Automatiser les tests fonctionnels avec JUnit

Livrables à déposer sur Ametice (voir la dernière page de ce document)

Outils et technologies utilisées

IDE : IntelliJ IDEA

Tests fonctionnels : JUnit, Selenium WebDriver API, Apache HttpClient

PARTIE 1 : Exécuter des tests fonctionnels Selenium WebDriver avec JUnit

JUnit permet de préparer et exécuter les tests fonctionnels réalisés avec Selenium. Selenium joue le test dans le navigateur et permet d'accéder au contenu de la page. Les assertions JUnit sont vérifiées sur le contenu fourni par Selenium, pour déterminer si le test est un succès.

1. Créer un premier test fonctionnel Selenium avec JUnit 5

- Créer une classe Java *YoutubeTest*

- Importer :

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
```

- Créer la méthode qui ouvre le navigateur entre chaque test

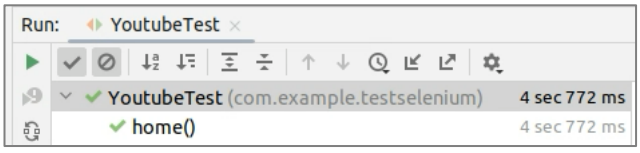
```
private WebDriver driver;

@BeforeEach // avant chaque test
public void openBrowser() {
    System.setProperty("webdriver.chrome.driver", "CHEMIN/chromedriver");
    driver = new ChromeDriver();
}
```

- Créer la méthode *quitBrowser* qui quitte le navigateur après chaque test

- Créer la méthode de test *home* qui charge la page https://www.youtube.com/ et qui utilise l'assertion *assertEquals* pour vérifier que le titre de la page est « YouTube ».

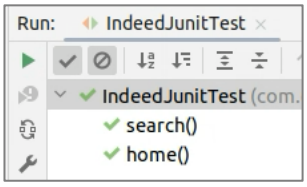
- Exécuter le test : le navigateur est ouvert, Selenium exécute les actions, l'assertion JUnit est vraie, le test est un succès :



2. Ecrire un test fonctionnel qui remplit et envoie un formulaire

- Créer un test *IndeedJunitTest*
- Importer les classes pour WebDriver et JUnit.
- Créer la méthode *openBrowser* qui lance le navigateur et charge l'URL d'indeed :

  https://fr.indeed.com
- Créer la méthode *quitBrowser* qui quitte le navigateur après chaque test.
- Créer la méthode *home* qui
  - Utilise une assertion pour vérifier si :
    - le titre de la page est celui attendu.
    - la page contient un lien « A propos ».
    - la page contient un lien dont une partie du texte est « Guide ».
  - Vérifie que la page contient le formulaire de recherche dont l'id est *jobsearch*. Utiliser la méthode *findElements*, pour obtenir une liste d'éléments. La liste sera vide si l'élément n'est pas trouvé ou de taille 1 si l'élément est trouvé. Vérifier la taille de la liste dans une assertion.
- Exécuter le test
- Créer la méthode *search* qui :



  - Entre la chaîne « développeur web » dans le champ input dont le nom est q.
  - Fait une tabulation
  - Entre la chaîne « Aix-en-Provence » dans le champ input dont le nom est l.
  - Envoie le formulaire.
  - Vérifie que la page de résultat contient un élément dont l'id est filter-radius.
- Exécuter le test

3. Récupérer et exécuter un test fonctionnel exporté depuis WebDriver

Ouvrir Selenium IDE dans chromium ou firefox :

- lancer l'enregistrement des actions suivantes dans la page web indeed : entrer la chaîne développeur web dans le champ Quoi, faire une tabulation, entrer la chaîne Aix-en-Provence dans le champ Où, cliquer sur le bouton pour envoyer le formulaire.
- Enregistrer le cas de test en langage Java JUnit.
- Ajouter le fichier Java généré dans le répertoire des tests.
- Le code généré est pour JUnit4. Modifier les imports pour exécuter le test avec JUnit5, modifier les noms des méthodes *Before* et *After*. Ajouter l'instruction définissant l'emplacement du pilote, puis exécuter le test.

