

Selenium, automatisation des tests fonctionnels des applications Web

Sommaire

- Introduction à la plateforme Selenium
 - Rappel sur les tests fonctionnels des applications Web.
 - Le projet Selenium (historique, roadmap).
 - Architecture de la plateforme Selenium.
 - Robots de tests Open Source disponibles.
 - Accès aux éléments d'une page Web.
- Création de tests avec le robot Selenium IDE
 - Enregistrement des cas de test, constitution des suites et rejeu sur Firefox ou Chrome.
 - Utilisation d'outils pour lancer les suites de test.
 - Lancement des tests sur d'autres navigateurs via Selenium server.
- Création de tests automatisés avec Katalon
 - Katalon Studio une suite d'automatisation des tests.
 - Katalon et Selenium.
- Présentation de concepts avancés
 - Démonstration de l'API WebDriver en Java.
 - Structuration en couches et création de mots-clés métiers (illustrations avec Cucumber et Robot Framework).
 - Lancement des tests par une plateforme d'intégration continue.
 - Ponts vers les gestionnaires de tests (SQUASH TM, Testlink).
 - Bonnes pratiques et conclusion.

Introduction à la plateforme Selenium

Rappel sur les tests fonctionnels des applications Web

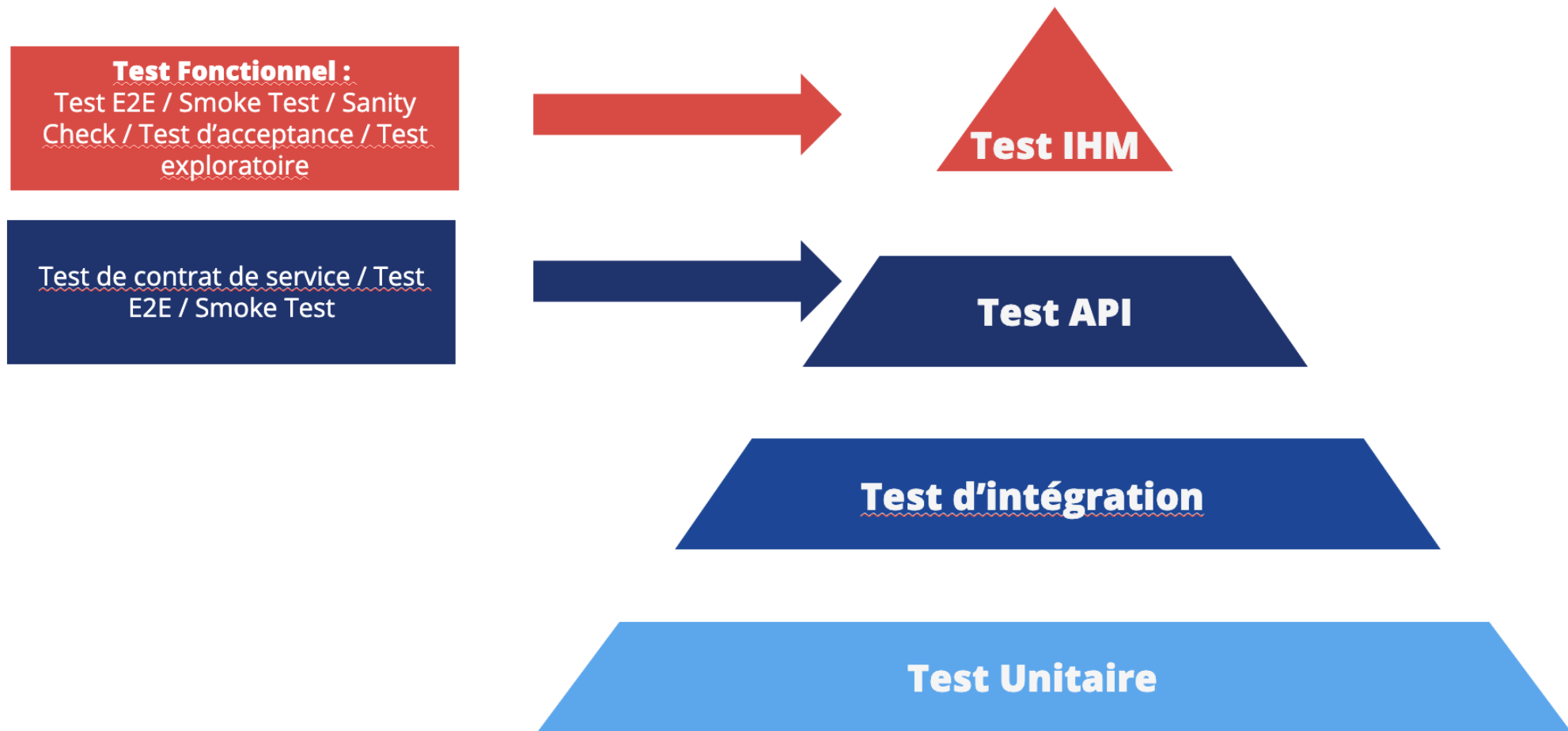
Définition :

Les tests fonctionnels des applications Web consistent à vérifier que les fonctionnalités d'une application répondent aux exigences spécifiées. Ils se concentrent sur **le comportement attendu du système** selon les besoins des utilisateurs et ne concernent pas les aspects internes (comme le code ou la structure).

Utilisation :

- Valider les flux métiers (exemple : processus de paiement dans une boutique en ligne).
- Identifier les régressions suite à des mises à jour de l'application.
- Assurer la compatibilité avec les navigateurs, appareils et systèmes d'exploitation.

Rappel sur les tests fonctionnels des applications Web



Smoke test



Définition :

Les smoke tests sont des tests simples qui vérifient le fonctionnement de base d'une application. Ils sont conçus pour être rapides à exécuter, et leur but est de vous donner l'assurance que les caractéristiques principales de votre système fonctionnent comme prévu.



Exécution :

Il est exécuté après chaque build de version, à chaque "pull request" "Merge". Si les tests ne détectent pas d'anomalie, alors on peut lancer un ensemble de tests plus exhaustifs



Comment ça marche ?

Atteindre des exigences de qualité par des tests rapides, peu coûteux et fiables . Ils sont écrits au début du cycle de vie du logiciel.

Sanity Check



Définition :

Les Sanity check sont des tests qui permettent de vérifier la cohérence de certaines fonctionnalités ou lorsqu'un bug a été détecté



Exécution :

Il est exécuté après chaque build de version, à chaque "pull request" "Merge". Si les tests ne détectent pas d'anomalie, alors on peut lancer un ensemble de tests plus exhaustifs



Comment ça marche ?

Ces tests sont écrits au fur et à mesure du développement du logiciel. Par exemple après la découverte d'un bug critique, un test est écrit pour vérifier que le bug est corrigé et qu'il ne survient pas de nouveau

Test de regression (TR ou TNR)



Définition :

Les TNR permettent d'identifier la régression d'une application entre une version n et une version n-1. Ils peuvent regrouper un ensemble de cas de tests très variés.



Exécution :

L'exécution peut être variable. Ils peuvent être lancés à un moment clé d'un projet : lors d'une mise en production (MEP), ou avant une campagne de tests "manuels". Cela dépend de la stratégie ou de la politique de test au sein de l'entreprise



Comment ça marche ?

