Selenium

Framework d'Automatisation des Tests Web

Testing • Automatisation • WebDriver

Plan du Cours

- Introduction
- Architecture WebDriver
- Prise en main & Premières interactions
- Sélection d'éléments (XPath vs CSS)
- Actions simples
- Attentes (Implicites & Explicites)
- Actions complexes
- POM (Page Object Model)

Introduction

- Framework **open-source** pour automatiser les tests web
- Langages supportés: **Java, Python, C#, JavaScript, Ruby**
- Navigateurs: **Chrome, Firefox, Safari, Edge, Opera**
- Simule les interactions d'un **utilisateur réel**
- Standard **W3C WebDriver Protocol** pour la communication
- **Gratuit** et largement adopté dans l'industrie

Architecture WebDriver

Les 4 composants principaux

- API disponible en : **Java, Python, C#, JavaScript, Ruby**
- **W3C WebDriver Protocol :** Protocole de communication standard
- **WebDriver Server :** Service qui reçoit et exécute les commandes
- **Browser Driver :** ChromeDriver, GeckoDriver, EdgeDriver, etc.

Inbtégrer Selenium

Dépendances dans le pom.xml

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.38.0</version>
</dependency>

<dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>5.7.0</version>
</dependency>
```

Première Classe de Test

```
import io.github.bonigarcia.wdm.WebDriverManager;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class BaseTest {

    private WebDriver driver;

    @BeforeEach
    public void setUp() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize();
    }

    @AfterEach
    public void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }

}
```

- @BeforeEach : attribut Junit pour une méthode appelée avant l'exécution des tests

- @AfterEAch = attribute Junit pour une méthode appelée après l'execution des tests

DOM : accéder à une page

Méthodes de base

```
// Charger une URL
driver.get("https://www.example.com");

// Naviguer (avec historique)
driver.navigate().to("https://www.example.com");
driver.navigate().back();
driver.navigate().forward();
driver.navigate().refresh();

// Informations du navigateur
String title = driver.getTitle();
String currentUrl = driver.getCurrentUrl();
```

DOM : locators “By”

```
// Trouver un seul élément
WebElement element = driver.findElement(By.id("myId"));

// Trouver plusieurs éléments
List<WebElement> elements =
driver.findElements(By.className("myClass"));

// Les 8 locators disponibles
By.id("elementId");
By.name("elementName");
By.className("myClass");
By.tagName("input");
By.linkText("Click Here");
By.partialLinkText("Click");
By.cssSelector("div.classname");
By.xpath("//div[@id='myId']");
```

Sélection d'Éléments - CSS Selectors

```
button              // tag
.submitBtn          // class
#myId               // id
button.primary      // tag + class

// Attributs
input[type="email"]
input[name="username"]
button[disabled]

// Pseudo-classes
input:first-child
button:last-of-type
li:nth-child(2)
li:nth-of-type(odd)
```

- input:first-child : le premier “input” enfant de leur parent
- button:last-of-type : le dernier élément “button” parmis ses frères directs
- li:nth-child(2) : sélectionne les éléments “li” qui sont le deuxième enfant de leur parent
- li:nth-of-type(odd) : le premier “input” enfant de leur parent

Sélection d'Éléments – Xpath basiques

```
// Chemin absolu : 1er bouton de la page
/html/body/div/button

// Chemin relatif : bouton dont l'id est vaut "submitBtn"
//button[@id='submitBtn']

// Attributs
//input[@type='email'] // récupère le 1er input de type email
//input[@name='username'] // récupère l'input de nom "username"
//div[@class='container'] // récupère la 1ère div ayant la classe "container"

// Text nodes
//button[text()='Click Me'] // récupère le 1er bouton avec le texte "Click Me"
//button[contains(text(), 'Click')] // récupère le 1er bouton dont le texte contient "Click"


// Index
//tr[1] // récupère le 1er "tr"
//tr[position()>2] // récupère les les lignes après la 2ème

// Combinaisons
//form//input[@type='text' and @required] // le 1er input text + required dans un form
```

Sélection d'Éléments - XPath Avancé

```
// Parent navigation
//button/ancestor::form // Le formulaire contenant le bouton
//input/parent::div // La div parente de l'input

// Siblings
//button/following-sibling::span // Tous les "span" qui ont le même parent que le button
//input/preceding-sibling::label // Tous les "label" qui sont des frères ainés à l'input

// Descendants
//form/descendant::input // Tous les "input" qui sont descendants de n'importe quel form
//div//p // Tous les "p" qui sont enfants, direct ou non, dans une div

// Wildcards
//*[@class='error'] // N'importe quel élément qui possède l'attribut class de valeur "error"
//*/button // Tous les "button"

