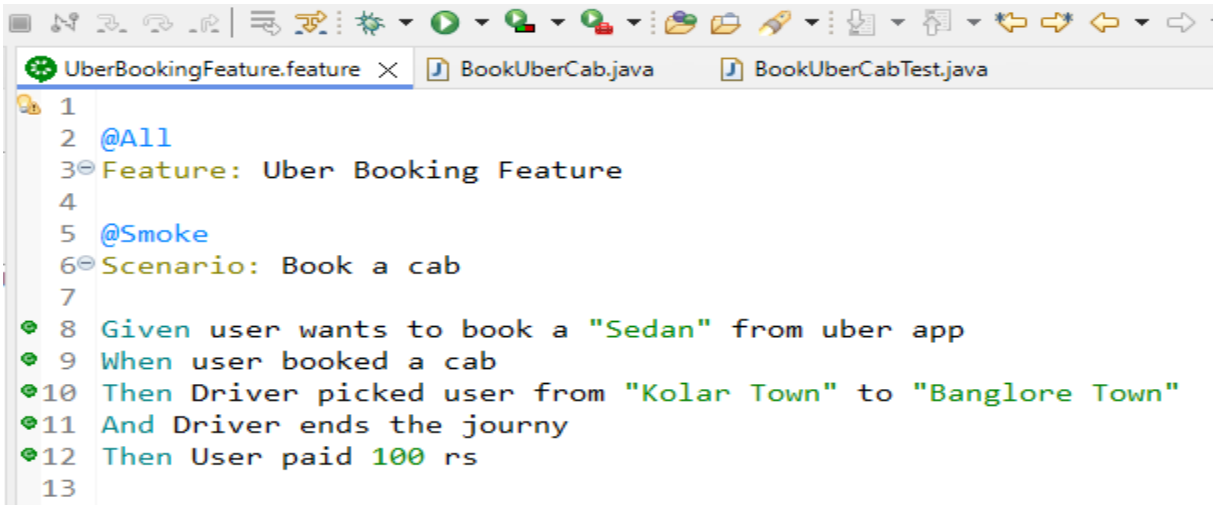
UberBookingFeature.feature | BookUberCab.java | BookUberCabTest.java

```java
    @When("user booked a cab")
    public void user_booked_a_cab() {
       System.out.println("User Booked a cab");
    }

    @Then("^Driver picked user from \"([^\"]+)\" to \"([^\"]+)\"$")
    public void driver_picked_user_from_to(String from, String to) {
        System.out.println("Driver picked user from :" + from + " to " + to);
    }

    @Then("Driver ends the journy")
    public void driver_ends_the_journy() {
       System.out.println("Driver ends the journy");
    }

    @Then("User paid {int} rs")
    public void user_paid_rs(Integer price) {
        System.out.println("User paid amount : "  + price + " rs");
    }

}
```

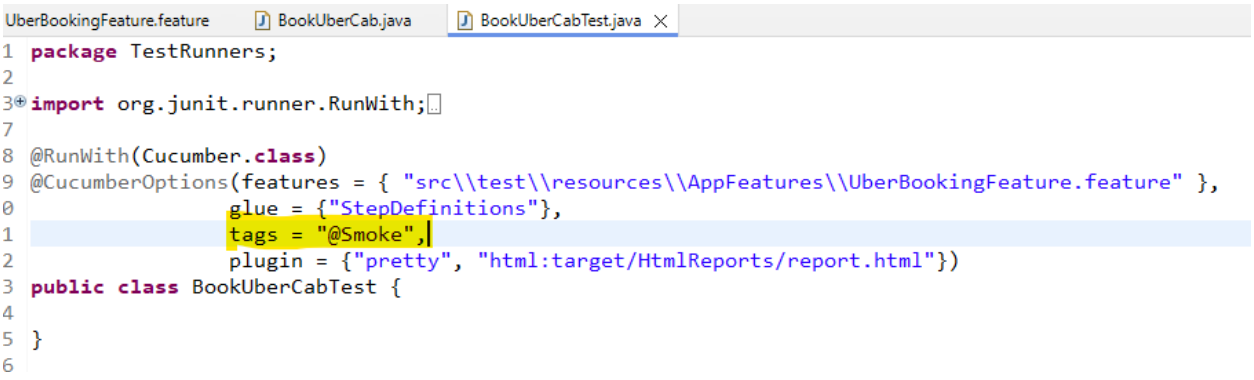
```
#Sample Feature Definition Template

Feature: Uber Booking Feature

Scenario: Book a cab

Given user wants to book a "Sedan" from uber app
When user booked a cab
Then Driver picked user from "Kolar Town" to "Banglore Town"
And Driver ends the journy
Then User paid 100 rs
```

