

Selenium - concevoir et automatiser un processus de tests d'interface utilisateur

Tom Avenel

2022-2023

Résumé

Dans ce cas pratique, nous allons automatiser et industrialiser le processus de test d'interface utilisateur du site Web <https://fr.wikipedia.org/>

Table des matières

Selenium IDE – Enregistrement des scénarii de tests	3
Selenium WebDriver – Automatisation des tests	3
Version Java	4
Version Python	4
Dans les deux cas	4
Selenium Grid – Industrialisation des tests	5
Démarrer la Grid en mode standalone	6
Démarrer la Grid en mode Hub / Node(s)	7
Exécuter les tests dans différents environnements	7
Intégration dans Jenkins®	8
Rendu attendu	8
Legal	9

Selenium IDE – Enregistrement des scénarii de tests

Dans cette première partie, nous allons utiliser le plugin Selenium IDE pour enregistrer les scénarios de test d'interface utilisateur.

Avant de tester notre projet plus complexe, on pourra utiliser le projet [Sample Todo App](#) pour s'entraîner à réaliser des tests simples.

1. Installer le plugin [Selenium IDE](#) correspondant à votre navigateur
2. Ouvrir le module Selenium IDE, créer un nouveau projet et accéder à l'interface Web du produit à tester.
3. Enregistrer les différentes actions de l'utilisateur pour un premier test. Les actions enregistrées sont affichées dans l'interface du test et seront utilisées pour le code d'automatisation dans la partie suivante.
4. Arrêter l'enregistrement : Selenium IDE génère alors un premier scénario de test. Attention : la target CSS récupérée peut être à retravailler : si le locator CSS choisi par Selenium IDE est un élément auto-généré par le framework frontend, celui-ci risque de ne pas fonctionner à la prochaine exécution du test !
5. Pour vérifier le bon enregistrement du test, cliquer sur le bouton **Lecture** du test afin de rejouer les actions enregistrées.
6. De la même manière, créer l'ensemble des tests d'interface utilisateur du produit (ces tests sont donc, en grande partie, ceux que l'on retrouvera dans la recette fonctionnelle du produit). On pourra utiliser les **Test Suites** (cliquer sur l'onglet **Tests** en haut à gauche) pour créer des groupes logiques de tests (par exemple : tests de connexion utilisateur). Ces suites de tests seront utiles pour architecturer les différents tests automatisés dans des packages dédiés dans la partie suivante.

[Plus d'infos sur les sélecteurs CSS.](#)

Selenium WebDriver – Automatisation des tests

Dans la partie précédente, nous avons créé un ensemble de scénarios de tests automatisés à exécuter dans votre navigateur Web.

Si cela permet un test rapide du produit et un rejeu rapide des tests, ceux-ci sont toutefois dépendant :

- De votre environnement personnel (plugin ajouté dans votre navigateur Web)
- De la présence physique du testeur pour rejouer les tests
- Les vérifications de comportement sont toujours réalisées manuellement

Afin de créer une véritable automatisation des tests, et pour commencer à s'abstraire de l'environnement d'exécution, nous allons utiliser le robot **Selenium WebDriver** pour coder l'exécution des tests et les vérifications à effectuer.

Récupérer les sources du projet de test :

```
git clone https://git.sr.ht/~toma/selenium
```

Ce projet contient un template de projet minimal pour lancer des tests **Selenium** en **Java** et en **Python** .

Version Java

- Le projet est packagé par un build `Gradle`
- L'ajout des dépendances Sélénium est déjà réalisé dans le `build.gradle` :
 - Le code de l'API `Selenium WebDriver`
 - La librairie `JUnit` pour exécuter les tests
- Un exemple de test simpliste se trouve dans le répertoire `src/test/java`
- Les tests sont exécutés par le framework de tests unitaires `JUnit` pour `Java`
- Attention à bien placer vos fichiers de test dans le répertoire `src/test` (et non `src/main`) pour que `JUnit` puisse les exécuter.

```
{Windows}$ .\gradlew.bat test
{Linux}$ ./gradlew test
```

Version Python

- L'ajout des dépendances Sélénium est déjà réalisé dans le projet :
 - Le code de l'API `Selenium WebDriver`
 - La librairie `Unittest` pour exécuter les tests
- Un exemple de test simpliste est fourni : `test_example.py`
- Les tests sont exécutés par le framework de tests unitaires `Unittest` pour `Python` :

```
$ python -m unittest test_example.py
```

Dans les deux cas

- On pourra regrouper les suites logiques de tests dans des packages dédiés.