// Functions
//button[starts-with(@id, 'btn_')] // Tous les "button" dont l'id commence par "btn_"
//input[contains(@placeholder, 'Enter')] // Tous les "input" dont le placeholder contient "Enter"
//span[normalize-space()='Username'] // Tous les "span" dont le texte, après avoir supprimé les
espaces, contient "Username"
```

XPath vs CSS - Comparaison

Critère	XPath	CSS
Navigation ascendante	Oui	Non
Performance	Lente	Rapide
Complexité	Plus complexe	Simple
Contenu spécifique	Oui	Non

Recommandation : Utiliser CSS quand possible (performance), XPath pour cas spéciaux (navigation ascendante, contenu spécifique)

DOM : locators “actions simples”

```
WebElement textField = driver.findElement(By.id("username"));
WebElement button = driver.findElement(By.id("sendForm"));

// Texte et input
textField.sendKeys("mon_utilisateur");
textField.clear();

// Clics et soumission
button.click();
textField.submit();

// Récupérer des informations
String text = textField.getText();
String attribute = textField.getAttribute("placeholder");
boolean isDisplayed = textField.isDisplayed();
boolean isEnabled = textField.isEnabled();
```

« sendKeys » simule l’action dans le form

Attentes - Waits Implicites

```
@BeforeEach
public void setUp() {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
}
```

Permet d'imposer une temporisation au « driver », permettant ainsi de laisser le temps aux éléments d'apparaître sur la page

Attentes - Waits Explicites (Recommandé)

Instancie le « WebDriverWait » :

```
WebDriverWait  wait = new WebDriverWait(driver, Duration.ofSeconds(10));
```

Attends que l'élément soit visible :

```
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("username")));
```

Attends que l'élément soit cliquable :

```
wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("button[type='submit']")));
```

```
protected WebElement waitForVisible(By locator) {
   return wait.until(ExpectedConditions.visibilityOfElementLocated(locator));
}
```

Attentes - Conditions Courantes

```
// Elements
wait.until(ExpectedConditions.elementToBeClickable(By.name("username")));
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("username")));
wait.until(ExpectedConditions.invisibilityOfElementLocated(By.name("username")));
wait.until(ExpectedConditions.presenceOfElementLocated(By.name("username")));

// Texte
wait.until(ExpectedConditions.textToBePresentInElement(element, "Text"));

// URL & Titre
wait.until(ExpectedConditions.urlContains("/page"));
wait.until(ExpectedConditions.titleContains("Title"));
wait.until(ExpectedConditions.titleIs("Exact title"));

// Attributes
wait.until(ExpectedConditions.attributeContains(element, "class", "active"));
```

Actions Complexes - Remplir des Formulaires

```
WebElement emailField = driver.findElement(By.id("email"));
WebElement passwordField = driver.findElement(By.id("password"));
WebElement rememberMe = driver.findElement(By.id("rememberMe"));

// Fill up the form
emailField.sendKeys("user@example.com");
passwordField.sendKeys("verygoodpassword123");

// Check a checkbox / radio
rememberMe.click();

// Type a specific key
passwordField.sendKeys(Keys.ENTER);
```

Actions Complexes - Menus Déroulants (Select)

```
WebElement dropdown = driver.findElement(By.id("country"));
Select select = new Select(dropdown);

// Select by visible text
select.selectByVisibleText("France");

// Select by value
select.selectByValue("FR");

// Select by index
select.selectByIndex(0);

// Select multiple (if allowed)
select.selectByVisibleText("Option 1");
select.selectByVisibleText("Option 2");

// Unselect
select.deselectByVisibleText("Option 1");
select.deselectAll();

// Get the selected options
List<WebElement> selected = select.getAllSelectedOptions();
```

Actions Complexes - Clics Avancés & Hover

```
Actions actions = new Actions(driver);
WebElement element = driver.findElement(By.id("myElement"));
WebElement menu = driver.findElement(By.id("menu"));

// Right click
actions.contextClick(element).perform();

// Double click
actions.doubleClick(element).perform();

// Hover
actions.moveToElement(menu).perform();

// Drag and Drop
WebElement source = driver.findElement(By.id("draggable"));
WebElement target = driver.findElement(By.id("droppable"));
actions.dragAndDrop(source, target).perform();
```

Actions Complexes - Fenêtres & Pop-ups

```
// Handle windows
String mainWindowHandle = driver.getWindowHandle();
Set<String> allHandles = driver.getWindowHandles();

for (String handle : allHandles) {
   driver.switchTo().window(handle);
}

driver.switchTo().window(mainWindowHandle);

// Handle les frames/iframes
driver.switchTo().frame(0);
driver.switchTo().frame("frameName");
driver.switchTo().frame(frameElement);
driver.switchTo().defaultContent();