Ces tests sont écrits à plus long terme dans le développement du logiciel. Certains tests de Sanity check peuvent devenir des TNR. Ces tests peuvent être créés aussi pour couvrir d'autres périmètres de tests moins exécutés manuellement.

Test End to End (E2E) – Bout-en-bout



Définition :

Test de bout en bout : Test plus complexe et plus exhaustif. Il consiste à tester tout le système, de son interface jusqu'à son mode de fonctionnement.



Exécution :

L'exécution peut être variable. Elle peut être lancée à un moment clé d'un projet. Ou lors d'une mise en production (MEP). Cela dépend de la stratégie ou de la politique de test au sein de l'entreprise



Comment ça marche ?

Ces tests sont écrits à plus long terme dans le développement du logiciel. Ces tests interviennent le plus souvent en dernier dans le processus d'automatisation des tests, car tout ce qui a été produit dans le logiciel va être testé de fond en comble.

Le projet Selenium (historique, roadmap)

Historique :

- **2004** : Création de Selenium par Jason Huggins chez ThoughtWorks pour tester une application Web de manière automatisée.
- **2006** : Naissance de Selenium Remote Control (RC), permettant d'exécuter des scripts à distance.
- **2008** : Fusion avec le projet WebDriver (créé par Google) pour former Selenium 2, avec une API plus moderne.
- **2016** : Sortie de Selenium 3, avec des améliorations en termes de performances et de compatibilité.
- **2021** : Lancement de Selenium 4, introduisant des fonctionnalités comme les **DevTools API** et une meilleure prise en charge des tests dans le cloud.

Roadmap :

Selenium continue d'évoluer avec :

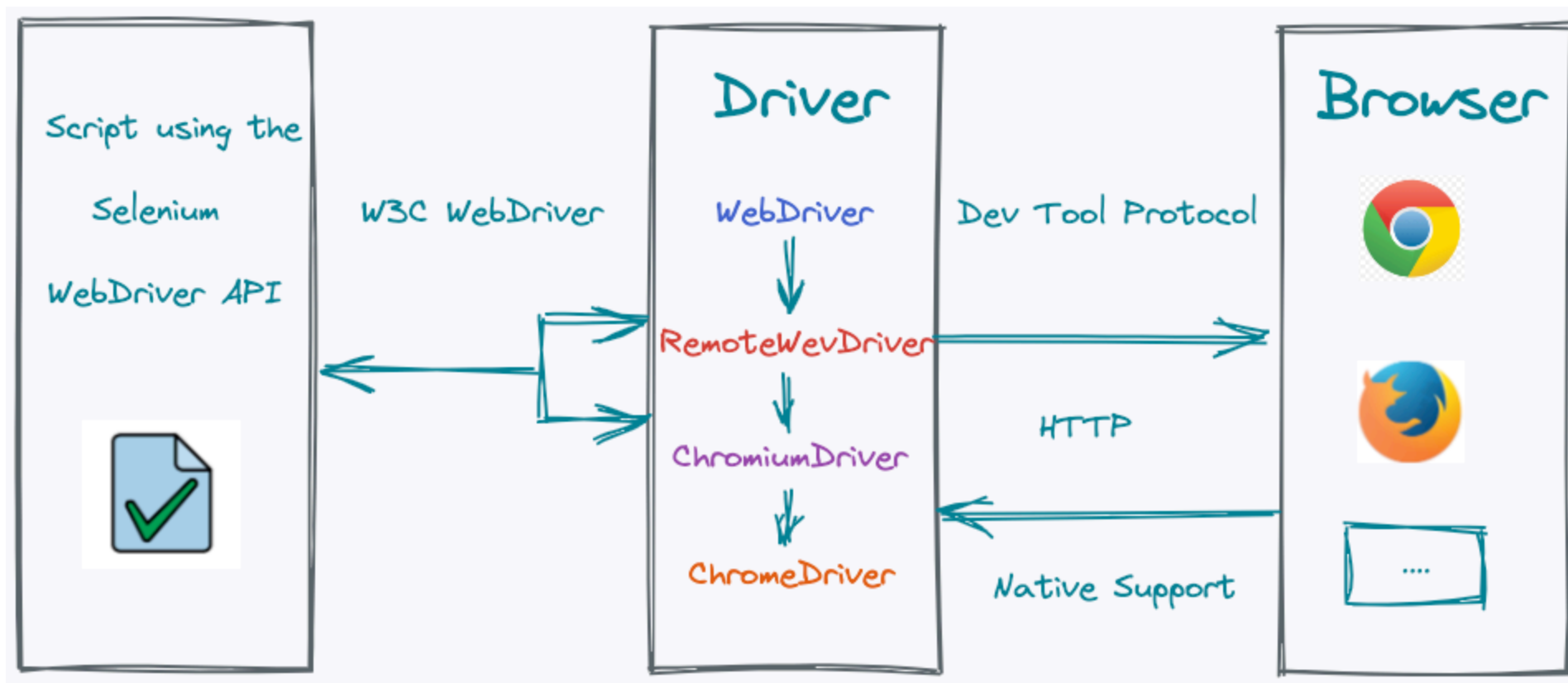
- Un support accru pour les navigateurs modernes.
- Une intégration plus simple avec les pipelines CI/CD.
- L'amélioration des performances des tests sur des plateformes comme Kubernetes ou Docker.

Architecture de la plateforme Selenium

Selenium utilise une architecture client-serveur qui repose sur le **WebDriver Protocol**. Elle se compose des éléments suivants :

- **Client (Test)** : Scripts écrits dans des langages comme Python, Java, ou C#.
- **WebDriver** : Interface entre le script et le navigateur. Chaque navigateur a son propre WebDriver (par exemple, ChromeDriver pour Chrome).
- **Navigateur** : Exécute les commandes envoyées par WebDriver.
- Le client envoie des commandes au WebDriver via HTTP.
- Le WebDriver contrôle le navigateur pour exécuter les actions demandées (clics, saisies, validations).
- Les résultats sont renvoyés au script.

Architecture de la plateforme Selenium



Robots de tests Open Source disponibles

Exemples de robots de tests :

1. **Selenium** : Leader pour les tests fonctionnels Web.
2. **Cypress** : Framework moderne pour les tests E2E, rapide et facile à utiliser.
3. **Playwright** : Créé par Microsoft, supporte des tests cross-browser et inclut des fonctionnalités avancées.
4. **Puppeteer** : Conçu pour automatiser les navigateurs basés sur Chromium.
5. **TestCafe** : Outil léger pour tester les applications Web sans nécessiter de WebDriver.

Outil	Points forts	Limites
Selenium	Multi-navigateur, extensible	Complexité initiale
Cypress	Facilité d'écriture des tests	Support limité pour certains navigateurs
Playwright	API moderne, prise en charge DevTools	Plus récent, moins mature

Accès aux éléments d'une page Web

Pour interagir avec une page Web, les outils comme Selenium identifient les éléments via leurs **sélecteurs**. Les méthodes courantes sont :

- **ID** : `find_element(By.ID, "username")`
- **Nom** : `find_element(By.NAME, "search")`
- **Classe** : `find_element(By.CLASS_NAME, "button")`
- **XPath** : `find_element(By.XPATH, "//button[@id='submit']")`
- **CSS Selector** : `find_element(By.CSS_SELECTOR, ".form input")`

Utilisation :

- Automatiser des actions : remplir des champs, cliquer sur des boutons, etc.
- Scraper des données en identifiant les balises HTML.

