CentraleSupélec | université PARIS-SACLAY

Automatisation Web

Avec Playwright

https://github.com/himhah-op/playwright-training

→ 18 mars 2025

onepoint.
beyond the obvious

Agenda

Module	Date	Horaires	Salle
Module 1 Introduction à la méthodologie de test	04-mars	8h15-11h30	EF.202, Eiffel
Module 2 TDD / BDD	10-mars	8h15-11h30	SD.103, Bouygues
Module 3 Test Web Services : Postman	17-mars	8h15-11h30	Amphi e.068, Bouygues
Module 4 Automatisation Web Playwright	18-mars	13h45-17h	EF.110, Eiffel
Module 5 Automatisation App Mobile	21-mars	8h15-11h30	EF.202, Eiffel
Module 6 Test de performance : Gatling	24-mars	8h15-11h30	Amphi sc.071, Bouygues
Module 7 Accessibilité RGAA, Green, Data/IA	07-avr	8h15-11h30	TBD
Module 8 Evaluation	08-avr	13h45-17h	TBD

Logistique

Assiduité et émargement

Pauses

Interactivité et questions

Utilisation de laptops et smartphones

Evacuation

Sommaire

0. Introduction aux tests Web

Pourquoi automatiser les tests ?

- Gagner du temps (sur les campagnes de TNR)
- Permettre aux développeurs de se focaliser sur le développement
- Réduire le nombre d'anomalies / Eviter les régressions / Améliorer la qualité du produit

Une application informatique est développée et peut ensuite être maintenue pendant plusieurs années

Tout au long de son existence, elle va subir des modifications (corrections, nouvelles fonctionnalités, migrations techniques…)

Chaque livraison de version va nécessiter des tests pour vérifier les nouveautés, mais aussi garantir le fonctionnement des fonctions existantes (non régression)

Dans un cycle agile, les nouvelles versions sont livrées aux testeurs sur un rythme très fréquent, à chaque itération (1-2 semaines), voire plusieurs fois au sein d'une même itération

Pourquoi automatiser les tests ?

La fréquence des livraisons de versions s'est considérablement accélérée

Les applications développées évoluent plus rapidement en fonction des changements plus fréquents exigés par le business

Les tests doivent donc être effectués plus souvent et plus vite

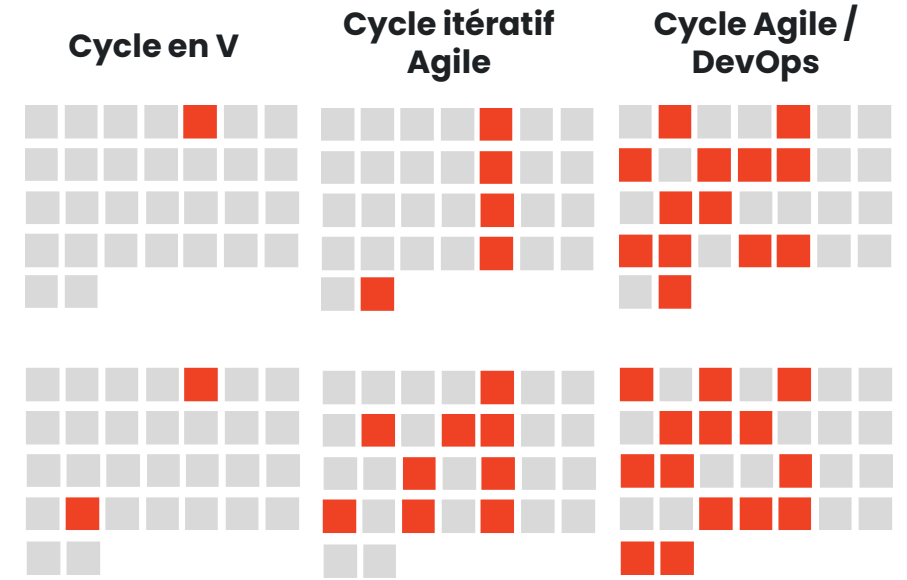
Ils sont effectués à chaque itération sur des versions en cours de construction

