What is Cucumber?

Cucumber is a testing approach which supports Behavior Driven Development (BDD). It explains the behavior of the application in a simple English text using Gherkin language.

Most of the organizations use Selenium for functional testing. These organizations which are using Selenium, want to integrate Selenium with Cucumber as Cucumber makes **it easy to read and to understand the application flow.**
Cucumber tool is based on the Behavior Driven Development framework that **acts as the bridge** between the following people:

1. Software Engineer and Business Analyst.
2. Manual Tester and Automation Tester.
3. Manual Tester and Developers.

Cucumber BDD framework also **benefits the client to understand the application code** as it uses Gherkin language which is in Plain Text. Anyone in the organization can understand the behavior of the software. The syntax's of Gherkin is in simple text which is readable and understandable.

Note: You may need to install the Cucumber Eclipse Plugin for this to work. Goto — Helps->Install New Software->copy paste the link http://cucumber.github.io/cucumber-eclipse/update-site/ and install

Cucumber Terminologies

It is a Business Readable, Domain Specific Language that lets you describe software's behavior.

Example: Pop up messaged is displayed when buttons are clicked and errors are gone

Keywords Used in Cucumber: Scenario, Feature, Feature file, Step Definition

Scenarios:

In Cucumber Testcases are represented as Scenarios.

Scenarios contain Steps which are equivalent to test Steps and use the following keywords (Gherkin syntax) to denote them: Given, When, Then, But, and And (case sensitive).

- **Given**: Preconditions are mentioned in the Given keyword
- **When**: The purpose of the When Steps is to describe the user action.
- **Then**: The purpose of Then Steps is to observe the expected output. The observations should be related to the business value/benefit of your Feature description.

When we specify a business requirement, sometimes there are multiple pre-conditions, user actions, and expected outcomes

we are going to add one more Scenario and will use the And and But keywords:

- And: This is used for statements that are an addition to the previous Steps and represent positive statements.
- But: This is used for statements that are an addition to previous Steps and represent negative statements.

Feature File

Feature represents Business requirement.

Feature File acts as a Test Suite which consists of all Scenarios.

In Cucumber, Feature files contain Scenarios. We can simply create feature file with. feature extension

Scenarios belonging to specific area of Application will be grouped into one Feature file

The text that immediately follows the Feature keyword, and is in the same line, is the Title of the Feature file

Feature file should contain either Scenario or Scenario Outline. The naming conventions for Feature files should be lowercase with. feature extension.

What is Step Definition?

A Step Definition is a small piece of *code* with a *pattern* attached to it or in other words a Step Definition is a java method in a class with an annotation above it. An annotation followed by the pattern is used to link the *Step Definition* to all the matching *Steps*, and the *code* is what *Cucumber* will execute when it sees a *Gherkin Step*. *Cucumber* finds the *Step Definition* file with the help of the Glue code in ***Cucumber Options***.

What is Cucumber Options?

@CucumberOptions **are like property files or settings for your test. Basically** *@CucumberOptions* **enables us to do all the things that we could have done if we have used cucumber command line. This is very helpful and of utmost importance, if we are using IDE such eclipse only to execute our project.**

```
package cucumberTest;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
                features = "Feature"
                ,glue={"stepDefinition"}
                )

public class TestRunner {

}
```

So in the above example, we have just set two different *Cucumber Options*. One is for *Feature File* and the other is for *Step Definition* file. We will talk about it in detail now but with this, we can say that *@CucumberOptions* are used to set some specific properties for the Cucumber test.

Following Main Options are available in Cucumber:

Options Type	Purpose	Default Value
dryRun	true: Checks if all the Steps have the Step Definition	false
features	set: The paths of the feature files	{}
glue	set: The paths of the step definition files	{}
tags	instruct: What tags in the features files should be executed	{}
monochrome	true: Display the console Output in much readable way	false
format	set: What all report formaters to use	false
strict	true: Will fail execution if there are undefined or pending steps	false

Dry Run

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

TestRunner Class

```
package cucumberTest;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
                features = "Feature"
                ,glue={"stepDefinition"}
                ,dryRun = true
                )

public class TestRunner {

}
```

Monochrome

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

TestRunner Class

```
package cucumberTest;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
                features = "Feature"
                ,glue={"stepDefinition"}
                ,monochrome = false
                )

public class TestRunner {

}
```