Création de tests avec le robot Selenium IDE

Création de tests avec le robot Selenium IDE

Selenium IDE (Integrated Development Environment) est un outil **no-code** qui permet de **créer, enregistrer, et exécuter des tests automatisés** directement dans le navigateur (Firefox ou Chrome). Il est conçu pour les débutants, leur permettant de générer des scripts sans écrire de code.

Utilisation :

- Enregistrer les actions utilisateur sur un site Web.
- Générer des scripts automatisés pour reproduire ces actions.
- Exporter les scripts dans des langages comme Python, Java ou C# pour les intégrer dans des frameworks plus complexes.

Création de tests avec le robot Selenium IDE

 Selenium IDE

[Docs](#) [API](#) [Plugins](#) [Blog](#) [Help](#)


Selenium IDE

Open source record and playback test automation for the web

 CHROME DOWNLOAD

 FIREFOX DOWNLOAD

 LATEST ZIP

 Star 2,270



Web Ready



Easy Debugging



Cross-browser Execution

Enregistrement des cas de test, constitution des suites et rejeu sur Firefox ou Chrome

Un cas de test est un enregistrement d'une séquence d'actions utilisateur sur une page Web. Selenium IDE capture :

- Les clics sur des éléments.
- Les saisies dans les champs.
- Les interactions spécifiques comme le défilement ou les sélections de menu.

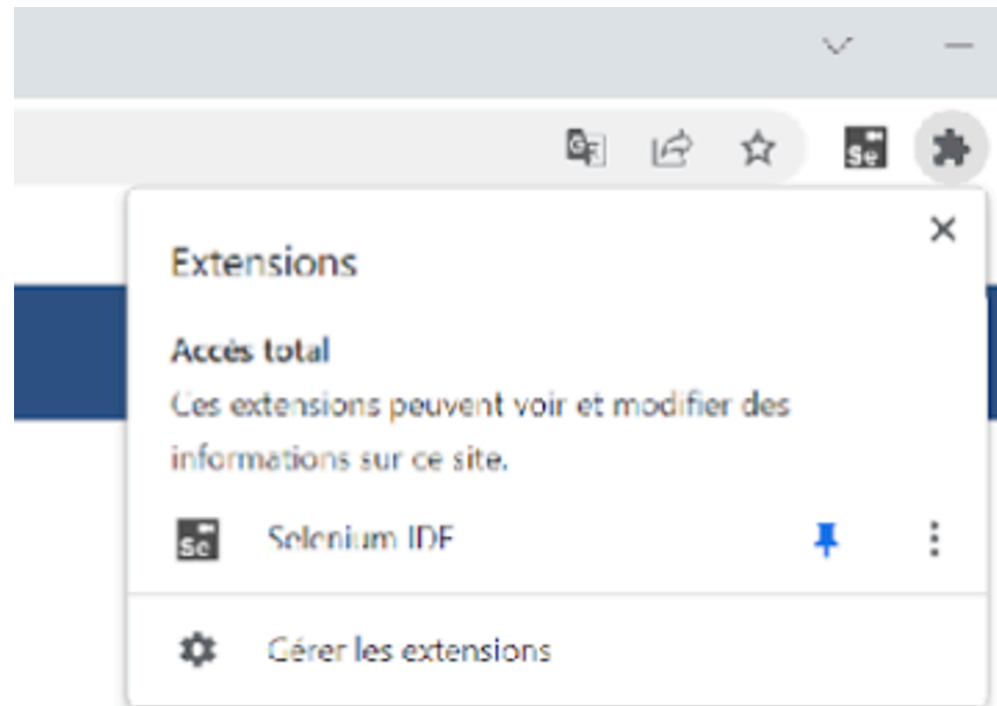
Constitution des suites :

- Une suite de tests est un ensemble de cas de test regroupés pour être exécutés ensemble.
- Par exemple, une suite peut inclure :
 1. Test de connexion.
 2. Test d'ajout au panier.
 3. Test de validation de commande.

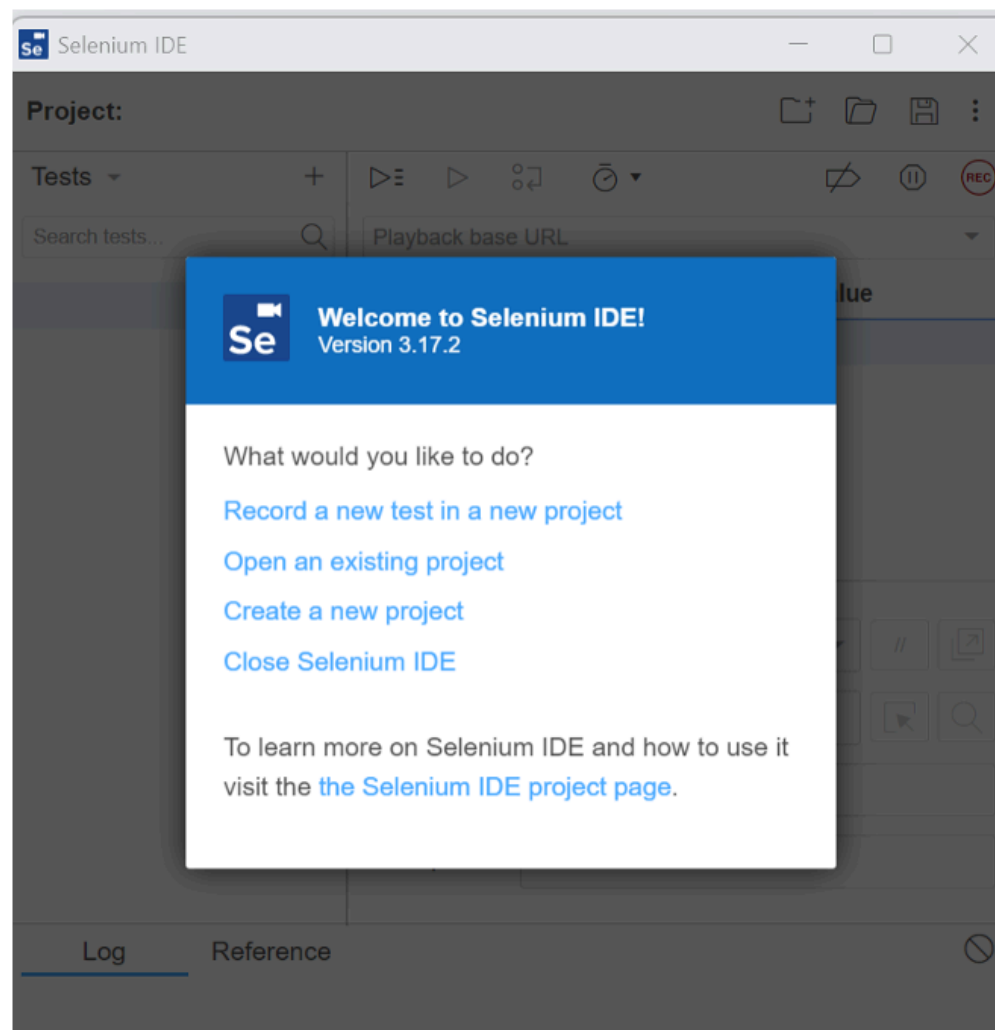
Rejeu sur Firefox ou Chrome :

Selenium IDE peut exécuter ces tests sur le navigateur dans lequel il est installé. Cela permet de vérifier rapidement si les fonctionnalités fonctionnent toujours comme prévu.

