Cahier de TP « Cucumber»

Pré-requis

- Système d'exploitation recommandé : Linux ou Windows 10
- JDK11+
- Git
- Système de build Maven
- IntelliJ IDEA, Eclipse avec Lombok

Atelier 1 : Hello Cucumber

Illustration des différentes étapes de mises au point d'une suite de tests Cucumber pour un projet Java

1.1 Création de projet via Maven

```
mvn archetype:generate                          \
   "-DarchetypeGroupId=io.cucumber"              \
   "-DarchetypeArtifactId=cucumber-archetype" \
   "-DarchetypeVersion=7.11.1"                  \
   "-DgroupId=hellocucumber"                     \
   "-DartifactId=hellocucumber"                  \
   "-Dpackage=hellocucumber"                     \
   "-Dversion=1.0.0-SNAPSHOT"                    \
   "-DinteractiveMode=false"
```

Intégration des tests Cucumber dans la phase de test de Maven

```
cd hellocucumber

mvn test
```

1.2 Mise au point d'un scénario

Reprendre le fichier *.feature* fourni et le copier dans ***src/test/resources/hellocucumber***

```
mvn test
```

Visualiser les erreurs rencontrées

Éditer le fichier de définitions d'étapes ***src/test/java/hellocucumber/StepDefinitions.java***

et y mettre les snippets proposés lors de la commande précédente
Exécuter à nouveau :

```
mvn test
```

Observer les différences au niveau des erreurs rencontrées

Modifier le fichier ***StepDefintions.java*** comme suit :

```
package hellocucumber;

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

```
import static org.junit.jupiter.api.Assertions.assertEquals;

class IsItFriday {
  static String isItFriday(String today) {
    return null;
  }
}

public class StepDefinitions {

  private String today;
  private String actualAnswer;

  @Given("today is Sunday")
  public void today_is_Sunday() {
    today = "Sunday";
  }

  @When("I ask whether it's Friday yet")
  public void i_ask_whether_it_s_Friday_yet() {
    actualAnswer = IsItFriday.isItFriday(today);
  }

  @Then("I should be told {string}")
  public void i_should_be_told(String expectedAnswer) {
    assertEquals(expectedAnswer, actualAnswer);
  }

}
```

Réexécuter
`mvn test`

Modifier ***StepsDefinition.java*** afin que le test passe

Ajouter un deuxième scénario :

```
  Scenario: Friday is Friday
    Given today is Friday
    When I ask whether it's Friday yet
    Then I should be told "TGIF"
```

Ajouter une définition d'étape correspondant à « `today is Friday` »

1.3 Variables et exemples

Modifier le fichier de *feature* comme suit :
Observer la notion de variables et d'exemples

```
Feature: Is it Friday yet?
  Everybody wants to know when it's Friday

  Scenario Outline: Today is or is not Friday
    Given today is "<day>"
    When I ask whether it's Friday yet
    Then I should be told "<answer>"

  Examples:
    | day            | answer |
    | Friday         | TGIF   |
    | Sunday         | Nope   |
    | anything else! | Nope   |
```

Modifier les définitions d'étapes en ajoutant les étapes variabilisées suivantes :

```
@Given("today is {string}")
    public void today_is(String today) {
        this.today = today;
    }
```

1.4 Mise en pace IDE (Optionnel)

Récupérer une distribution Eclipse (ou l'IDE de votre choix), Installer le plugin Cucumber.

Sous Eclipse :

Help → Eclipse Market Place

Puis saisir : *cucumber*

Atelier 2. Gherkin

Le système cible est un magasin en ligne. Nous nous concentrons sur la fonctionnalité *Passer une commande*

Lors du passage d'une commande, le système cherche à identifier le client si il est connu, il lui évite de saisir les informations nécessaires à la livraison.

En fonction de l'adresse postale, du client différents modes de livraison sont possibles :

- Livraison en magasin toujours possible
- Livraison à domicile possible pour certains départements (> 50 et <99)
- Livraison expresse moins de 24h pour région parisienne, lilloise

2.1 Exploration via Feature Mapping

Identifier les acteurs et les tâches

Essayer de formaliser 2 exemples.

Vous pouvez utiliser l'outil https://cardboardit.com/ par exemple

Comparaison des solutions proposées par les stagiaires.

Visualiser la solution proposée

2.2 Syntaxe Gherkin

Le but de cette section est de rédiger (et préciser) les spécifications sous forme de feature(s) Gherkin

- Importer le projet Maven fourni contenant les dépendances et quelques sources :
  - Classes du modèle
  - Règle métier implémentée dans *SimpleLivraisonService.java*
- Écrire la ou les fonctionnalités
- Exécuter un *mvn test* et visualiser les snippets proposés par *Cucumber*

Affiner l'écriture de fonctionnalité en fonction des snippets proposés.

Mettre à jour *StepDefinitions.java* avec les implémentations proposées.

Atelier 3. Définitions d’étapes

3.1 1ère Implémentation

Récupérer les sources fournis comportant :

- Les classes du modèle de notre système
- L’implémentation de règle et service métier

Implémenter *StepDefinitions.java*

3.2 Réutilisation, lisibilité

Réécrire la fonctionnalité utilisant :

- **Contexte** : Démarrage du scénario « Passer une commande », initialisation dus sytème
- **Plan du scénario** : Permettant de lister de façon exhaustive les différents cas de figure pour les combinaisons code Postal / modes de livraison
- **@DataTableType** : Pouvoir convertir une dataTable en classe Client

3.3 Tags et Hooks

Pour les scénarios qui implique la validation de l’email, on aimerait pouvoir initialiser une base de données avec des données de test.

Du code source simulant une base est fournie.

- Tagger les scénarios qui nécessite un jeu de test
- Écrire un Hook qui initialise la base avec un jeu de test
- Réécrire les scénarios concernés

Via une commande Maven n’exécuter que ses scénarios

3.3 Reporting

Essayer les différentes options de reporting en modifiant les options JUnit ou via la ligne de commande :

- Format JUnit seulement
- Format pretty, html et json
- Services cloud de Cucumber

Atelier 4. Cas d'usages

Démarrer l'application et accéder aux URLs suivantes pour découvrir l'application:

- http://localhost:8080 (Application Web)
- http://localhost:8080/swagger-ui.html (API Rest applicative)
- http://localhost:8080/actuator/ (API Rest de monitoring)

Créer un projet avec la commande Maven suivante :

