**Cucumber is a testing tool that supports Behavior Driven Development (BDD) framework. It defines application behavior using simple English text, defined by a language called Gherkin.**

Cucumber allows automation functional validation that is easily read and understood. **Cucumber was initially implemented in Ruby and then extended to Java framework**. Both the tools support native JUnit.

**Behavior Driven Development gives us an opportunity to create test scripts** from both the developer’s and the customer’s perspective as well. So in the beginning, developers, project managers, QAs, user acceptance testers and the product owner (stockholder), all get together and brainstorm about which test scenarios should be passed in order to call this software/application successful. This way they come up with a set of test scenarios. **All these test scripts are in simple English language**, so it serves the purpose of documentation also.

## Example

If we are developing a user authentication feature, then the following can be few key test scenarios, which needs to get passed in order to call it a success.

* The user should be able to login with correct username and correct password.
* The user should not be able to login with incorrect username and correct password.
* The user should not be able to login with correct username and incorrect password.

## How it Works

By the time the code is ready, test scripts are ready too. The code has to pass the test scripts defined in BDD. If it does not happen, code refactoring will be needed. Code gets freezed only after successful execution of defined test scripts.



Cucumber is one such open source tool, which supports behavior driven development. To be more precise, Cucumber can be defined as a testing framework, driven by plain English text. It serves as documentation, automated tests, and a development aid – all in one.

Cucumber reads the code written in plain English text (Language Gherkin – to be introduced later in this tutorial) in the feature file (to be introduced later).

It finds the exact match of each step in the step definition

The piece of code to be executed can be different software frameworks like Selenium, Ruby on Rails, etc. Not every BDD framework tool supports every tool.

This has become the reason for Cucumber's popularity over other frameworks, like JBehave, JDave, Easyb, etc.

**Cucumber supports over a dozen different software platforms like** −

* Ruby on Rails
* Selenium
* PicoContainer
* Spring Framework
* Watir

## Advantages of Cucumber Over Other Tools

* **Cucumber supports different languages like Java.net and Ruby.**
* It acts as a bridge between the business and technical language. We can accomplish this by creating a test case in plain English text.
* **It allows the test script to be written without knowledge of any code, it allows the involvement of non-programmers as well.**
* It serves the purpose of end-to-end test framework unlike other tools.
* **Due to simple test script architecture, Cucumber provides code reusability.**

Dependencies for Cucumber:

Step 1 − Add dependency for **selenium**: This will indicate Maven which Selenium jar files are to be downloaded from the central repository to the local repository.

* Open pom.xml is in the edit mode, create dependencies tag (<dependencies></dependencies>), inside the project tag.
* Inside the dependencies tag, create dependency tag (<dependency></dependency>).
* Provide the following information within the dependency tag.

Step 2 − Add dependency for **Cucumber-Java**: This will indicate Maven, which Cucumber files are to be downloaded from the central repository to the local repository.

* Create one more dependency tag.
* Provide the following information within the dependency tag

Step 3 − Add dependency for **Cucumber-JUnit**: This will indicate Maven, which Cucumber JUnit files are to be downloaded from the central repository to the local repository.

* Create one more dependency tag.
* Provide the following information within the dependency tag

Step 4 − Add dependency for **JUnit**: This will indicate Maven, which JUnit files are to be downloaded from the central repository to the local repository.

* Create one more dependency tag.
* Provide the following information within the dependency tag.

Step 5 − Verify binaries.

* Once pom.xml is edited successfully, save it.
* Go to Project → Clean − It will take a few minutes.

The language, in which this executable feature files is written, is known as Gherkin. Gherkin is a plain English text language, which helps the tool - Cucumber to interpret and execute the test scripts.

One may think that, it has been discussed many times that Cucumber supports simple English text then why we need a separate language - Gherkins.

As discussed earlier, we had seen that BDD incorporates different prospectives while creating test scripts. It can be development prospective, business prospective, etc. That said, we will need people from different community like developers, project managers, product owners, and testers while developing test scripts. As these people do not belong to the same category, there is a risk of not using the common language for test script conceptualizing. This was the evolution point for Gherkins.

Gherkin provides the common set of keywords in English text, which can be used by people amongst the different community and yet get the same output in the form of test scripts.

Another interesting fact about Gherkin is, it supports not only English but many other native languages such as French, Finnish, Indonesian, Hungarian, Hindi, Urdu, Gujarati, etc.