Ils ne doivent cependant pas compromettre la qualité

L'exécution fréquente des tests de non-régression devient inévitable et nécessite leur automatisation. Le reporting de test doit restituer la qualité d'ensemble

La qualité devient l'affaire de tous

L'ensemble de l'équipe doit participer à l'effort de test



Avantages et inconvénients

Tester plus vite

- Réduire les charges de test manuel
- Réduire le travail répétitif
- Augmenter la productivité des tests

Tester plus

- Elargir le périmètre des tests
- Concentrer les tests manuels sur les tâches à valeur ajoutée (nouvelles fonctionnalités, tests exploratoires…)

Tester mieux

- Diminuer les erreurs de manipulation
- Fournir une évaluation objective
- Capitaliser d'une campagne à l'autre
- Améliorer l'accessibilité des informations

Charges de réalisation des automates trop importantes

- Problèmes techniques à contourner
- Ecriture de scripts trop détaillés et difficiles à réutiliser
- Interface applicative instable
- Maintenance des scripts couteuse

Implication limitée des testeurs

- Testeurs ayant peur du niveau de technicité
- Développeurs / automaticiens ayant élaboré les tests en complète autonomie
- Absence de lien entre le référentiel de test manuel et les tests automatisés

Perception négative du ratio coûts / résultats

- Attentes irréalistes sur le périmètre à couvrir
- Absence de capitalisation
- Résultats de l'automatisation non mesurés

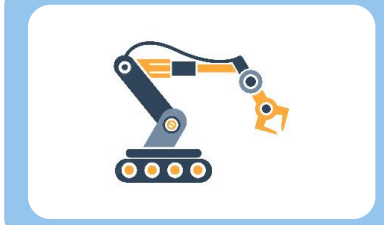
Les facteurs clé d'une automatisation réussie

Equipe intégrée

- L'automatisation consiste en un ensemble d'activités
- La responsabilité de chaque activité est confiée à un acteur de l'équipe (testeur, développeur…)
- Le sponsor (client final ou référent) est impliqué dans la démarche

Pilotage

- Il faut un « porteur » du sujet (coordinateur)
- Un processus de collection et priorisation des demandes est nécessaire : gestion du backlog d'automatisation
- Les coûts et les gains obtenus doivent être mesurés en permanence



Industrialisation

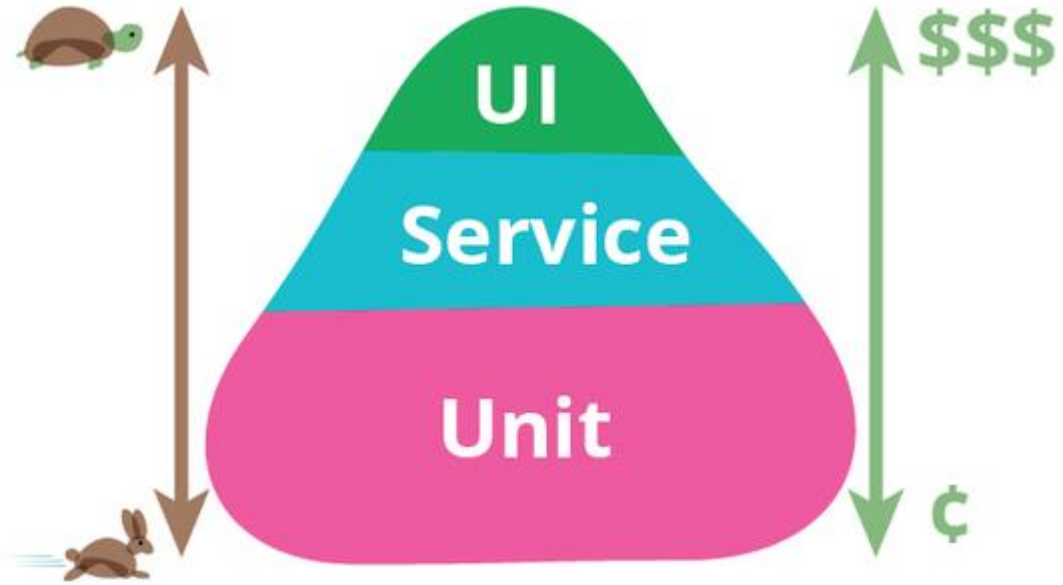
- L'automatisation est gérée comme un projet de développement
- L'objectif poursuivi reste néanmoins la création d'un référentiel de tests réutilisables
- La mise en place d'un référentiel (Framework) est vitale
- L'exécution des tests automatisés doit se faire au plus près du développement