Si une exception est levée lors du clic sur le bouton d'envoi du formulaire, il faudra cliquer préalablement sur le bouton pour accepter les cookies (si le bloc des cookies recouvre le bouton), ou fermer la fenêtre popup de connexion, si elle est présente. Avant de fermer le bandeau des cookies, utiliser *WebDriverWait* pour attendre que le bouton du bandeau soit cliquable après le chargement de la page :

https://www.selenium.dev/documentation/webdriver/waits/

PARTIE 2 : Créer des tests de fumée

Les tests de fumée sont des tests préliminaires qui garantissent les fonctionnalités de base du logiciel. Ils détectent des problèmes tels que l'échec de chargement de parties de l'application (codes HTTP 404, 5xx), l'absence de bouton d'envoi dans un formulaire, l'absence de menu dans la page…

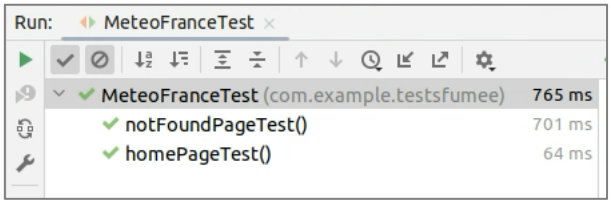
1. Créer un test de fumée avec Apache HttpClient

- Créer un test JUnit5 *MeteoFranceTest*.
- Créer la méthode privée *getStatusCode* qui reçoit en paramètre l'URL à tester et qui retourne le code de la réponse HTTP.

  Utiliser les instructions suivantes pour récupérer le code de la réponse HTTP :

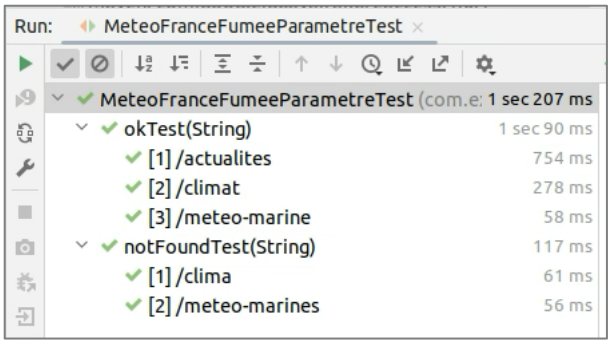
```
URL url = new URL("https://meteofrance.com/");
HttpURLConnection http = (HttpURLConnection)url.openConnection();
int code = http.getResponseCode();
```

- Créer la méthode de test *homePageTest* qui vérifie si le code de la réponse HTTP est 200 quand on charge la page d'accueil du site http://www.meteofrance.com/. Utiliser la méhode *getStatusCode* dans *homePageTest.*
- Créer la méthode de test *notFoundPageTest* pour vérifier que le code 404 est obtenu pour l'URL *https://meteofrance.com/meteo-marines*
- Exécuter le test



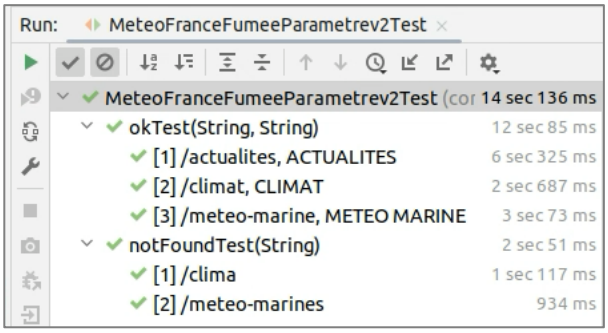
2. Créer un test de fumée paramétré

- Créer un test JUnit5 *MeteoFranceFumeeParametreTest*.
- Copier la méthode *getStatusCode* dans la classe, la modifier pour qu'elle construise l'URL à charger à partir de la propriété privée *baseUrl* et du chemin reçu en paramètre.
- Créer une méthode okTest qui reçoit en paramètre le chemin à tester à partir de la racine du site web. Le code HTTP attendu est 200. Utiliser une source CSV pour les paramètres. Tester la méthode avec les URL du menu du site : /actualites, /climat, /meteo-marine.
- Créer une méthode notFoundTest qui reçoit en paramètre le chemin à tester à partir de la racine du site web. Le code HTTP attendu est 404. Utiliser une source CSV pour les paramètres.
- Exécuter le test.



3. Créer des tests de fumée paramétrés avec Apache HttpClient et Selenium WebDriver

- Ajouter dans *MeteoFranceFumeeParametreTest* les méthodes *openBrowser* (ne pas charger d'URL dans cette méthode) et *quitBrowser*, qui seront respectivement exécutées avant et après chaque test.
- Modifier la méthode *okTest* :
    - Passer un 2e argument définissant le texte qui doit être contenu dans le titre de la page chargée.
    - Définir les valeurs dans les paramètres pour ce 2e argument. Par exemple, pour /actualites, l'élément *title* commence par « ACTUALITES », pour /meteo-marine, l'élément *title* commence par «METEO MARINE ».
    - Ouvrir la page avec Selenium WebDriver.
    - Récupérer le titre de la page.
    - Utiliser une assertion pour indiquer si la page contient bien le titre attendu.
- Exécuter le test de fumée paramétré. Le navigateur est lancé par Selenium pour chaque test.



Livrables à déposer sur Ametice en indiquant les noms du binôme

- Copies d'écran montrant l'exécution des tests de fumée paramétrés avec Apache HttpClient et Selenium.
- Fichier MeteoFranceFumeeParametreTest.

Références

Selenium WebDriver : https://www.selenium.dev/documentation/webdriver/

JUnit : https://junit.org/junit5/