Création de tests avec le robot Selenium IDE



Création de tests avec le robot Selenium IDE



Création de tests avec le robot Selenium IDE

×

Name your new project

Please provide a name for your new project.

PROJECT NAME

You can change the name of your project at any time by clicking it and entering a new name.

OK

CANCEL

Sujet de TP

Apprendre à utiliser Selenium IDE pour automatiser des tests d'interaction avec des pages Web. Ce TP permettra aux participants de découvrir les fonctionnalités de base de Selenium IDE, comme l'enregistrement et l'exécution de tests, l'accès aux éléments d'une page Web, et la validation de résultats.

Partie 1 : Installation et découverte de Selenium IDE

Objectif :

Installer Selenium IDE sous forme d'extension dans un navigateur et apprendre à enregistrer et exécuter des tests.

1. Installer Selenium IDE :

- Ouvrir le navigateur Chrome ou Firefox.
- Accéder au store d'extensions (Chrome Web Store pour Chrome ou Add-ons pour Firefox).
- Rechercher **Selenium IDE** et ajouter l'extension au navigateur.

2. Découverte de l'interface :

- Lancer Selenium IDE en cliquant sur l'icône de l'extension.
- Créer un nouveau projet dans Selenium IDE et nommez-le par exemple "TP_Test_Automatisation".
- Explorer les sections principales :

3. Test de bon fonctionnement :

- Enregistrer un test simple qui ouvre `https://example.com`, vérifie que la page se charge correctement, puis ferme la page.

Sujet de TP

Partie 2 : Enregistrement d'un scénario de test simple

Objectif :

Créer un test pour vérifier la fonctionnalité de connexion sur une page de démo.

Scénario :

1. Ouvrir la page `https://www.saucedemo.com/`.
2. Saisir le nom d'utilisateur `standard_user`.
3. Saisir le mot de passe `secret_sauce`.
4. Cliquer sur le bouton de connexion.
5. Vérifier que la page de produits s'affiche bien après la connexion.

Étapes :

1. Dans Selenium IDE, cliquer sur **Record** pour commencer l'enregistrement des actions.
2. Naviguer vers l'URL `https://www.saucedemo.com/`.
3. Remplir les champs **Nom d'utilisateur** et **Mot de passe** avec les valeurs fournies.
4. Cliquer sur le bouton de connexion.
5. Arrêter l'enregistrement et nommer ce test "**Test_Connexion**".

Sujet de TP

Partie 3 : Exercice d'accès aux éléments d'une page Web

Objectif :

Apprendre à accéder aux éléments spécifiques d'une page Web et tester différentes interactions.

Scénario :

Sur la page de produits :

1. Filtrer les produits par prix (du plus bas au plus élevé).
2. Sélectionner un produit en cliquant dessus pour voir son détail.
3. Ajouter le produit sélectionné au panier.
4. Vérifier que le panier contient bien ce produit.

Sujet de TP

Partie 4 : Création d'une suite de tests

Objectif :

Organiser les tests dans une suite de tests et exécuter la suite complète pour valider les fonctionnalités.

Étapes :

1. Dans Selenium IDE, créer une nouvelle suite de tests et nommez-la "**Suite_Tests_Produits**".
2. Ajouter les tests suivants à cette suite :
 - **Test_Connexion**
 - **Test_Filtrer_Produits**
 - **Test_Ajouter_Au_Panier**
3. Exécuter la suite de tests pour vérifier que toutes les étapes s'enchaînent correctement.

Sujet de TP

Partie 5 : Validation des résultats et analyse

Objectif :

Analyser les résultats des tests pour identifier les éventuelles erreurs et en comprendre les causes.

Étapes :

1. Observer les logs générés par Selenium IDE pour chaque test.
2. Identifier les étapes qui ont échoué (s'il y en a) et comprendre les raisons :
 - Mauvaise localisation d'un élément.
 - Délai de chargement de la page (penser à utiliser des commandes d'attente si nécessaire).
3. Ajouter des **Commandes de vérification** (comme `assertText`, `verifyElementPresent`) pour rendre les tests plus robustes.

Utilisation d'outils pour lancer les suites de test

Pour exécuter des suites de test de manière automatisée et à grande échelle, des outils complémentaires sont utilisés. Ces outils permettent :

- **Planification des tests** : Exécuter les tests régulièrement (par exemple, après chaque déploiement).
- **Rapport des résultats** : Générer des rapports détaillés sur les succès et échecs.
- **Exécution sur plusieurs navigateurs ou appareils simultanément.**

Outils couramment utilisés :

1. **JUnit/TestNG** : Utilisé avec Selenium WebDriver pour exécuter les tests de manière structurée.
2. **Jenkins** : Permet de déclencher les suites de test dans une pipeline CI/CD.
3. **BrowserStack/Sauce Labs** : Exécute les tests sur différents navigateurs et plateformes (mobiles, bureau).

Lancement des tests sur d'autres navigateurs via Selenium Server

Selenium Server (ou Selenium Grid) permet d'exécuter des tests sur plusieurs navigateurs et environnements simultanément. Il est utile pour tester la compatibilité entre navigateurs et plateformes.

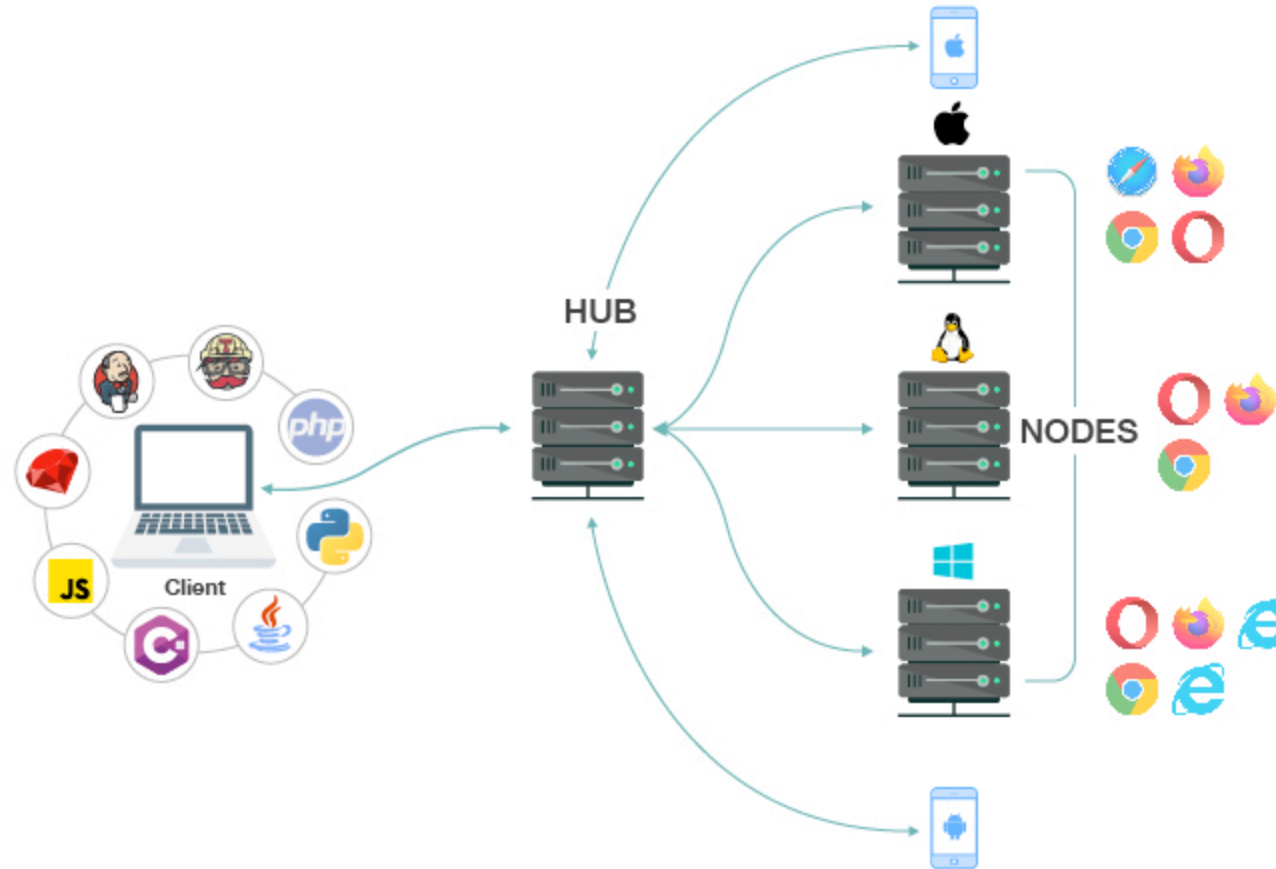
Architecture :