Features

Features Options helps *Cucumber* to locate the *Feature file* in the project folder structure. You must have notices that we have been specifying the *Feature Option* in the ***TestRunner*** class since the first

chapter. All we need to do is to specify the folder path and Cucumber will automatically find all the '*.features*' extension files in the folder. It can be specified like:

features = "Feature"

Or if the Feature file is in the deep folder structure

features = "src/test/features"

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

glue = "stepDefinition"

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

glue = "src/test/stepDeinition"

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

glue = "stepDefinition"

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

glue = "src/test/stepDeinition"

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

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

format = {"pretty"}

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

format = {"html:Folder_Name"}

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

format = {"json:Folder_Name/cucumber.json"}

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

format = { "junit:Folder_Name/cucumber.xml"}

Simple Program

Feature File

```
Feature: Application login
  Scenario: login
    Given User is on page
    When User login into the application
    Then Home Page is displayed
```

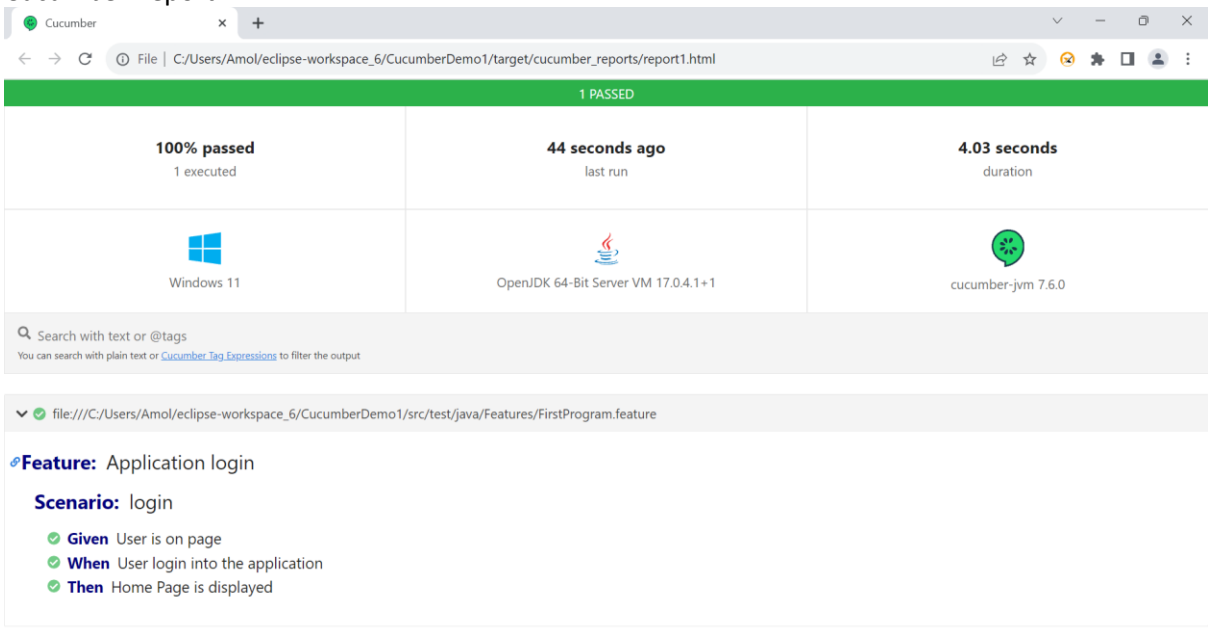
Step Definition File

```
package StepDefinition;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class Steps {
	WebDriver driver;
	@Given("User is on page")
	public void user_is_on_page() {
		System.setProperty("webdriver.chrome.driver",
"D:\\chromedriver_win32\\New4\\chromedriver.exe");
	    driver = new ChromeDriver();
		driver.get("https://www.saucedemo.com/");
	}
	@When("User login into the application")
	public void user_login_into_the_application() {
		WebElement username =
driver.findElement(By.xpath("//input[contains(@name,'user-name')]"));
		username.sendKeys("standard_user");
		WebElement password = driver.findElement(By.xpath("//input[@type =
'password']"));
		password.sendKeys("secret_sauce");
	}
	@Then("Home Page is displayed")
	public void home_page_is_displayed() {
		WebElement login = driver.findElement(By.xpath("//input[@type = 'submit']"));
		login.click();
	}
}
```