Pour pouvoir exécuter un test d'interface utilisateur dans un navigateur Web, `Selenium WebDriver` délègue les actions à réaliser à un driver spécifique au navigateur. Celui-ci n'est pas inclus dans le template de projet :

- Récupérer le driver de votre navigateur : https://www.selenium.dev/documentation/en/webdriver/driver_requirements/
- En `Java` : il n'est pas obligatoire de mettre à jour le `$PATH` car le chemin vers le driver est codé en dur dans le code du test.
- En `Python` : attention à bien mettre à jour le `$PATH` vers le chemin du driver utilisé dans le code du test.
- Dans un contexte d'environnement continu, on injecterait cette variable depuis l'environnement d'exécution plutôt que de la coder en dur dans le code.

Dans le fichier de test, mettre à jour la ligne de code suivante si nécessaire pour décrire l'emplacement du driver. Attention à modifier le nom de la propriété si le driver n'est pas un driver `Google Chrome` :

```
System.setProperty("webdriver.chrome.driver",
System.getProperty("user.dir") + "/src/main/resources/chromedriver");

self.driver = webdriver.Chrome()
```

L'exemple fourni utilise `Google Chrome`. Si nécessaire, changer le code de l'exemple pour utiliser le driver de votre navigateur.

Exécuter le test fourni en exemple : `$. \gradlew.bat test` (Java) ou `$ python -m unittest test_example.py` (Python)

- Les résultats de test sont disponibles pour chaque test dans le répertoire : `build/reports/test`
 - En Java, le build `Gradle` ajoute le plugin `build-dashboard` qui permet d'aggréger tous les rapports de build : on pourra donc utiliser le fichier `build/reports/buildDashboard/index.html` comme point d'entrée pour tous les rapports de test.

Une fois l'exemple de test correctement exécuté, en s'inspirant de cet exemple et en utilisant les enregistrements de `Selenium IDE`, coder l'automatisation des tests décrits dans la partie précédente.

Attention à bien gérer les problèmes d'état entre les différents tests ! `JUnit` et `Unittest` sont des framework de tests unitaires, qui s'attendent à ce que chaque test soit indépendant et puisse être lancé en parallèle des autres tests.

Note : Les tests sont exécutés à travers le framework `JUnit` ou `Unittest`, qui s'intègre très bien avec la plupart des IDE, il est donc possible de lancer les tests directement depuis l'IDE.

Selenium Grid – Industrialisation des tests

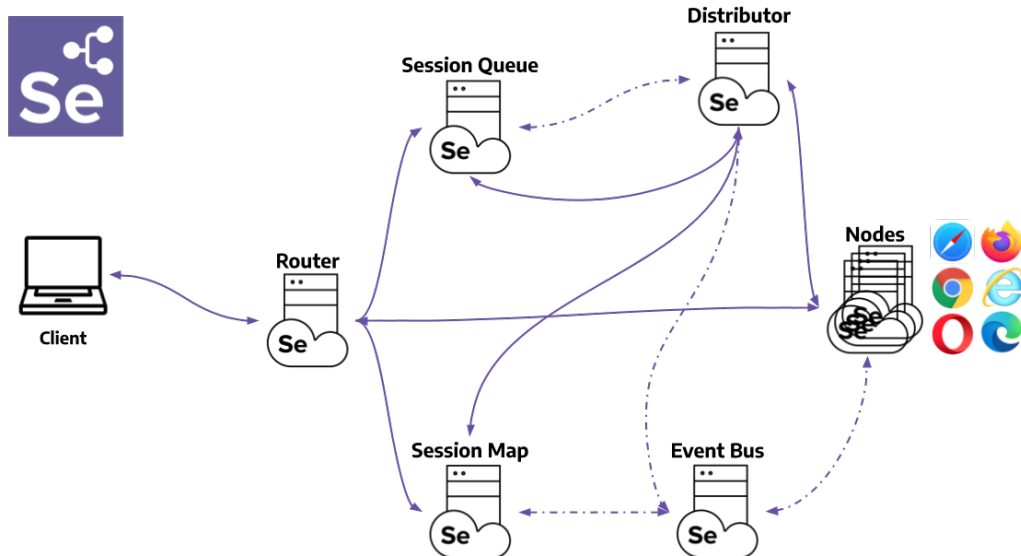
Dans cette partie, nous allons utiliser `Selenium Grid` pour industrialiser notre déploiement de tests en déléguant l'exécution des tests à des agents distants.