Les niveaux d'automatisation

Quel que soit le cycle de vie utilisé, les tests peuvent être automatisés à plusieurs niveaux :

- Développement : tests unitaires automatisés par les développeurs avant de commit de code
- Intégration : tests de builds automatisés avant de livrer une version aux testeurs
- Système : tests manuels sur l'interface utilisateur, tests de non-régression…
- Acceptation : tests de bout-en-bout, tests de performance…

La principale difficulté reste de savoir quoi automatiser, à quel niveau et dans quelle quantité…



https://martinfowler.com/bliki/TestPyramid.html

Les différents types d'outils

Open Source	Editeur
Coût financier très limité	Investissement financier parfois important (licences + support)
Usage souvent unique	Support de plusieurs technologies
Compétences en développement souhaitables	Outils souvent graphiques (facilité d'utilisation)
Intégration limitée ou à fabriquer soi-même	Souvent en suite intégrée avec des outils du même éditeur ou API
Forge Open source existante et support de la communauté	Accès au support et à la maintenance de l'éditeur
Approche projet ou technologie	Approche globale pour toute l'organisation
Expertise interne	Disponibilité des ressources compétentes

Le coût de l'acquisition des licences des solutions commerciales ne représente parfois qu'un faible pourcentage du coût global de l'automatisation !!

1. Installation et configuration Playwright

Installation environnement

Python 3.11

1. Télécharger la dernière version de Python 3.11 à l'adresse suivante: https://www.python.org/downloads/
2. Une fois téléchargé, lancer l'exécutable pour installer python.
3. Cocher la case « Add Python 3.11 to PATH » puis cliquer sur « Customize installation »
4. Cocher toutes les cases si ce n'est pas déjà fait, puis cliquer sur « Next »
5. Cocher bien la case "Add Python environment variables".
6. Changer le répertoire d'installation vers "C:\Python\Python311". Cliquer sur « Install ».
7. Vérifier que Python s'est bien installé sur l'ordinateur et a correctement mis à jour le PATH en lançant un invité de commande et en exécutant la commande python --version

Pytest

1. Installer pytest en utilisant pip via cette commande : **pip install pytest**
2. Vérifier l'installation de pytest en exécutant la commande suivante dans l'invite de commande : **pytest –version**

Playwright

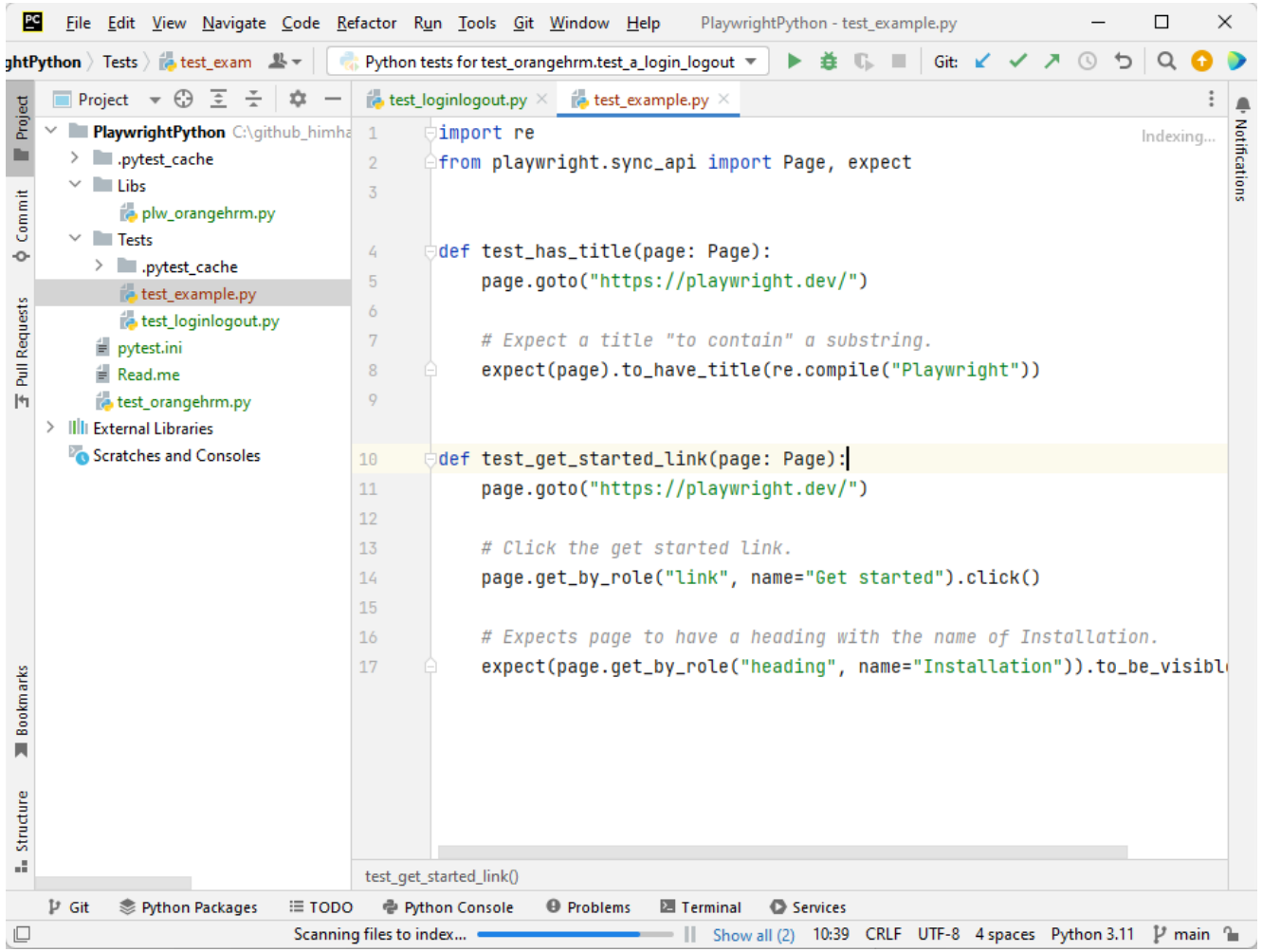
Installer Playwright avec pytest plugin: pip install pytest-playwright
playwright install

PyCharm Community Edition

https://www.jetbrains.com/pycharm/download/?section=windows
https://www.jetbrains.com/pycharm/download/?section=mac

Création du projet

1. Depuis l'écran d'accueil PyCharm, créer un nouveau dossier « PlaywrightPython ». Ce sera le dossier racine du projet
2. En Python, une des conventions est de créer un dossier « ***Tests*** » sous la racine, pour y archiver tous les cas de tests
3. Nous allons également créer un dossier « Libs » pour y stocker les librairies utilisées dans les tests.
4. Dans le dossier test, nous allons créer un premier cas de test et le nommer « ***test_xxx.py*** »



L’écosystème Playwright

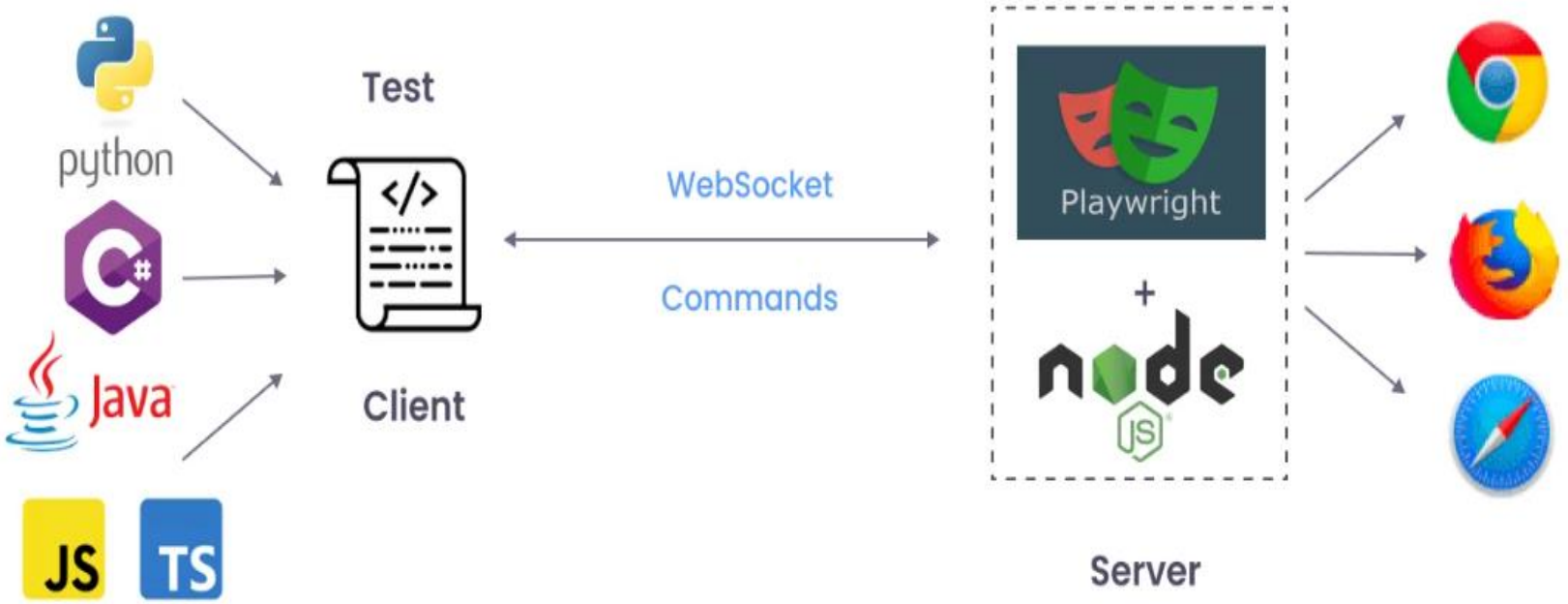
Playwright est aujourd'hui un projet open source soutenu par une communauté croissante d'utilisateurs à travers le monde.

Ce projet est maintenu par une équipe active et propose une gamme d'outils pour l'automatisation des tests d'applications web modernes.