- **Hub** : Machine centrale qui gère la communication avec les nœuds.
- **Nœuds** : Machines distantes (ou locales) exécutant les tests sur des navigateurs spécifiques.

Utilisation :

1. Installer Selenium Server.
2. Démarrer un **hub** sur une machine centrale.
3. Connecter des **nœuds** exécutant différents navigateurs (Chrome, Firefox, Edge).
4. Exécuter les tests via le hub, qui les distribue aux nœuds appropriés.

Lancement des tests sur d'autres navigateurs via Selenium Server



Création de tests automatisés avec Katalon

Création de tests automatisés avec Katalon

Katalon est une plateforme d'automatisation de tests complète et facile à utiliser. Elle permet de créer, exécuter et gérer des tests pour différents types d'applications (Web, API, Mobile, et Desktop) sans nécessiter une expertise approfondie en programmation. Son interface conviviale aide les débutants à créer des tests rapidement tout en permettant aux experts d'utiliser des fonctionnalités avancées.

Étapes de création de tests avec Katalon

1. Installation de Katalon Studio :

- Télécharger et installer depuis katalon.com.
- Lancer l'application et créer un projet (exemple : "Test Application Web").

2. Création d'un cas de test :

- Aller dans l'onglet **Test Cases** et cliquer sur **New Test Case**.
- Ajouter des étapes de test à l'aide de l'éditeur **Keyword View** (interface glisser-déposer) ou **Script View** (écriture de code en Groovy).

3. Enregistrement d'un scénario :

- Lancer le **Web Recorder**.
- Naviguer sur votre site en cliquant sur les éléments souhaités (formulaires, boutons, liens).
- Katalon génère automatiquement les étapes correspondantes.

Étapes de création de tests avec Katalon

4. Personnalisation des étapes :

- Ajouter des assertions pour vérifier des éléments (exemple : "Le message de confirmation s'affiche").
- Gérer des données dynamiques avec des variables.

5. Exécution des tests :

- Sélectionner le navigateur souhaité (Chrome, Firefox, Edge, etc.).
- Lancer le test et analyser les résultats directement dans l'interface de Katalon.

Katalon Studio : Une suite d'automatisation des tests

Katalon Studio est une **suite d'automatisation des tests complète**. Elle propose des outils pour la conception, l'exécution et la gestion des tests, intégrés dans un seul environnement. Elle supporte les tests fonctionnels, les tests de régression, et les tests E2E.

Fonctionnalités clés :

1. Tests multi-plateformes :

- Automatisation des tests Web, API, mobile et desktop.
- Support pour des navigateurs modernes (Chrome, Firefox, Edge) et des environnements comme Android/iOS.

2. Enregistreur intégré :

- Capture les interactions utilisateur pour créer automatiquement des scripts.

3. Rapports détaillés :

- Génération de rapports après chaque exécution.
- Capture des erreurs avec des captures d'écran.

Katalon Studio : Une suite d'automatisation des tests

4. Exécution distribuée :

- Tests parallèles sur plusieurs environnements.

5. Intégration DevOps :

- Connectivité avec Jenkins, Git, et des outils CI/CD.

Avantages de Katalon Studio :

- **Accessibilité** : Interface intuitive pour les débutants, fonctionnalités avancées pour les experts.
- **Bibliothèque de mots-clés** : Automatisation simplifiée grâce à une bibliothèque riche en mots-clés prédéfinis (login, vérification de page, clics, etc.).
- **Flexibilité** : Personnalisation des tests avec le langage Groovy.

Katalon et Selenium

Relation entre Katalon et Selenium :

1. Selenium comme moteur sous-jacent :

- Katalon utilise Selenium WebDriver pour exécuter les tests Web. Cela signifie qu'il s'appuie sur les fonctionnalités robustes de Selenium pour interagir avec les navigateurs.

2. Différence entre les deux :

- **Selenium** est une bibliothèque pour les développeurs. Il nécessite des compétences en programmation pour créer des scripts.
- **Katalon Studio** est une solution complète avec une interface utilisateur qui simplifie l'automatisation des tests. Il ajoute une couche visuelle et des fonctionnalités prêtes à l'emploi.

3. Interopérabilité :

- Katalon permet d'importer ou d'utiliser des scripts Selenium existants, ce qui facilite la migration des projets.

Comparaison entre Katalon et Selenium

Caractéristique	Katalon Studio	Selenium
Public cible	Débutants et experts	Développeurs et testeurs experts
Interface utilisateur	Interface graphique	Aucun (scripts uniquement)
Rapports intégrés	Oui (rapports détaillés)	Non (nécessite des outils externes)
Type d'applications	Web, API, Mobile, Desktop	Web uniquement
Code requis	Faible ou modéré	Obligatoire

Exemple détaillé : Création et exécution de tests dans Katalon

1. Créer un projet Web :

- Nom du projet : `Test_Login_Web`.

2. Enregistrer un scénario :

- Ouvrir l'enregistreur et naviguer sur un site de test (par exemple, `https://example.com`).
- Cliquer sur "Login", entrer un nom d'utilisateur et un mot de passe, et soumettre.

3. Ajouter des assertions :

- Vérifier qu'un élément (par exemple, "Bienvenue, utilisateur") est présent après la connexion.
- Dans le mode **Keyword View**, ajouter une étape `Verify Element Text`.

4. Exécuter sur plusieurs navigateurs :

- Sélectionner Chrome, puis Firefox, pour lancer les tests.

5. Analyser les résultats :

- Après l'exécution, consulter les logs pour vérifier les étapes réussies et les erreurs.
- Les erreurs sont accompagnées de captures d'écran pour faciliter le débogage.