`Selenium Grid` permet principalement de :

- Tester notre code sur des environnements différents (OS, Navigateurs Web)
- Paralléliser l'exécution des tests sur plusieurs machines pour réduire le temps global d'exécution de la suite de tests.

L'architecture de `Selenium Grid` est une architecture classique :

- Un `hub` gère l'ordonnancement des tests et l'envoi des commandes à exécuter. Depuis `Grid 4`, le `Hub` est maintenant découpé en sous-composants (`Router`, `Distributor`, `Session Map`, `New Session Queuer`, `Event Bus`) permettant un déploiement beaucoup plus fin - le terme `Hub` décrit alors le regroupement de tous ces composants sur une machine unique. Nous utiliserons uniquement un déploiement de type `Hub` dans ce projet.
- Des noeuds de la grille fonctionnent comme des agents qui attendent les commandes à exécuter et réalisent l'opération sur le ou les navigateurs installés sur la même machine.



Principaux composants Selenium Grid (source et crédits : <https://www.selenium.dev>)

Note :

En cas de problème avec l'exécution des tests dans la **Grid**, on pourra utiliser les fonctionnalités d'observabilité de **Selenium Grid** pour tracer l'exécution des tests.

Selenium Grid 4 introduit également une **API GraphQL** qui est très pratique pour interagir avec les **Nodes**.

Nous allons donc maintenant séparer :

- L'ordonnancement des tests, par le framework **JUnit** ou **Unitest**, lancé comme précédemment.
- Le code des tests sera modifié pour utiliser un **Hub**.
- Les différents **Nodes** serviront à exécuter les actions de test, sur des OS et navigateurs dédiés.

Démarrer la Grid en mode standalone

Pour commencer, nous allons lancer l'ensemble de la **Grid** (**Hub** + **Node**) depuis un **JAR** unique. Cela nous permettra de tester le code de nos tests et de vérifier que la **Grid** est exécutée correctement.

Télécharger le serveur **Selenium** (version **Grid**).

Exécuter le **JAR** de la **Grid** en mode standalone :

```
java -jar selenium-server-4.1.1.jar standalone
```

Attention : l'exécution standalone récupère le driver pour contrôler le navigateur dans le même répertoire que le **JAR**. Copier le driver utilisé (par exemple : **chromedriver**) dans le répertoire

`selenium/grid` avant de lancer la `Grid` .

Dans le code des tests, modifier la configuration des drivers de test pour utiliser un driver distant :

```
String Hub = "http://localhost:4444";
DesiredCapabilities caps = new DesiredCapabilities();
caps.setBrowserName("chrome");
driver = new RemoteWebDriver(new URL(Hub), caps);

self.driver = webdriver.Remote(
    command_executor='http://127.0.0.1:4444/wd/hub',
    options=webdriver.ChromeOptions()
)
```

Exécuter les tests précédents et vérifier leur bon fonctionnement :

- `$.gradlew.bat test` (Java) ou `$ python -m unittest discover` (Python)
- La console de la `Grid` affiche des informations sur l'exécution des tests
- L'API de la `Grid` permet de vérifier la bonne exécution des tests : <http://localhost:4444/status>

Démarrer la Grid en mode Hub / Node(s)

Nous allons maintenant séparer le `Node` (moteur d'exécution du test) du `Hub` (contrôleur du test).