La communauté et les contributeurs de Playwright travaillent en continu pour offrir des fonctionnalités avancées, notamment :

- Support multi-navigateurs : Automatisation des tests sur Chrome, Firefox, Safari et Microsoft Edge.
- Test cross-plateforme : Exécution des tests sur différentes plateformes (Windows, MacOs, Linux).
- Prise en charge de plusieurs langages : Disponible en Python, Node.js, Java et C#.
- Tests en parallèle et isolés : Playwright permet de lancer des tests en parallèle dans des contextes de navigateur isolés. Captures d’écran, vidéos et journaux

Architecture Playwright



Présentation de Playwright

Playwright est un Framework d'automatisation des tests de bout en bout à code source ouvert, issu de Puppeteer, qui est basé sur node.js et maintenu par Microsoft.
Dans l'architecture traditionnelle du WebDriver Selenium, chaque requête HTTP est envoyée séparément et reçoit une réponse JSON, ce qui entraîne une communication en va-et-vient qui ralentit l'ensemble du processus.
Playwright, en revanche, utilise une API unique et un seul Web Socket pour communiquer avec tous les pilotes.
C’est pourquoi Playwright se révèle être l’un des Framework d’automatisation les plus rapides et les plus simples dans le domaine des tests logiciels.

Pourquoi utiliser Playwright

1. **Support de Web Sockets Unique** : Playwright utilise un seul Web Socket pour communiquer avec tous les navigateurs, ce qui améliore la performance et la réactivité des tests. En comparaison, Sélénium peut nécessiter plusieurs canaux de communication, ce qui peut introduire une latence supplémentaire.
2. **Prise en Charge de Navigateurs** Modernes : Playwright offre une prise en charge native et simultanée de plusieurs navigateurs modernes, notamment Chromium, Firefox et WebKit. Cela garantit une couverture étendue et cohérente des différents environnements de navigation sans nécessiter de configurations complexes. Playwright permet l'émulation d'appareil mobile.
3. **Tests Plus Stables et Rapides** : Grâce à son architecture moderne, Playwright réduit les problèmes de synchronisation courants dans Selenium, offrant des tests plus stables et plus rapides. Son modèle de gestion des requêtes et des réponses le rend moins sujet aux échecs dus à des délais de chargement ou à des interactions asynchrones.

Pourquoi utiliser Playwright ? Suite

API Moderne et Facile à Utiliser : Playwright propose une API plus moderne et intuitive qui simplifie la rédaction des scripts de test. Ses fonctionnalités intégrées permettent d'interagir facilement avec des éléments de page (attente automatique), de gérer les contextes de navigateur et de manipuler les cookies et le stockage local.

Automatisation de Scénarios Complexes : Playwright permet de simuler des scénarios utilisateur complexes, comme les interactions avec des pop-ups, des notifications et des frames intégrées, de manière plus fluide que Sélénium. Cela est particulièrement utile pour tester des applications web riches en fonctionnalités. Playwright peut intercepter et modifier les requêtes réseau.

Test de Performances et de Sécurité : Avec Playwright, il est plus facile d'intégrer des tests de performance et de sécurité grâce à des outils intégrés et des configurations simplifiées. Il offre également des capacités avancées pour capturer des vidéos et des captures d'écran des tests, facilitant ainsi le diagnostic des problèmes.