TestRunner File

```
package TestRunner;
import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
@RunWith(Cucumber.class)
@CucumberOptions(features = "C_Feature",
glue = "StepDefinition",
dryRun = false,
monochrome = true,
plugin = {"pretty", "html:target/cucumber_reports/report1.html"})

public class TestRunner {

}
```

Cucumber Report



Note : You can download this project from below GitHub URL

https://github.com/Dhanashree2023/Cucumber.git

Data Driven Testing Using Examples Keyword

Scenario Outline

In Gherkin language, scenario outline is the keyword which is used to run the same scenario multiple times.

It is also defined as "Scenario outlines are used when the same test is performed multiple times with a different combination of values."

Scenario outline is exactly similar to the scenario structure, but the only difference is the provision of multiple inputs. In order to use scenario outlines, we do not need any smart idea, we just need to copy the same steps and re-execute the code.

Example keyword can only be used with the *Scenario Outline* Keyword.

- ***Scenario Outline*** *- This is used to run the same scenario for 2 or more different sets of test data. E.g. In our scenario, if you want to register another user you can data drive the same scenario twice.*
- ***Examples****– All scenario outlines have to be followed with the Examples section. This contains the data that has to be passed on to the scenario.*

```
Feature: Login Action

Scenario Outline: Successful Login with Valid Credentials
        Given User is on Home Page
        When User Navigate to LogIn Page
        And User enters "<username>" and "<password>"
        Then Message displayed Login Successfully
Examples:
    | username   | password |
    | testuser_1 | Test@153 |
    | testuser_2 | Test@153 |
```

Program 2

Feature File

```
Feature: Login Action
Scenario: Sucessfull login with valid credentials
Given User is on Home Login Page
When User enters Email as "admin@yourstore.com" and Password as "admin"
And Click on Login
Then Page Title should be "Dashboard / nopCommerce administration"


Scenario Outline: Sucessfull login with valid credentials with DDT
Given User is on Home Login Page
When User enters Email as "<email>" and Password as "<password>"
And Click on Login
```

```
Then Page Title should be "Dashboard / nopCommerce administration"
Examples:
|email|password|
|admin@yourstore.com|admin|
|test@yourstore.com|admin|
```