// Handle les alertes
Alert alert = driver.switchTo().alert();
alert.accept();
alert.dismiss();
String alertText = alert.getText();
alert.sendKeys("input text");
```

POM (Page Object Model) - Concept

- **Selenium WebDriver** : Approche standard, universelle
- **Encapsulation :** Chaque page = 1 classe Java héritant de BasePage
- **Locators privés** : Les **By** sont déclarés comme attributs privés
- **Méthodes publiques :** représentent les actions utilisateurs
- **Séparation des responsabilités :** logique de chaque page différente de la logique de test
- **Maintenabilité :** modification du DOM = un seul endroit à modifier

POM – BasePage

- **Convention de design recommandée par la communauté Selenium pour implémenter le pattern « POM »**
- Cette classe va implementer les méthodes utilitaires réutilisables
  - waitUntil
  - waitClickable
  - type
  - getText
- Elle va avoir la gestion du WebDriver
- Les attentes explicites communes

POM – BaseTest implémentée

- Elle ne gère que le navigateur maintenant !

```
public class BaseTest {

    protected WebDriver driver;

    @BeforeEach
    protected void setUp() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize();
    }

    @AfterEach
    protected void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }

}
```

POM – BasePage implémentée

- Regroupement de méthodes utilitaires pour les réutiliser dans les tests
- Utilisation du « driver »
- Instanciation du « wait »
- Aucune méthode annotée de « **@Test** » !

```
public class BasePage {

    protected WebDriver driver;
    protected WebDriverWait wait;

    public BasePage(WebDriver driver) {
        this.driver = driver;
        this.wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    }

    protected void goTo(String url) {
        driver.get(url);
    }

    protected WebElement waitUntil(By locator) {
        return wait.until(ExpectedConditions.visibilityOfElementLocated(locator));
    }

    protected WebElement waitClickable(By locator) {
        return wait.until(ExpectedConditions.elementToBeClickable(locator));
    }

    protected String getText(By locator) {
        return waitUntil(locator).getText();
    }

    protected void type(By locator, String text) {
        WebElement element = waitUntil(locator);
        element.click();
        element.clear();
        element.sendKeys(text);
    }

}
```

POM - Exemple “LoginPage”

- Elle étend « BasePage »
- On définit les propriétés des éléments relatifs à la page, sous forme de « By »
- Elle peut contenir des méthodes correspondant aux actions de la page
- Getter pour les propriétés

```
public class LoginPage extends BasePage {

    private final By usernameField = By.cssSelector("input[name='username']");
    private final By passwordField = By.cssSelector("input[name='password']");
    private final By submitButton = By.cssSelector("input[id='login-button']");

    public LoginPage(WebDriver driver) {
        super(driver);
    }

    public void login(String username, String password) {
        goTo(ROOT_URL);
        type(usernameField, username);
        type(passwordField, password);
        waitClickable(submitButton).click();
    }

    public By getUsernameField() {
        return usernameField;
    }

    public By getPasswordField() {
        return passwordField;
    }

    public By getSubmitButton() {
        return submitButton;
    }
}
```

POM - Exemple “LoginTest”

- Elle étend « BaseTest »
- Elle contient les méthodes annotées de « @Test »
- On instancie nos objets « Page » en fonction du besoin en passant le « driver » en paramètre (récupérer via « BaseTest »)

```
public class LoginTest extends BaseTest {

   @Test
   public void testLoginWithValidCredentials() {
      LoginPage loginPage = new LoginPage(driver);
      loginPage.login("user", "motdepasse");
    // assert on redirect
   }

}
```

POM – Notion de “chaînage”

- Le but est de fluidifier l’écriture du code
- Ici on a un exemple de « login », non chaîné, c’est assez lourd à écrire

```
@Test
public void testLogin() {
    LoginPage loginPage = new LoginPage(driver);
    loginPage.open();
    loginPage.enterUsername("username");
    loginPage.enterPassword("password");
    ProductsPage productsPage = loginPage.clickLogin();

    String title = productsPage.getTitle();
    assertEquals("Products", title);
}
```

POM – Notion de “chaînage”

- Il s’agit du même code que précédemment, sauf que cette fois, tout est « chaîné », cela permet d’écrire les tests plus simplement
- Pour cela la méthode « **open()** » renvoie « **this** », donc un objet de type « **LoginPage** », depuis lequel on récupère un objet de type « **ProductsPage** », auquel on appelle la méthode « **getTitle** » qui renvoie une chaîne de caractère

```
@Test
public void testLogin() {
   String title = new LoginPage(driver)
         .open()
         .login("username", "password ")
         .getTitle();

   assertEquals("Products", title);
}
```