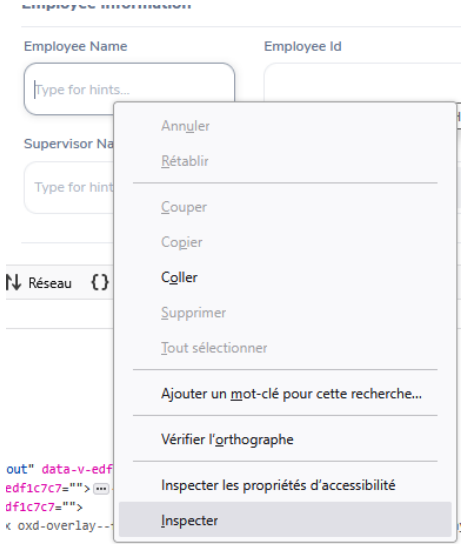
2. Identification des objets Web

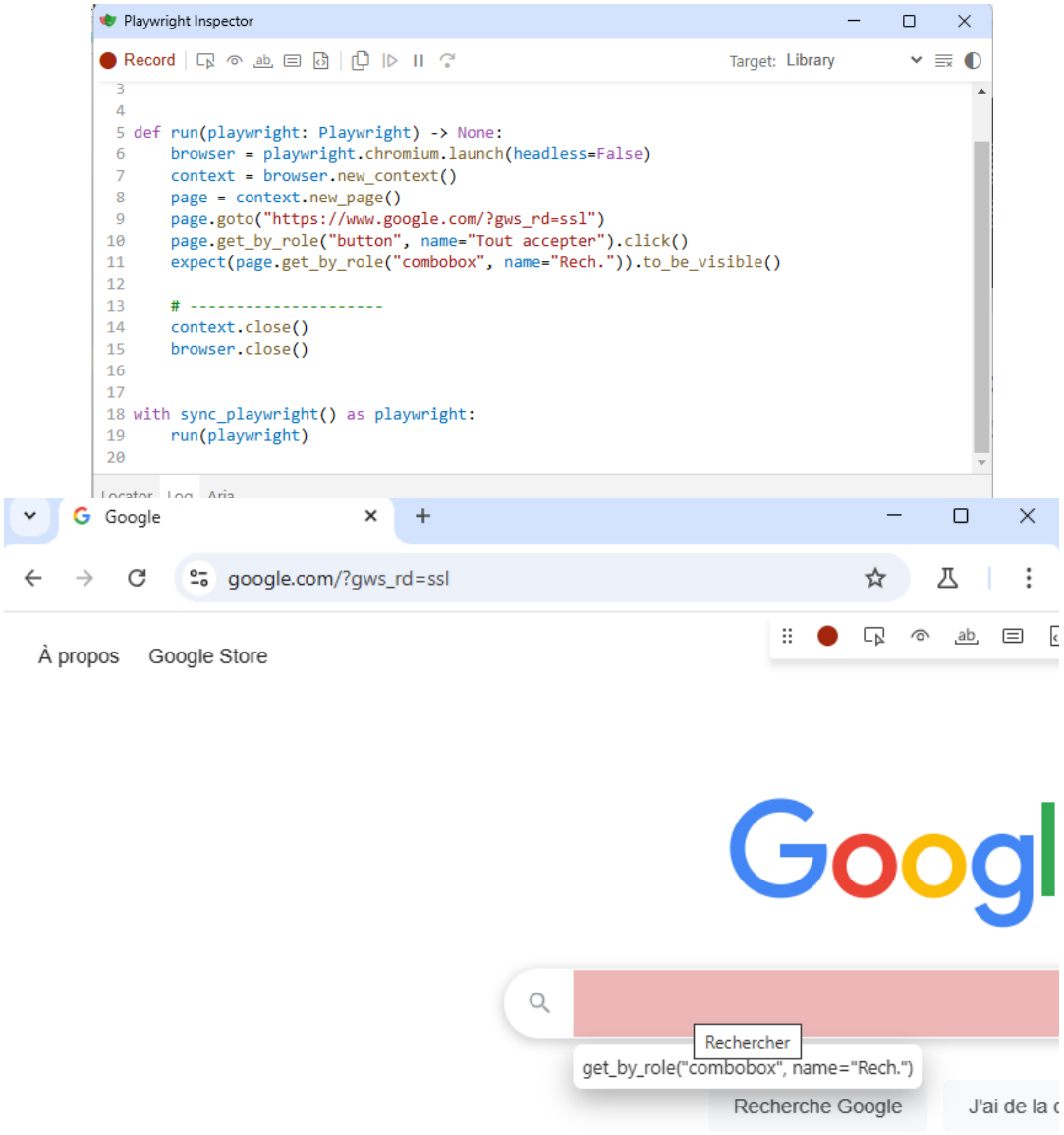
Playwright Test Generator

Test Generator : Playwright inclut un générateur de tests intégré qui permet de créer des scripts d'automatisation rapidement en enregistrant les interactions avec le navigateur. Cette fonctionnalité est conçue pour simplifier la création de tests sans nécessiter une connaissance approfondie de la programmation. Il possède lui-même sa propre interface de développement
Richesse des fonctionnalités : Playwright permet également d'ajuster et d'améliorer les scripts générés pour des scénarios complexes, en tirant parti de ses capacités avancées comme la gestion des contextes de navigation et des interactions dynamiques.