**A Feature can be defined as a standalone unit or functionality of a project**. Let’s take a very common example of a social networking site. How does the feature of this product/project look like? Few basic features can be determined as −

* Create and remove the user from the social networking site.
* User login functionality for the social networking site.
* Sharing photos or videos on the social networking site.
* Sending a friend request.
* Logout.

By now, it is clear that, each independent functionality of the product under test can be termed as a feature when we talk about Cucumber. It is a best practice later when you start testing, that before deriving the test scripts, we should determine the features to be tested.

A feature usually contains a list of scenarios to be tested for that feature. A file in which we store features, description about the features and scenarios to be tested is known as Feature File. We will see more about feature files in the following chapter.

The keyword to represent a feature under test in Gherkins is “Feature”. The suggested best practice is to write a small description of the feature beneath the feature title in the feature file. This will fulfill the need of a good documentation as well.

### Example

Feature − Login functionality for a social networking site.

The user should be able to login into the social networking site if the username and the password are correct.

The user should be shown the error message if the username and the password are incorrect.

The user should be navigated to the home page, if the username and password are correct.

## **Feature Files**

**The file, in which Cucumber tests are written, is known as feature files. It is advisable that** there should be a separate feature file, for each feature under test. The extension of the feature file needs to be “.feature”.

One can create as many feature files as needed. To have an organized structure, each feature should have one feature file.

For Example −

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Feature** | **Feature File name** |
| 1 | User Login | userLogin.feature |
| 2 | Share the Post | sharePost.feature |
| 3 | Create Account | createAccount.feature |
| 4 | Delete Account | deleteAccount.feature |

The naming convention to be used for feature name, feature file name depends on the individual’s choice. There is no ground rule in Cucumber about names.

A simple feature file consists of the following keywords/parts −

* **Feature** − Name of the feature under test.
* **Description (optional)** − Describe about feature under test.
* **Scenario** − What is the test scenario.
* **Given** − Prerequisite before the test steps get executed.
* **When** − Specific condition which should match in order to execute the next step.
* **Then** − What should happen if the condition mentioned in WHEN is satisfied.

### Example

Feature − User login on social networking site.

The user should be able to login into the social networking site when the username and the password are correct.

The user should be shown an error message when the username and the password are incorrect.

The user should be navigated to the home page if the username and the password are correct.

Outline − Login functionality for a social networking site.

The given user navigates to Facebook. When I enter Username as "<username>" and Password as "<password>". Then, login should be unsuccessful.

| username | password |

| username1 | password1 |

\* AND keyword is used to show conjunction between two conditions. AND can be used with any other keywords like GIVEN, WHEN and THEN.

There are no logic details written in the feature file.

Annotation is a predefined text, which holds a specific meaning. It lets the compiler/interpreter know, what should be done upon execution. Cucumber has got the following few annotations −

* Given −
  + It describes the pre-requisite for the test to be executed.
  + Example − GIVEN I am a Facebook user
* When −
  + It defines the trigger point for any test scenario execution.
  + Example − WHEN I enter "<username>"
* Then −
  + Then holds the expected result for the test to be executed.
  + Example − THEN login should be successful.
* And −
  + It provides the logical AND condition between any two statements. AND can be used in conjunction with GIVEN, WHEN and THEN statement.
  + Example − WHEN I enter my "<username>" AND I enter my "<password>"
* But −
  + It signifies logical OR condition between any two statements. OR can be used in conjunction with GIVEN, WHEN and THEN statement.
  + Example − THEN login should be successful. BUT home page should not be missing.
* Scenario −
  + Details about the scenario under the test needs to be captured after the keyword “Scenario:”
  + Example –
  + Feature – Validating Login across W3Schools  
    Scenario: Validating Login with valid cerentials  
    GIVEN I am a Facebook user  
    WHEN I enter my username   
    AND I enter my password  
    THEN login should be successful.  
    BUT home page should not be missing.
* Scenario Outline − (To be covered later)
* Examples − (To be covered later)
* **Background −**
  + **Background generally has the instruction on what to setup before each scenario runs.** However, it gets executed after “Before” hook (to be covered later). So this is ideal to be used for code when we want to set up the web-browser or we want to establish the database connectivity.
    - Example −  
      **Background:  
      Go to Facebook home page.**

Following Main Options are available in Cucumber Runner file:

