TD outil de développement logiciel : test Web avec Selenium

Rendre un rapport au format pdf + code sous forme d'archive à la fin du TP sous Celene ; un rapport par étudiant

Consignes générales

- Tout ce qui est nécessaire au TP est installé dans la VM. Il y a un compte "sti" mot de passe "sti"
- Vous rendez vos rapports pour le contrôle continu sur Celene, avant minuit le jour du TP, sous forme d'archive contenant un rapport en pdf ainsi que tout code utile
- **Par défaut le clavier est en anglais,** `setxkbmap fr` **résoudra ce problème.**

Prologue au TP

Dans le précédent TPs, nous avons utilisé des outils de test orienté vers le développement. Il existe d'autres outils de test plutôt *boîte noire* qui permettent de tester des produits plus ou moins finis, grâce à des tests externes.

Selenium est l'outil libre le plus utilisé pour faire des tests sur du Web. Des solutions payantes équivalentes existent, ainsi que des outils libres qui sont des surcouches sur du Selenium. Dans ce TP nous allons utiliser Selenium pour tester un site Web, bien sûr dédié au LCM.

Selenium

Selenium s'organise autour de 3 outils différents :

- Selenium IDE, qui, sous forme d'add-on pour votre navigateur (firefox, chrome) vous sert à enregistrer des scenarii de consultation de site web, de rajouter des asserts, etc, pour ensuite les utiliser comme base pour des tests
- Selenium WebDriver s'appuie sur un serveur qui va s'interfacer avec d'autres machines pour qu'elles lancent tout ou partie des tests. Ca permet par exemple de tester différents browsers sous différents OS, de faire des tests de charge réaliste, tester la ségrégation de contenu, etc, etc.
- Enfin les Selenium bindings permettent de gérer les tests dans un programme classique que vous écrivez dans votre langage favori (pour ce TP, ce sera Java).

NOTA BENE : En 2018, Selenium vient d'abandonner le support de Selenium IDE pour firefox et se tourne vers une réécriture complète de son plugin pour le mettre sur Chrome. Pour l'instant (mars 2018) le plugin Chrome ne permet pas d'exporter en Java les tests suites, mais il y a toutes les raisons de croire que ce sera le cas avant la fin de l'année.

1 Première automatisation de tests avec Selenium IDE

Il va falloir dans un premier temps que vous lanciez le serveur nginx qui va héberger dans votre VM la page Web que vous voulez tester.

Tapez dans un terminal la commande suivante :

```
sudo service nginx start
```

Puis vérifiez que votre page préférée est disponible à l'adresse `http://localhost/lcm.html` depuis Chrome dans la VM.

L'IDE Selenium est installé sous Chrome dans la VM. Pour le lancer, vous devez cliquer en haut à droite sur l'icône gris "Se". Une interface s'ouvre alors pour enregistrer un test. Quand vous appuyez sur le bouton rouge, cela enregistre vos actions. Vous pouvez ensuite rejouer le test, et éventuellement ajouter des assertions, comme par exemple `verify value` qui vous permettra de savoir si une valeur est conforme à vos attentes. **Sauvegardez les tests suivants pour vous en inspirer plus tard.**

Question Ecrivez un test qui vérife que le calcul du LCM fonctionne, donnez une copie d'écran dans votre rapport.

Question Pensez-vous pouvoir être capable de couvrir toutes les branches dans vos tests sans recourir à la lecture du code de la page Web ? Justifiez.

Question Ecrivez un test qui vérifie que le lien vers le jar est bien présent sur la page et donnez la copie d'écran.

Question Ecrivez un test qui vérifie que le jar est disponible à la bonne adresse et donnez la copie d'écran.

2 Ecriture d'une test suite à base de Selenium

Il existe un autre moyen d'utiliser Selenium, en automatisant les tests depuis Java.

Pour cela, le programme de test va utiliser un *driver* qui va prendre en main un de vos navigateur pour jouer la séquence d'actions que vous voulez faire faire.

On va voir 4 nouvelles annotations JUnit en plus du test : @Before et @After, @BeforeClass et @AfterClass. Chaque méthode qui est annoté @Before sera lancée avant chaque test. @After sera lancé après chaque test. Cela permet par exemple de nettoyer après un test qui a potentiellement généré des fichiers temporaires. @BeforeClass et @AfterClass correspondent elles a des actions qui ne seront exécuté qu'une fois, avant ou après l'exécution de tous les tests.

Action Mettez en place un nouveau projet Maven sous Eclipse. Il doit dépendre de Selenium et de JUnit, donc, quelque part, vous devrez inclure ceci :

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.11.0</version>
    <scope>test</scope>
</dependency>
</dependencies>
```

Question Faites une test suite qui teste :

— Que le titre est correct
— Que le lien vers le jar est bien présent
— Que les valeurs calculées sont correctes, en utilisant les valeurs que vous renvoie la méthode correcte de johnArihtmetics.

Vous pourrez pour cela vous inspirer du code donné en annexe. Ajoutez le code à votre archive.

Question D'après vous, quel est l'intérêt de tester le comportement d'une page Web de cette façon plutôt qu'en envoyant directement des requêtes http et en analysant le contenu de la réponse ?

Annexe

```
import java.util.concurrent.TimeUnit;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
```

```
import org.openqa.selenium.support.ui.Select;


public class TestSelenium {
  private WebDriver driver;
  private String baseUrl;
  private StringBuffer verificationErrors = new StringBuffer();

  @Before
  public void setUp() throws Exception {
    // On instancie notre driver, et on configure notre temps d'attente
System.setProperty("webdriver.chrome.driver","/home/lbobelin/Downloads/chromedriver");
driver = new ChromeDriver();
    baseUrl = "http://localhost/";
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
  }

  @Test
  public void testSelenium() throws Exception {
    // On se connecte au site
    driver.get(baseUrl + "lcm.html");

    // Bon, comment est le titre ?
    String expectedTitle = "Welcome to LCM calculator !";
    String actualTitle = "";

    // get the actual value of the title
    actualTitle = driver.getTitle();

    // Remplissons le formulaire
    driver.findElement(By.id("inputText")).clear();
    driver.findElement(By.id("outputText")).clear();
    driver.findElement(By.id("inputText")).sendKeys("3, 4");
    driver.findElement(By.id("button1")).click();

    // Quelle est donc cette valeur ? Vérifions avec notre bonne version d'Arithmetics !
    String maybeLCM = driver.findElement(By.id("outputText")).getAttribute("value");
    System.out.println(maybeLCM + " is the value calculated by the web page");

    }

  @After
  public void tearDown() throws Exception {
    driver.quit();
    String verificationErrorString = verificationErrors.toString();
    if (!"".equals(verificationErrorString)) {
     // fail(verificationErrorString);
    }
  }
}
```