TAGS IN CUCUMBER :

- Features & Scenarios can be marked with Tags
- Tags use @ symbol with some text e.g : **@Smoke**



- In Test Runner we can run with specific tags



- A feature or scenario can have multiple tags eg : **@Smoke @regression**
- Can run with single or multiple Tags
- Can run with combination of tags (or) using **and, or** conditions
- Can skip scenarios having specific tag
- **@All** means all the scenarios are inherited.

UberBookingFeature.feature | BookUberCab.java | BookUberCabTest.java

```
1
2  @All
3  Feature: Uber Booking Feature
4
5  @Smoke @Regression
6  Scenario: Book a cab Sedan
7
8  Given user wants to book a "Sedan" from uber app
9  When user booked a cab
10 Then Driver picked user from "Kolar Town" to "Banglore Town"
11 And Driver ends the journy
12 Then User paid 100 rs
13
14 @Smoke @important
15 Scenario: Book a cab SUV
16
17 Given user wants to book a "SUV" from uber app
18 When user booked a cab
19 Then Driver picked user from "Kolar Town" to "Banglore Town"
20 And Driver ends the journy
21 Then User paid 100 rs
```

UberBookingFeature.feature | BookUberCab.java | BookUberCabTest.java

```
1  package TestRunners;
2
3  import org.junit.runner.RunWith;
7
8  @RunWith(Cucumber.class)
9  @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\UberBookingFeature.feature" },
0                  glue = {"StepDefinitions"},
1                  tags = "@Smoke and @Regression",
2                  plugin = {"pretty", "html:target/HtmlReports/report.html"})
3  public class BookUberCabTest {
4
5  }
```

UberBookingFeature.feature | BookUberCab.java | BookUberCabTest.java

```
1  package TestRunners;
2
3  import org.junit.runner.RunWith;
7
8  @RunWith(Cucumber.class)
9  @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\UberBookingFeature.feature" },
10                 glue = {"StepDefinitions"},
11                 tags = "(@Smoke or @Regression) and not @Smoke",
12                 plugin = {"pretty", "html:target/HtmlReports/report.html"})
13 public class BookUberCabTest {
14
15 }
```

UberBookingFeature.feature | BookUberCab.java | BookUberCabTest.java

```
1  package TestRunners;
2
3  import org.junit.runner.RunWith;
7
8  @RunWith(Cucumber.class)
9  @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\UberBookingFeature.feature" },
10                 glue = {"StepDefinitions"},
11                 tags = "@All",
12                 plugin = {"pretty", "html:target/HtmlReports/report.html"})
13 public class BookUberCabTest {
14
15 }
16
```

Background Keyword

- The Background keyword in Cucumber is used to define a set of steps that are common to all scenarios in a feature file.
- This helps avoid repetition by allowing you to specify these common steps once, and they will be executed before each scenario in the feature file.

*Orders.feature

```
Feature: Home Page
      In order to check my order details
      As a rigistered user
      I want to specify the features of the order details page

  Background:
    Given a rigistered user exists
    Then user is on amazon login page
    When user enters username
    And user enters password
    And user clicks on login button
    Then user navigates to order page
    When user clicks on order link

  Scenario: Check previous order details
    Then user check previous order details

  Scenario: Check open order details
    Then user check open order details

  Scenario: Check cancelled order details
    Then user check cancelled order details