Pour cela :

1. Démarrer le Hub :

```
java -jar selenium-server-4.1.1.jar hub
```

2. Démarrer un `Node` (pour l'instant sur la même machine). Pour se référencer auprès du `Hub` , il faut définir les points d'accès pour publier et recevoir les événements de test :

```
java -jar selenium-server-4.1.1.jar node \
--publish-events "tcp://localhost:4442" \
--subscribe-events "tcp://localhost:4443"
```

3. Exécuter les tests : `$.gradlew.bat test` (Java) ou `$ python -m unittest discover` (Python)
4. Vérifier que l'API du `Hub` a bien reçu l'ordre et exécuté le test, et vérifier dans la console du `Node` que le test a bien été reçu.

Note : Comme décrit précédemment, il est même possible depuis `Grid 4` de séparer le `Hub` en différents composants pour des besoins de déploiement plus complexes.

Exécuter les tests dans différents environnements

En déployant des `Nodes` sur différents systèmes (on pourra utiliser des machines virtuelles `VirtualBox` par exemple), exécuter les tests dans différents environnements :

- `Windows` et `Linux`
- `Google Chrome` , `Firefox` , ...

On utilisera pour cela les `DesiredCapabilities` lors de la mise en place de l'environnement du test, qui permettent d'atteindre un `Node` respectant les contraintes fixées.

Intégration dans Jenkins®

Nous allons maintenant intégrer les tests réalisés au serveur d'intégration continue `Jenkins®` pour automatiser leur exécution en cas de changement dans le code source.

1. Créer un nouveau job dans `Jenkins®` utilisant le code source de l'application testée.
2. Ajouter le code source des tests d'interface utilisateur dans le projet.
3. Intégrer l'exécution des tests `Sélénium` dans `Jenkins®`. Les tests `Sélénium` étant exécutés (dans notre cas) par le framework `JUnit`, on pourra réutiliser le même type d'exécution que dans la partie précédente.

On pourra commencer par exécuter des déploiements `Sélénium` standalone. Lors de l'exécution dans la `Grid Sélénium`, on pourra démarrer la `Grid` manuellement.

Rendu attendu

Il est attendu une archive contenant le code source de l'ensemble des tests et leur configuration dans la `Grid Selenium`, pour tester le projet sur au moins 2 navigateurs différents et dans au moins 2 `Nodes` distincts.

Il n'est pas nécessaire d'exécuter 100% des tests sur chaque environnement, certains tests peuvent être dédiés à un environnement pour y tester une spécificité, et il peut être redondant d'exécuter un même test sur plusieurs environnements. On choisira donc avec soin le plan d'exécution de tous les tests.

On fournira également des captures d'écran d'exécution des tests `Selenium` dans un job `Jenkins®`.

Legal

- © 2022 Tom Avenel under CC BY-SA 4.0
- SELENIUM is a trademark of Software Freedom Conservancy, Inc.
- GRADLE is a trademark of GRADLE, INC
- Oracle and Java are registered trademarks of Oracle and/or its affiliates.
- Jenkins® is a registered trademark of LF Charities Inc.
- “Python” is a registered trademark of the PSF. The Python logos (in several variants) are use trademarks of the PSF as well.

Ce document a été généré grâce à l’outil [pandoc](#).

Les sources au format Markdown de ce document sont disponibles sur le [site web](#).

Ce document est mis à disposition selon les termes de la [CC BY-SA 4.0 : Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 4.0 International](#).

Vous êtes autorisé à :

- Partager — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats
- Adapter — remixer, transformer et créer à partir du matériel
- pour toute utilisation, y compris commerciale.

Selon les conditions suivantes :

- Attribution — Vous devez créditer l’Œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l’Œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l’Offrant vous soutient ou soutient la façon dont vous avez utilisé son Œuvre.
- Partage dans les Mêmes Conditions — Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant l’Œuvre originale, vous devez diffuser l’Œuvre modifiée dans les même conditions, c’est à dire avec la même licence avec laquelle l’Œuvre originale a été diffusée.
- Pas de restrictions complémentaires — Vous n’êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser l’Œuvre dans les conditions décrites par la licence.

Pour plus d’informations : <http://creativecommons.org/licenses/by-sa/4.0/>.