Cucumber:
Behavioral Driven Testing 101

Presented by:
Dounya Gourja
Imane Benrazzouk

Supervised by:
Pr. NAFIL KHALID

What is Cucumber?
Why was it developed?

Cucumber testing is a software testing process that deals with an application's behavior. It operates in a world of behavior-driven development (BDD).
But what does that mean?

Behavioral Driven Development

Behavior-Driven Development (BDD):

- BDD is a way of testing that focuses on how our software should behave.
- Instead of just testing features, BDD explores the behavior of the application.
- It's like telling a story: 'When this happens, the software should do that.'

In a nutshell, BDD with Cucumber is about making testing accessible and easy to understand. It's like telling the story of our software's behavior in a language everyone can follow.

Tests become scenarios: 'Given this situation, when I do this, then this should happen.'

01. Improved communication among team members

02. Reduced need for lengthy discussions about the code

03. Easier to change than traditional code-based tests

Cucumber testing

Traditional testing

Key of differences

Cucumber testing is a behavior-driven development (BDD) approach to testing software.

Tests are written in a simple, natural language that anyone can understand.

VS

Traditional testing approaches are typically code-based. This makes them more challenging to change and maintain.

How do I get started with Cucumber testing ?

“ Getting started is easy, and it involves a few simple actions.”

Setup Steps

1. Setup a Java Project

Create a new Java project using your preferred IDE (Eclipse, IntelliJ, etc.) or use a simple text editor to write your Java code. Make sure you have Java installed on your machine.

Setup Steps

```
<dependencies>
<dependency>
<groupId>io.cucumber</groupId>
<artifactId>cucumber-java</artifactId>
<version>7.15</version> <!-- Use the latest version available -->
<scope>test</scope>
</dependency>
<dependency>
<groupId>io.cucumber</groupId>
<artifactId>cucumber-junit</artifactId>
<version>7.15</version> <!-- Use the latest version available -->
<scope>test</scope>
</dependency>
</dependencies>
```

2. Adding Cucumber dependencies

Add the following dependencies to your project's pom.xml file if you're using Maven:

Setup Steps

3. Create a Feature File

Create a new file named signup.feature in the src/test/resources directory with the following content

```
# src/test/resources/signup.feature

Feature: Sign up for an account

  Scenario: Successful signup
    Given I am on the signup page
    When I fill out the form with my information
    Then I should see a confirmation message
```

This feature file will house your scenarios and is written in Gherkin* syntax.

***Gherkin** is a human-readable language specifically designed for expressing scenarios in Cucumber testing. In this example The Gherkin syntax makes it clear that the user needs to be on the signup page, enter valid credentials, and, as a result, should be logged in successfully.

Setup Steps

4. Write Step Definitions

Create a step definitions class, for example, StepDefinitions.java

Write step definitions for each Gherkin step using Cucumber annotations (@Given, @When, @Then).

These step definitions contain the actual code to execute when each step is encountered.

Gherkin

```
// src/test/java/StepDefinitions.java

import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;

public class StepDefinitions {

    @Given("I am on the signup page")
    public void iAmOnTheSignupPage() {
        // Implementation for navigating to the signup page
    }

    @When("I fill out the form with my information")
    public void iFillOutTheFormWithMyInformation() {
        // Implementation for filling out the form
    }

    @Then("I should see a confirmation message")
    public void iShouldSeeAConfirmationMessage() {
        // Implementation for verifying the confirmation message
    }
}
```

Setup Steps

5. Run Cucumber Tests

Run your Cucumber tests using the JUnit runner. You can do this from your IDE or use the command line. If you're using Maven, you can run the tests with the following command:

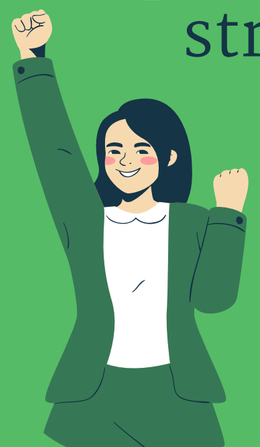
mvn test

6. Explore Test Results

- After running the tests, Cucumber provides feedback on the success or failure of each step.
- Review the results in the console output or explore detailed reports generated by Cucumber.

That's it!

You've now successfully created and executed a simple Cucumber test using Java on Windows. Adjust the versions and file paths according to your project structure and requirements.



For a deeper understanding and hands-on practice, we encourage you to explore more in our lab session. In the lab, you'll have the opportunity to delve into real-world scenarios, refining your skills and gaining practical experience with Cucumber testing.

Conslusion

In summary, Cucumber testing combines human-readable scenarios with executable code, fostering collaboration.
Gherkin, the language of Cucumber, bridges the gap between technical and non-technical team members.
Today, we've covered the basics of Cucumber, emphasizing the significance of Gherkin.
Join our upcoming lab session for a hands-on experience and deeper exploration.