```

```
Scenario: Check open order details   # src/test/resources/AppFeatures/Orders.feature:18
  Given a rigistered user exists     # StepDefinitions.OrderSteps.a_rigistered_user_exists()
  Then user is on amazon login page  # StepDefinitions.OrderSteps.user_is_on_amazon_login_page()
  When user enters username          # StepDefinitions.OrderSteps.user_enters_username()
  And user enters password           # StepDefinitions.OrderSteps.user_enters_password()
  And user clicks on login button    # StepDefinitions.OrderSteps.user_clicks_on_login_button()
  Then user navigates to order page  # StepDefinitions.OrderSteps.user_navigates_to_order_page()
  When user clicks on order link     # StepDefinitions.OrderSteps.user_clicks_on_order_link()
  Then user check open order details # StepDefinitions.OrderSteps.user_check_open_order_details()

Scenario: Check cancelled order details   # src/test/resources/AppFeatures/Orders.feature:21
  Given a rigistered user exists          # StepDefinitions.OrderSteps.a_rigistered_user_exists()
  Then user is on amazon login page       # StepDefinitions.OrderSteps.user_is_on_amazon_login_page()
  When user enters username               # StepDefinitions.OrderSteps.user_enters_username()
  And user enters password                # StepDefinitions.OrderSteps.user_enters_password()
  And user clicks on login button         # StepDefinitions.OrderSteps.user_clicks_on_login_button()
  Then user navigates to order page       # StepDefinitions.OrderSteps.user_navigates_to_order_page()
  When user clicks on order link          # StepDefinitions.OrderSteps.user_clicks_on_order_link()
  Then user check cancelled order details # StepDefinitions.OrderSteps.user_check_cancelled_order_details()

3 Scenarios (3 passed)
24 Steps (24 passed)
```

Orders.feature | *OrderSteps.java

```java
    @Given("a rigistered user exists")
    public void a_rigistered_user_exists() {
    }
    @Then("user is on amazon login page")
    public void user_is_on_amazon_login_page() {
    }
    @When("user enters username")
    public void user_enters_username() {
    }
    @When("user enters password")
    public void user_enters_password() {
    }
    @When("user clicks on login button")
    public void user_clicks_on_login_button() {
    }
    @Then("user navigates to order page")
    public void user_navigates_to_order_page() {
    }
    @When("user clicks on order link")
    public void user_clicks_on_order_link() {
    }
    @Then("user check previous order details")
    public void user_check_previous_order_details() {
    }
    @Then("user check open order details")
    public void user_check_open_order_details() {
    }
    @Then("user check cancelled order details")
    public void user_check_cancelled_order_details() {
    }
```

HOOKS

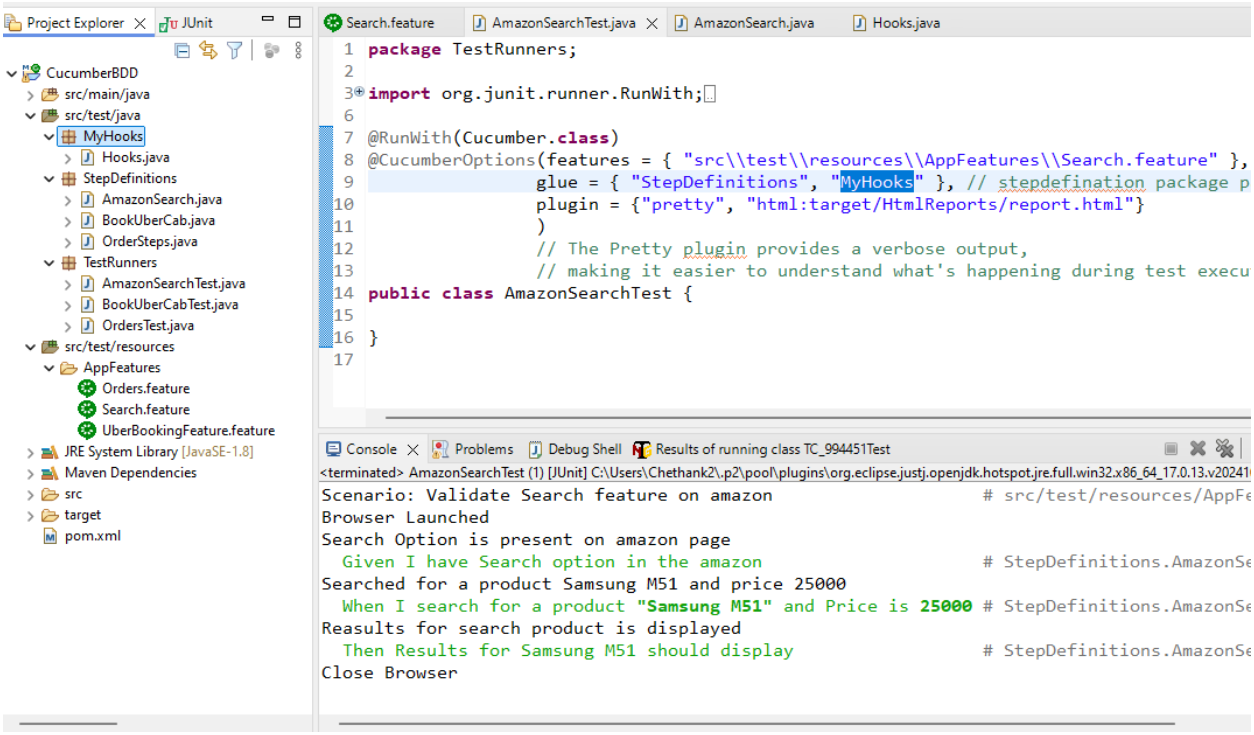
- Blocks of code that runs before or after each scenario.
- Hooks in cucumber are like listeners in TestNg
- Can define Hook using **@Before @After**

Scenario Hooks

- Runs Before and after each scenario

```java
@Before
public void SetUp() {
    System.out.println("Browser Launched");
}
@After
public void tearDown() {
    System.out.println("Close Browser");
}
```

Provide hook package path in glue



Step Hooks

- Runs Before and after each step

```
public class Hooks {

    @Before
    public void SetUp() {
        System.out.println("Browser Launched");
    }
    @After
    public void tearDown() {
        System.out.println("Close Browser");
    }

    @BeforeStep
    public void refreshPage() {
        System.out.println("refresh Page");
    }
    @AfterStep
    public void takeScreenShot() {
        System.out.println("Capture Screenshot");
    }

}
```

Execution with Order

Search.feature | AmazonSearchTest.java | AmazonSearch.java | Hooks.java

```
7
8  public class Hooks {
9
10     @Before(order = 1)
11     public void launchBrowser() {
12         System.out.println("Browser Launched");
13     }
14
15     @Before(order = 2)
16     public void launchUrl() {
17         System.out.println("url Launched");
18     }
19
20     @After(order = 1)
21     public void logout() {
22         System.out.println("user logout");
23     }
24
25     @After(order = 2)
26     public void tearDown() {
27         System.out.println("Close Browser");
28     }
```

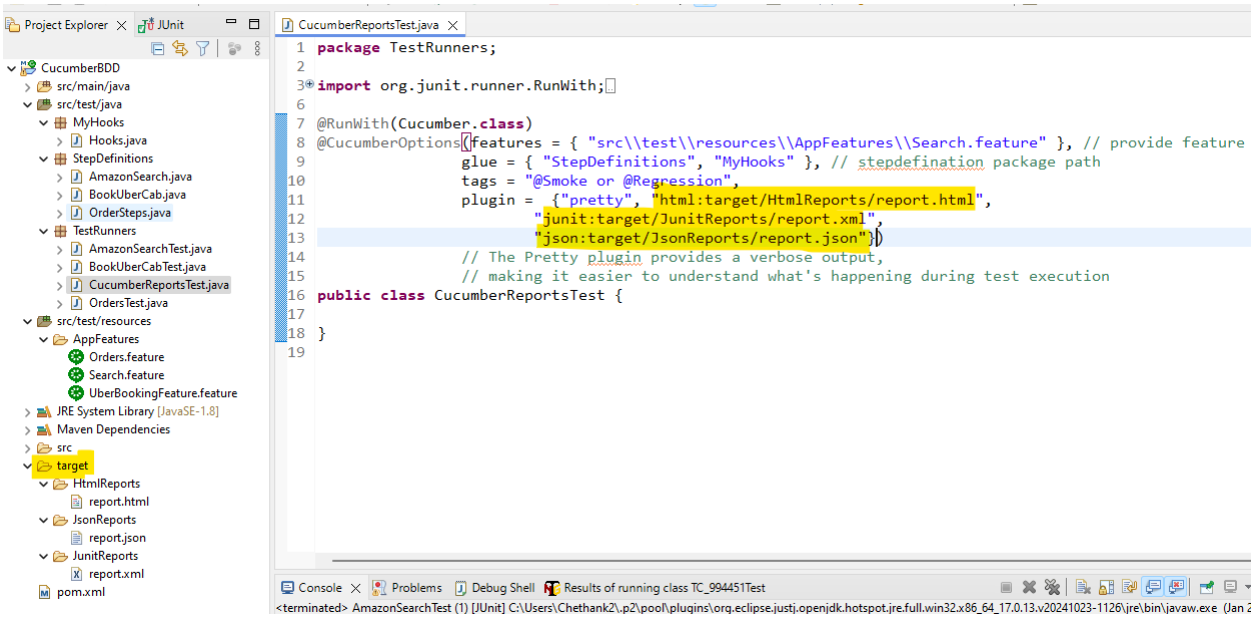
Conditional Hooks

- Hooks associated with tags for conditional execution

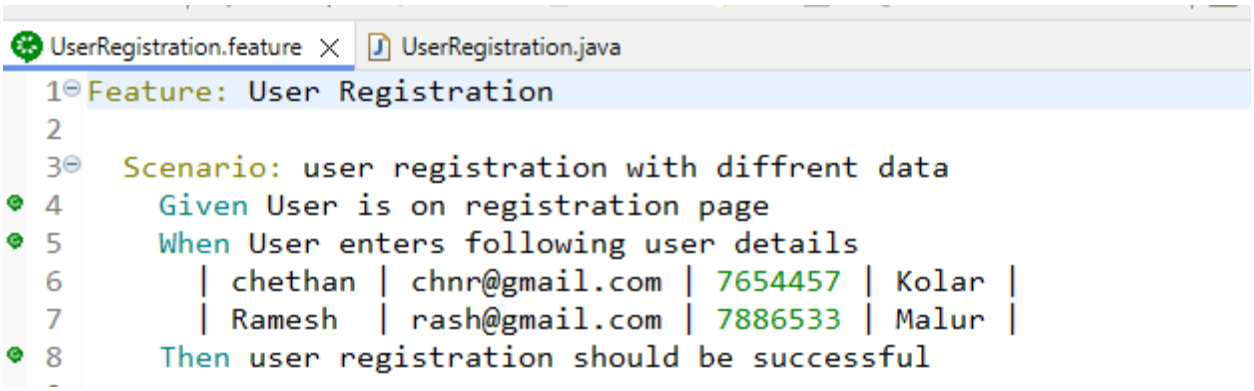
Tags can be used with @Before, @After, @BeforeStep, @AfterStep

```
public class Hooks {

    @Before(value="@Smoke or @Regression")
    public void launchBrowser() {
        System.out.println("Browser Launched");
    }


    @After("@Smoke")
    public void logout() {
        System.out.println("user logout");
    }

    @After(value="@Smoke", order = 1)
    public void closeBrowser() {
        System.out.println("close browser");
    }
```

Generate Different Reports With Cucumber



Data Table (asLists):




```
@When("User enters following user details")
public void user_enters_following_user_details(DataTable dataTable) {

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

    for(List<String> list : userDetails) {
        System.out.println(list);
    }

}
```

Data Table (asMaps):

```
Scenario: user registration with diffrent data with columns
  Given User is on registration page
  When User enters following user details with columns
    | user    | gmail           | number  | city   |
    | chethan | chnr@gmail.com | 7654457 | Kolar |
    | Ramesh  | rash@gmail.com | 7886533 | Malur |
  Then user registration should be successful
```

```
@When("User enters following user details with columns")
public void user_enters_following_user_details_with_columns(DataTable dataTable) {

    List<Map<String, String>> userDetails = dataTable.asMaps(String.class, String.class);
    userDetails.get(0).get("user");
    userDetails.get(1).get("user");

    for (Map<String, String> e : userDetails) {
        System.out.println(e.get("user"));
        System.out.println(e.get("gmail"));
        System.out.println(e.get("number"));
        System.out.println(e.get("city"));
    }
}
```

```
chethan
chnr@gmail.com
7654457
Kolar
Ramesh
rash@gmail.com
7886533
Malur
  When User enters following user details with columns      # StepDefinitions.UserRegistration.user_enters_following_
    | user    | gmail           | number  | city   |
    | chethan | chnr@gmail.com | 7654457 | Kolar |
    | Ramesh  | rash@gmail.com | 7886533 | Malur |
```

Scenario Outline with Examples Keyword (Data Driven Testing in cucumber)

login.feature | login.java

```
1  Feature: login feature
2
3    Scenario Outline: login Test
4      Given user is on login page
5      When Enter the user name "<username>"
6      And Enter password "<password>"
7      Then click on login button
8
9      Examples:
10       | username | password |
11       | chethan  |     5757 |
12       | ramesh   |     7847 |
13
```

```java
public class login {

    @Given("user is on login page")
    public void user_is_on_login_page() {
        System.out.println("user is on login page");
    }

    @When("Enter the user name {string}")
    public void enter_the_user_name(String userName) {
        System.out.println("Use enters the username is " + userName);
    }

    @When("Enter password {string}")
    public void enter_password(String password) {
        System.out.println("Use enters the password is " + password);
    }

    @Then("click on login button")
    public void click_on_login_button() {
       System.out.println("Clicked on login button");
    }

}
```

Console × Problems Debug Shell Results of running class TC_994451Test

<terminated> LoginStep [JUnit] C:\Users\Chethank2\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.13.v20241023-1126\jre\bin\javaw.exe (Jan 26, 2025, 12:07:33 AM

```
Scenario Outline: login Test          # src/test/resources/AppFeatures/login.feature:11
user is on login page
  Given user is on login page         # StepDefinitions.login.user_is_on_login_page()
Use enters the username is chethan
  When Enter the user name "chethan" # StepDefinitions.login.enter_the_user_name(java.lang.String)
Use enters the password is 5757
  And Enter password "5757"           # StepDefinitions.login.enter_password(java.lang.String)
Clicked on login button
  Then click on login button          # StepDefinitions.login.click_on_login_button()

Scenario Outline: login Test         # src/test/resources/AppFeatures/login.feature:12
user is on login page
  Given user is on login page        # StepDefinitions.login.user_is_on_login_page()
Use enters the username is ramesh
  When Enter the user name "ramesh" # StepDefinitions.login.enter_the_user_name(java.lang.String)
Use enters the password is 7847
  And Enter password "7847"          # StepDefinitions.login.enter_password(java.lang.String)
Clicked on login button
  Then click on login button         # StepDefinitions.login.click_on_login_button()
```

Hybrid Framework with Page Object Model (POM) with Cucumber BDD & Selenium (Initial Design).

Designing different components in this Framework:

1. Feature Files

2. Step Definition Classes

3. Configuration Files

4. Cucumber Hooks with before and after

5. Element Utilities/Libraries/Generic Functions

6. Cucumber 6 Extent Report Adaptor for Spark HTML / PDF Reports

7. Test Runners in JUnit

8. Page Classes for POM

9. Maven with pom.xml with different dependencies and plugins

10. Parallel Execution

11. Cucumber 6 Web HTML Reports

12. Screenshot for Failure scenarios

13. Integration with GIT Repo

14. Running test cases from Jenkins

15. Running test cases on Dockerized Selenium GRID

And much more.....

Technologies Used:

1. Selenium WebDriver with Java Language binding

2. Cucumber 6.x JVM library

3. WebDriverManager

4. JDK 1.8

5. Maven (Build tool)

6. Maven Plugins

7. Cucumber extent report 6 adapter

8. JUnit 4.x library

9. Log4j

10. GIT HUB - Git Repo

11. Docker

12. Jenkins

13. Eclipse (IDE)