Step Definition File

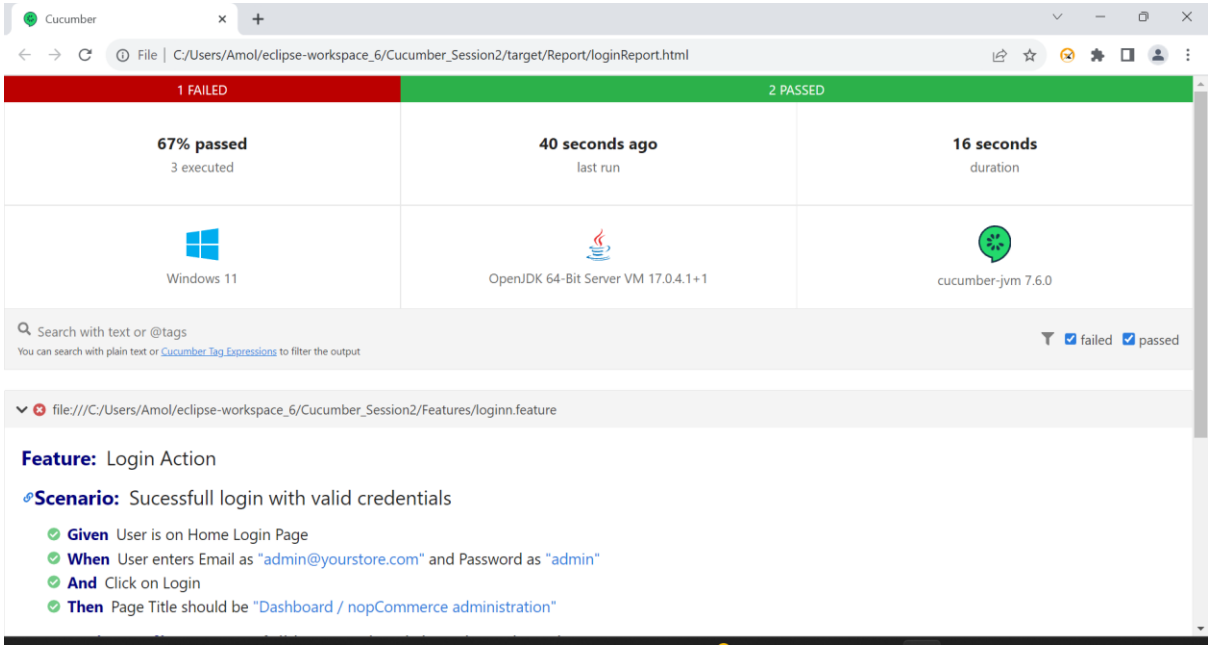
```
package StepDefinition;
import org.junit.Assert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class Steps {
	WebDriver driver;
	@Given("User is on Home Login Page")
	public void user_is_on_home_login_page() {
		System.setProperty("webdriver.chrome.driver", ".\\ChromeDriver\\chromedriver.exe");
	   driver = new ChromeDriver();
		driver.get("http://admin-demo.nopcommerce.com/login");
	}

	@When("User enters Email as {string} and Password as {string}")
	public void user_enters_email_as_and_password_as(String username, String password) {
		WebElement user = driver.findElement(By.xpath("//input[@type = 'email']"));
		user.clear();
		user.sendKeys(username);
		WebElement pwd = driver.findElement(By.xpath("//input[@type = 'password']"));
		pwd.clear();
		pwd.sendKeys(password);
	}

	@And("Click on Login")
	public void click_on_login() {
		WebElement login = driver.findElement(By.xpath("//button[@type = 'submit']"));
		login.click();
	}

	@Then("Page Title should be {string}")
	public void page_title_should_be(String ExpectedTitle) {
		String actualTitle = driver.getTitle();
		System.out.println("title is = "+actualTitle);
		if(actualTitle.equals(ExpectedTitle))
		{
			Assert.assertTrue(true); //pass
		}
		else
		{
			Assert.assertTrue(false); // Fail
		}
	}
}
```

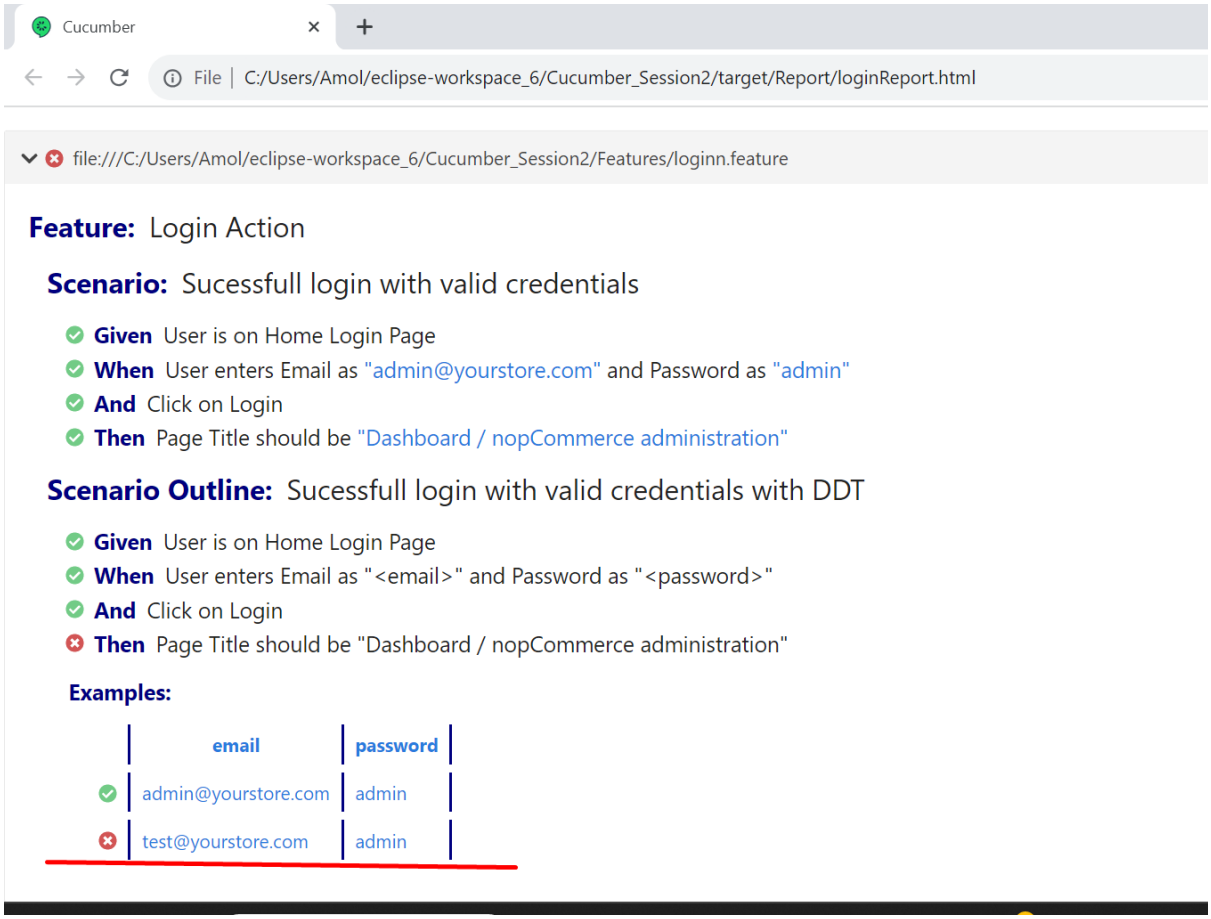
Test Runner File

```
package TestRunner;
import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.class)
@CucumberOptions(features = "Features",
```

```
glue = "StepDefinition",
dryRun = false,
monochrome = true,
plugin = {"pretty", "html:target/Report/loginReport.html","html:target/Report/loginReport.xml","html:target/Report/loginReport.json"})
public class TestRunner {

}
```