Avantages combinés de Katalon et Selenium

1. **Accessibilité** : Katalon simplifie les tests grâce à son interface, tout en tirant parti de la puissance de Selenium.
2. **Migration facile** : Les équipes utilisant déjà Selenium peuvent adopter Katalon pour bénéficier d'une interface graphique et de fonctionnalités avancées sans perdre leur travail existant.
3. **Écosystème riche** : Katalon ajoute des fonctionnalités telles que des intégrations CI/CD, des enregistreurs, et des rapports, qui ne sont pas disponibles dans Selenium seul.

Sujet de TP

Objectif général :

Ce TP a pour objectif d'explorer et de mettre en œuvre les fonctionnalités de base de Katalon Studio en utilisant Selenium comme moteur sous-jacent. Vous apprendrez à :

1. Installer et configurer Katalon Studio.
2. Créer et exécuter des tests automatisés.
3. Comprendre comment Katalon utilise Selenium pour interagir avec les éléments d'une application Web.

Sujet de TP

Partie 1 : Installation et configuration de Katalon Studio

Objectif : Installer Katalon Studio, créer un projet, et découvrir son interface.

Étapes :

1. Téléchargement de Katalon Studio :

- Accéder au site officiel : <https://katalon.com/>.
- Télécharger Katalon Studio et l'installer sur votre machine.

2. Lancement de Katalon Studio :

- Créez un compte si nécessaire et connectez-vous à l'application.
- Créez un nouveau projet appelé "**TP_Test_Automatisé**".

3. Découverte de l'interface :

- **Test Cases** : Permet de gérer les cas de test.
- **Object Repository** : Stocke les objets (éléments de la page Web) utilisés dans les tests.
- **Test Suite** : Permet de regrouper plusieurs tests pour une exécution collective.
- **Execution Log** : Affiche les résultats et logs d'exécution.

Sujet de TP

Partie 2 : Création d'un test automatisé simple

Objectif : Créer un test automatisé pour vérifier le fonctionnement d'un formulaire de connexion sur un site Web.

Scénario de test :

1. Accéder au site `https://www.saucedemo.com/`.
2. Saisir un nom d'utilisateur valide `standard_user` et un mot de passe valide `secret_sauce`.
3. Cliquer sur le bouton **Login**.
4. Vérifier que la page des produits s'affiche.

Étapes :

1. Enregistrement du test :

- Cliquez sur **Record Web** dans Katalon.
- Entrez l'URL de départ : `https://www.saucedemo.com/`.
- Effectuez les actions nécessaires (saisie des identifiants et clic sur le bouton **Login**).
- Arrêtez l'enregistrement et enregistrez le cas de test sous le nom "**Test_Connexion**".

Sujet de TP

2. Création des vérifications (Assertions) :

- Ouvrez le cas de test enregistré.
- Ajoutez une commande `Verify Element Text` pour valider que la page des produits contient le texte **"Products"**.

3. Exécution du test :

- Exécutez le test dans un navigateur (par exemple, Chrome).
- Vérifiez que les actions s'exécutent correctement et que les validations passent.

Sujet de TP

Partie 3 : Utilisation du moteur Selenium dans Katalon

****Objectif :**** Explorer comment Katalon utilise Selenium comme moteur sous-jacent et comprendre les commandes Selenium générées.

Étapes :

1. Examiner les étapes du test :

- Ouvrez le **Script View** pour le test enregistré.
- Identifiez les commandes Selenium correspondantes, comme `driver.findElement()` ou `driver.get()`.

2. Modifier un test pour utiliser une autre méthode de localisation :

- Accédez à un élément de la page (par exemple, le champ de connexion) en utilisant son XPath au lieu de son ID.
- Dans le **Object Repository**, modifiez le sélecteur de l'élément et exécutez à nouveau le test pour vérifier.

Sujet de TP

Partie 4 : Création d'un test avec des interactions avancées

Objectif : Automatiser des actions complexes comme le tri des produits, l'ajout au panier, et la gestion du panier.

Scénario de test :

1. Filtrer les produits par prix décroissant.
2. Ajouter le premier produit affiché au panier.
3. Accéder au panier et vérifier que le produit ajouté est bien présent.

Étapes :

1. Créer un nouveau cas de test :

- Cliquez sur **New Test Case** et nommez-le "**Test_Panier**".
- Enregistrez les actions suivantes :
 - Sélectionner l'option "Price (high to low)" dans le menu déroulant.
 - Ajouter le premier produit au panier.
 - Ouvrir le panier et vérifier le contenu.

Sujet de TP

2. Ajouter des vérifications :

- Ajoutez une commande `Verify Element Present` pour valider la présence du produit dans le panier.
- Ajoutez une commande `Verify Element Text` pour vérifier le nom ou le prix du produit.

3. Exécuter et valider le test :

- Lancez le test dans le navigateur.
- Vérifiez que toutes les actions s'exécutent correctement.

Sujet de TP

Partie 5 : Création d'une suite de tests

Objectif : Regrouper plusieurs tests dans une suite pour les exécuter ensemble.

Étapes :

1. Créer une Test Suite :

- Cliquez sur **New Test Suite** et nommez-la "**Suite_Fonctionnelle**".
- Ajoutez les tests suivants :
 - **Test_Connexion**
 - **Test_Panier**

2. Exécuter la Test Suite :

- Lancez l'exécution de la suite et observez les résultats pour chaque test.

Sujet de TP

Partie 6 : Génération de rapports et analyse

Objectif : Analyser les résultats d'exécution des tests et générer un rapport.

Étapes :

1. Visualiser les résultats dans Katalon :

- Consultez l'onglet **Execution Log** pour chaque test.
- Identifiez les étapes réussies et celles ayant échoué (le cas échéant).

2. Exporter les résultats :

- Générez un rapport d'exécution en PDF ou HTML pour partager les résultats.

3. Analyse des résultats :

- Si des tests échouent, identifiez les causes possibles (erreur de localisation d'un élément, problème de délai, etc.).
- Ajustez les scripts ou ajoutez des commandes `Wait` si nécessaire.

Présentation de concepts avancés

Démonstration de l'API WebDriver en Java

Qu'est-ce que l'API WebDriver ?

L'API WebDriver est un composant principal de Selenium qui permet de piloter des navigateurs Web de manière programmatique. Il repose sur un protocole standardisé (W3C WebDriver) pour envoyer des commandes au navigateur et récupérer des informations sur son état ou les éléments qu'il affiche.

Objectif principal :

- Automatiser les interactions utilisateur avec un navigateur (clics, saisies, défilements, etc.).
- Tester des applications Web sur différents navigateurs (Chrome, Firefox, Edge, Safari, etc.).

Les concepts essentiels de WebDriver :

1. Initialisation d'un navigateur :

Chaque test commence par créer une instance de WebDriver, spécifique au navigateur souhaité. Cela nécessite des pilotes comme **ChromeDriver** ou **GeckoDriver** pour établir une communication entre Selenium et le navigateur.

2. Navigation dans une application Web :

WebDriver permet de charger des pages Web via des URL, de naviguer entre les pages, et de gérer des actions comme revenir à la page précédente ou actualiser la page.