```
mvn archetype:generate                               \
   "-DarchetypeGroupId=io.cucumber"                    \
   "-DarchetypeArtifactId=cucumber-archetype" \
   "-DarchetypeVersion=5.4.0"                        \
   "-DgroupId=org.formation"                           \
   "-DartifactId=product-service"                        \
   "-Dpackage=org.formation"                           \
   "-Dversion=1.0.0-SNAPSHOT"                          \
   "-DinteractiveMode=false"
```

4.1 Rest API avec Karate

Ajouter les dépendances suivantes dans le fichier pom.xml

```
<dependency>
      <groupId>com.intuit.karate</groupId>
      <artifactId>karate-apache</artifactId>
      <version>0.6.0</version>
      <scope>test</scope>
</dependency>
<dependency>
      <groupId>com.intuit.karate</groupId>
      <artifactId>karate-junit4</artifactId>
      <version>0.6.0</version>
      <scope>test</scope>
</dependency>
```

Écrire une feature qui teste l'accès à la ressource http://localhost:8080/actuator/health. Par exemple :

```
Fonctionnalité: Ressources actuator


  Scénario: Health endpoint
    Etant donné url 'http://localhost:8080/actuator/health'
    Lorsque method GET
    Alors status 200
```

Ecrire la classe *KarateTest* permettant de démarrer les tests comme suit :

```
@RunWith(Karate.class)
@CucumberOptions(features = "classpath:org/formation/karate")
public class KarateTest {

}
```

Exécuter les tests.

Ensuite, écrire une fonctionnalité validant que toutes les URLS POST renvoient un 201 et un attribut json ***id***

4.2 Application web avec Selenium

Ajouter la dépendance suivante dans le *pom.xml*

```
<dependency>
     <groupId>org.seleniumhq.selenium</groupId>
     <artifactId>selenium-java</artifactId>
     <version>3.141.59</version>
</dependency>
```

Installer un driver pour un des navigateurs de votre système :
https://www.selenium.dev/documentation/fr/selenium_installation/installing_webdriver_binaries/

Écrire des scénarios et les étapes associées pour tester la fonctionnalité « ***Recherche de produits par fournisseur***. »

4.3 Intégration continue

Définir un profil Maven permettant de lancer Cucumber dans la phase ***integration-test.***
Tester en ayant démarrer l'application auparavant

Reprendre la distribution Jenkins fournie. Le démarrer avec le script fourni. (*jenkins.sh* ou *jenkins.bat*).
Se connecter en tant qu'administrateur avec ***admin/welcome1*** et vérifier que le plugin « ***Cucumber Report*** » est installé.
Créer une job de type « Free Style » :

- Configurer sur un dépôt git local ou
- Une tâche de build effectuant la tâche Maven précédente
- Une action de post-build qui utilise le plugin *CucumberReport*

Tester l'exécution du job Jenkins et visualiser les résultat Jenkins

4.4 Intégration Spring (Optionnel)

Récupérer les sources fournies, visualiser le fichier pom.xml et visualiser les tests unitaires de l’application.

Mettre au point une classe d’intégration *SpringIntegrationTest* annoter *@SprirngBootTest*

Exécuter les tests d’acceptation et bien comprendre l’intégration avec SpringBoot