Report





Note: You can Download this project from below location
https://github.com/Dhanashree2023/Cucumber_Session2.git

1) What is Cucumber? What are the advantages of Cucumber?

To run functional tests written in a plain text Cucumber tool is used. It is written in a Ruby programming language.

Advantages of Cucumber are:

- You can involve business stakeholders who cannot code
- End-user experience is a priority
- High code reuse

2) What are the two files required to execute a Cucumber test scenario?

Two files required to execute a Cucumber test scenario are

- Features
- Step Definition

3) Give an example of a behavior is driven test in plain text?

- **Feature:** Visit **XYZ** page in abc.com
- **Scenario:** Visit abc.com
- **Given:** I am on abc.com
- **When:** I click on XYZ page
- **Then:** I should see ABC page

4) What is the usage of a Scenario Outline in the Cucumber tool?

In Cucumber, a Scenario outline is used as a parameter of scenarios. This is used when the same scenario needs to be executed for multiple sets of data; however, the test steps remain the same. Scenario Outline must be followed by the keyword 'Examples', which specify the set of values for each parameter

5) Explain the term step definition in Cucumber

A step definition is the actual code implementation of the feature mentioned in the feature file.

6) What is BDD?

BDD or Behavior-driven development is a process of developing software based on TDD (Test Driven Development) which focusses on the behavioral specification of software testing units.

7) What is a Feature file?

Features file contain a high-level description of the Test Scenario in simple language. It is known as Gherkin which is a plain English text language. Feature File consists of the following components like:

- Feature: It describes the current test script which has to be executed.
- Scenario: It is steps and expected outcome for a specific test case.
- Scenario outline: Scenario can be executed for multiple sets of data using scenario outline.
- Given: It specifies the context of the text to be executed.
- When: specifies the test action which has to perform.
- Then: Expected outcome of the test can be represented by “Then”

8) Why use Cucumber with Selenium?

Cucumber and Selenium are two popular technologies. Many organizations use Selenium for functional testing. These organizations which are using Selenium want to integrate Cucumber with Selenium as Cucumber helps you to read and to understand the application flow.

9) Advantages of Cucumber

Here, are some prominent advantages of using Cucumber.

- It is helpful to involve business stakeholders who can’t easily read the code
- Cucumber Testing enhances the end-user experience
- Style of writing tests allow for easier reuse of code in the tests
- Allows quick and easy setup and execution

10) What is a Step Definition?

Step definition maps the Test Case Steps in the feature files to code. It executes the steps on Application Under Test and checks the outcomes against expected results. In order to execute step definition it must match the given component in a feature.