3. Localisation des éléments HTML :

WebDriver fournit plusieurs méthodes pour cibler les éléments sur une page Web :

- **ID** : Utilisé pour les éléments ayant des identifiants uniques.
- **Nom** : Approche classique pour des champs de formulaire.
- **XPath** : Permet de cibler des éléments même dans des structures complexes.
- **CSS Selectors** : Utile pour des localisations précises basées sur des styles.

Les concepts essentiels de WebDriver :

4. Interactions utilisateur :

WebDriver prend en charge les actions comme :

- Clic sur un bouton.
- Saisie de texte dans un champ.
- Défilement dans une page.
- Double clics ou glisser-déposer.

5. Vérifications (Assertions) :

Les tests nécessitent des vérifications pour s'assurer que les résultats obtenus correspondent aux attentes. Par exemple, vérifier qu'un message de confirmation s'affiche après l'envoi d'un formulaire.

Exercice :

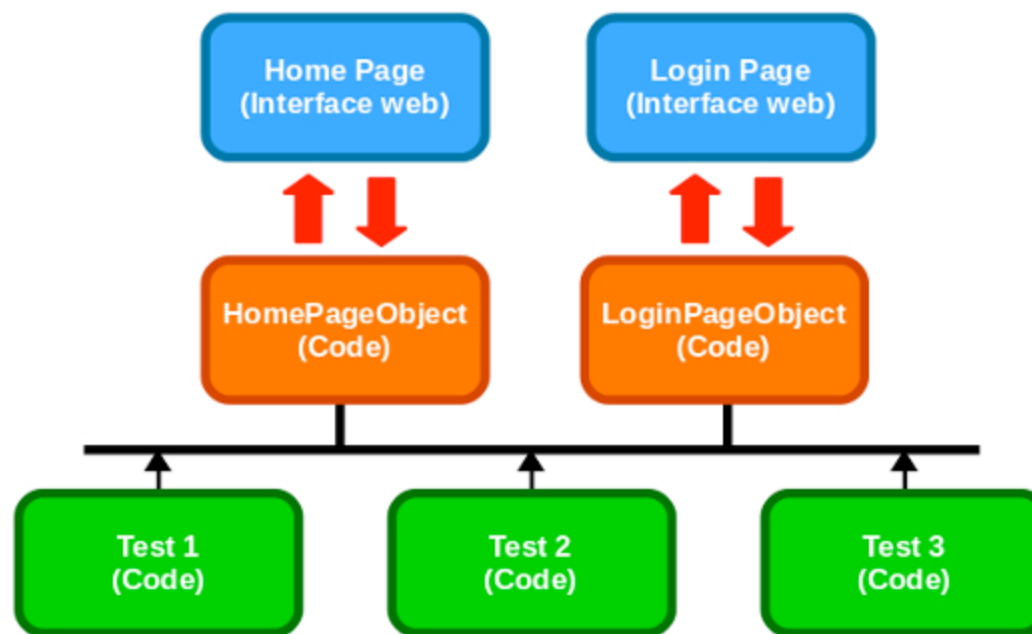
Scénario : Tester un moteur de recherche

1. Ouvrir un navigateur.
2. Naviguer vers un moteur de recherche (par exemple, Google).
3. Saisir "Selenium WebDriver" dans le champ de recherche.
4. Cliquer sur le bouton de recherche.
5. Vérifier que les résultats affichés contiennent le terme recherché.

Structuration en couches et création de mots-clés métiers

Définition de la structuration en couches :

Structurer les tests en couches consiste à séparer les responsabilités dans un script de test. Chaque couche a un rôle précis et est indépendante des autres. Cette approche améliore la lisibilité, la réutilisabilité et la maintenance des scripts.



Les principales couches dans les tests automatisés :

1. Couche Page Object (Objets des pages) :

- Regroupe les définitions des éléments de l'interface utilisateur (boutons, champs, etc.) pour une page donnée.
- Exemple : Une classe "LoginPage" contient des méthodes pour saisir le nom d'utilisateur et le mot de passe, et cliquer sur le bouton de connexion.

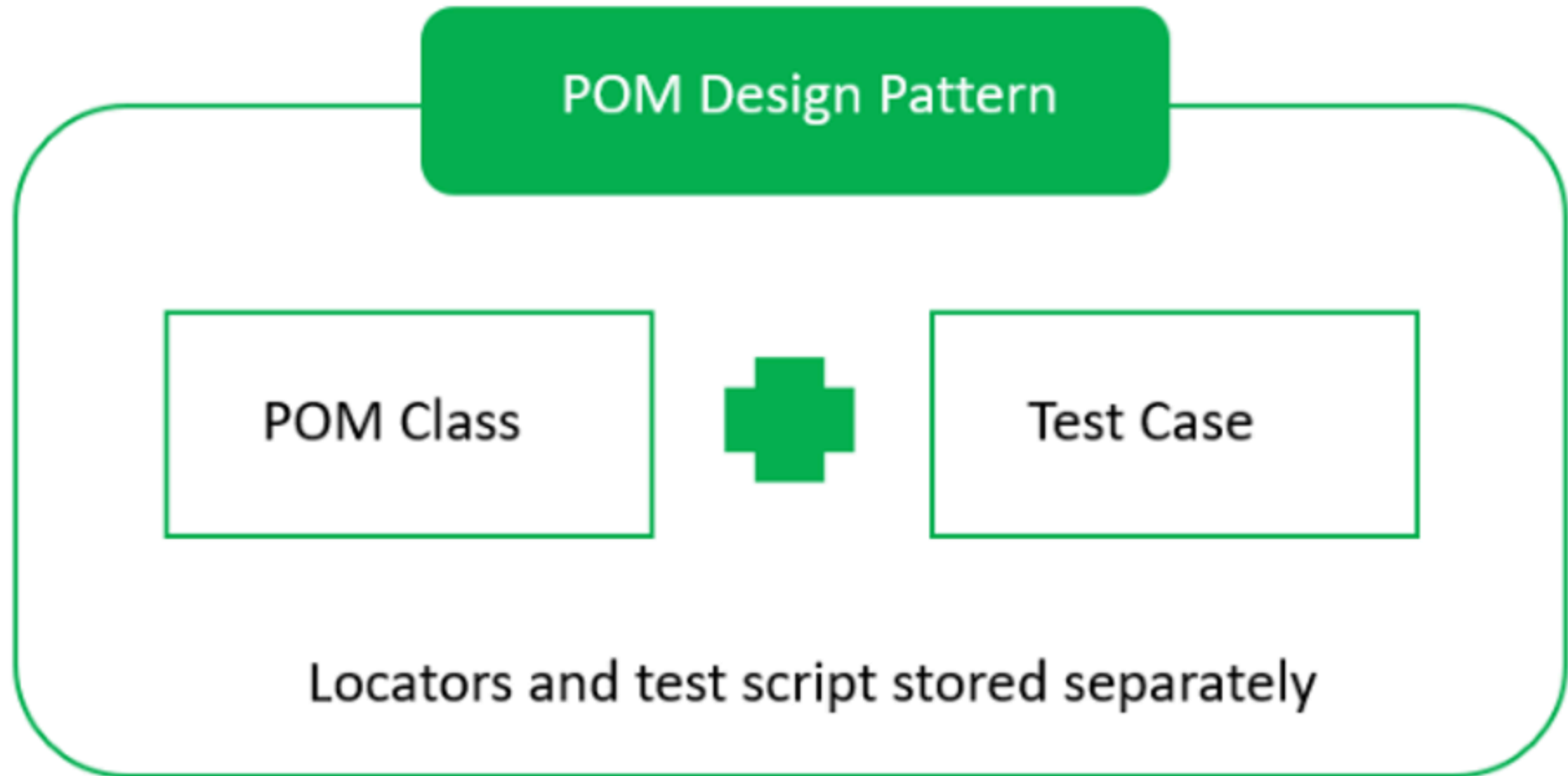
2. Couche des mots-clés métier :

- Définit des actions métiers indépendantes du design technique, comme "se connecter" ou "ajouter un produit au panier".
- Ces mots-clés encapsulent plusieurs actions techniques en une seule action métier.

3. Couche des scénarios :

- Combine plusieurs mots-clés métier pour créer des cas de test complets, comme "tester un processus de paiement".

Les principales couches dans les tests automatisés :



Exemple concret

Scénario métier : Tester l'ajout d'un produit au panier

1. Page Object : Définir les éléments de la page produit (bouton "Ajouter au panier", champ quantité).
2. Mot-clé métier : "Ajouter un produit au panier" combine l'action de sélectionner un produit, définir une quantité, et cliquer sur le bouton.
3. Scénario : Inclure le mot-clé métier "Ajouter un produit au panier" dans un flux complet comme "Acheter un produit".

Illustration avec des frameworks:

1. Cucumber (BDD) :

- Permet de décrire des scénarios dans un langage naturel structuré.
- Exemple :

```
Feature: Login functionality
  Scenario: Successful login
    Given I am on the login page
    When I enter valid credentials
    Then I should see my dashboard
```

Illustration avec des frameworks:

2. Robot Framework :

- Utilise des mots-clés réutilisables pour décrire les actions.
- Exemple :

```
Login Test
  Open Browser      https://example.com    Chrome
  Input Text        username_field        user
  Input Text        password_field        password
  Click Button      login_button
  Page Should Contain Welcome
```

Lancement des tests par une plateforme d'intégration continue

Une plateforme CI, comme **Jenkins**, **GitLab CI**, ou **CircleCI**, exécute automatiquement des tests après chaque modification du code source. Cela permet de vérifier rapidement que le nouveau code n'a pas introduit de régressions.



Étapes dans une pipeline CI/CD

1. Configuration initiale :

- Connecter la plateforme CI à un dépôt de code source (GitHub, GitLab).
- Définir un fichier de configuration (par exemple, `.gitlab-ci.yml` ou `Jenkinsfile`).

2. Exécution des tests :

- Planifier l'exécution des tests à chaque push, pull request, ou de manière périodique.
- Charger les dépendances nécessaires (pilotes WebDriver, navigateurs).

3. Analyse des résultats :

- Si un test échoue, la plateforme alerte l'équipe via des notifications ou des e-mails.
- Les rapports d'exécution sont générés automatiquement.

Avantages :

- Identifie rapidement les bugs introduits par les nouvelles modifications.
- Automatisation complète des tests, réduisant les efforts manuels.
- Intégration avec des outils de reporting pour une visibilité claire sur la qualité du code.

Ponts vers les gestionnaires de tests (SQUASH TM, Testlink)

Qu'est-ce qu'un gestionnaire de tests ?

Un gestionnaire de tests est un outil conçu pour organiser et suivre les tests, qu'ils soient manuels ou automatisés. Les deux outils mentionnés, **SQUASH TM** et **Testlink**, permettent de centraliser toutes les activités de test dans un environnement unique.

Fonctionnalités principales :

1. Planification des tests :

- Création de campagnes de tests (groupes de cas de test pour des fonctionnalités spécifiques).
- Suivi de l'exécution des tests.

2. Suivi des anomalies :

- Documenter les erreurs rencontrées lors des tests.
- Prioriser les corrections à apporter.

3. Intégration avec des outils automatisés :

- Associer des scripts Selenium aux cas de test documentés.
- Importer les résultats des tests automatisés pour enrichir les rapports.

Bonnes pratiques et conclusion

Bonnes pratiques pour les tests automatisés :

1. Structuration en couches :

- Séparer les interactions avec l'interface utilisateur, la logique métier, et les scénarios.

2. Tests prioritaires :

- Automatiser en priorité les tests critiques (par exemple, processus de paiement).

3. Exécution régulière :

- Planifier des exécutions fréquentes via une plateforme CI.

4. Documentation claire :

- Associer des tests automatisés à des cas de test documentés pour une traçabilité complète.

Sujet TP

Contexte : L'objectif est d'automatiser le test d'une application Web (<https://www.saucedemo.com/>) en utilisant **Cucumber** et **Selenium WebDriver**. Vous devrez créer une structure en couches pour séparer les étapes métier, les interactions avec la page, et les scénarios de tests.

Objectifs pédagogiques :

1. Comprendre la structuration en couches pour des tests automatisés (Page Object Model).
2. Rédiger des scénarios de tests en Gherkin (Cucumber).
3. Implémenter des étapes de tests en Java avec Selenium WebDriver.
4. Exécuter les tests dans un environnement configuré.

Sujet TP

1. Préparation de l'environnement

1. Installer les outils nécessaires :
 - Java JDK.
 - Maven.
 - IntelliJ IDEA ou Eclipse.
 - ChromeDriver ou GeckoDriver (selon le navigateur utilisé).
2. Créer un projet Maven et ajouter les dépendances suivantes :
 - **Cucumber** pour les scénarios en Gherkin.
 - **Selenium WebDriver** pour l'automatisation des actions.
3. Configurer une structure de projet respectant les standards Maven.

Sujet TP

2. Scénarios de tests à automatiser

Scénario 1 : Connexion utilisateur

- **Objectif :** Vérifier qu'un utilisateur peut se connecter avec des identifiants valides.
- **Étapes :**
 1. Ouvrir la page de connexion.
 2. Saisir un nom d'utilisateur et un mot de passe valides.
 3. Cliquer sur le bouton de connexion.
 4. Vérifier que la page de produits est affichée.

Scénario 2 : Tri des produits

- **Objectif :** Vérifier que le tri des produits fonctionne correctement.
- **Étapes :**
 1. Appliquer le tri "Price (high to low)" sur la liste des produits.
 2. Vérifier que les produits sont affichés dans l'ordre décroissant des prix.

Sujet TP

Scénario 3 : Gestion du panier

- **Objectif :** Vérifier que l'ajout d'un produit au panier fonctionne.
- **Étapes :**
 1. Ajouter un produit spécifique au panier.
 2. Accéder au panier et vérifier que le produit ajouté y est bien présent.

Sujet TP

3. Structure demandée

1. Structure du projet :

- `src/main/java/pages/` : Contiendra les objets des pages (Page Object Model).
- `src/test/java/steps/` : Contiendra les fichiers implémentant les étapes des scénarios.
- `src/test/resources/features/` : Contiendra les fichiers `.feature` pour les scénarios.

2. Page Object Model :

- Créer des classes pour chaque page Web, comme `LoginPage` ou `ProductsPage`, encapsulant les interactions avec les éléments HTML (clics, saisie, etc.).

3. Etapes des tests :

- Les étapes doivent être implémentées dans des fichiers Java distincts, en suivant les fichiers `.feature`.

4. Exécution :

- Configurer un fichier Runner avec Cucumber pour exécuter les scénarios.