Inconvénient :
Les localisateurs générés ne sont pas toujours pertinents.
Retravailler les localisateurs avec les devtools du navigateur ou avec un plugin.

Exécuter :
playwright codegen





Localisateurs courants

	Selenium	Playwright	Explication
ID	driver.find_element(By.ID, "id_value")	page.locator("#id_value")	Identifie un élément par son attribut ID unique.
Nom	driver.find_element(By.NAME, "name_value")	page.locator("[name='name_value']")	Cible un élément par son attribut name.
Classe	driver.find_element(By.CLASS_NAME, "class_value")	page.locator(".class_value")	Sélectionne des éléments par leur classe CSS.
Texte du lien	driver.find_element(By.LINK_TEXT, "link_text")	page.get_by_text("link_text")	Trouve des liens par leur texte visible.
XPath	driver.find_element(By.XPATH, "xpath_expression")	page.locator("xpath=xpath_expression")	Utilise des expressions XPath pour localiser des éléments.
CSS Selector	driver.find_element(By.CSS_SELECTOR, "css_selector")	page.locator("css_selector")	Emploie des sélecteurs CSS pour cibler des éléments.
Label	Non disponible nativement	page.get_by_label("label_text")	Trouve des éléments par le texte de leur label associé.
Placeholder	Non disponible nativement	page.get_by_placeholder("placeholder_text")	

Localisateurs spécifiques Playwright

page.get_by_text() : Localise un élément par son texte visible.

page.get_by_role() : Trouve des éléments basés sur leur rôle ARIA.

page.get_by_label() : Cible des éléments associés à un label spécifique.

page.get_by_placeholder() : Localise des éléments input par leur texte placeholder.

page.get_by_alt_text() : Trouve des images par leur texte alternatif.

page.get_by_title() : Sélectionne des éléments par leur attribut title.

page.get_by_test_id() : Cible des éléments avec un attribut data-testid spécifique.

page.frame_locator() : Localise des éléments à l'intérieur d'un iframe.

page.get_by_aria_label() : Trouve des éléments par leur attribut aria-label.

page.get_by_aria_role() : Sélectionne des éléments basés sur leur rôle ARIA, similaire à get_by_role mais plus spécifique.

Localisateurs CSS

Sélecteur	Description
div	Recherche tous les éléments de type div
div.class	Recherche tous les éléments de type div contenant une classe appelée "class"
.class	Recherche tous les éléments contenant une classe appelée "class"
#objectId	Recherche tous les éléments qui ont un Id égal à "objectId"
input[name]	Recherche tous les éléments de type "input" contenant l'attribut "name"
input[name="a"]	Recherche tous les éléments de type "input" dont l'attribut "name" prend la valeur "a"
input[name*="a"]	Recherche tous les éléments de type "input" dont l'attribut "name" contient la valeur "a"

Sélecteur	Description
p:nth-child(n)	Recherche tous les éléments de type "p" qui sont les n-ièmes enfants de leur parent
p:nth-of-type(n)	Recherche tous les éléments de type "p" qui sont les n-ièmes enfants de type "p" de leur parent

https://www.w3schools.com/cssref/css_selectors.php

Localisateurs XPATH

Expression	Description
/html/body	Recherche la section body directement sous la racine html
//input	Recherche tous les éléments input dans la page
//input[@type='text']	Recherche tous les éléments input dont l'attribut type prend la valeur 'text'
//input[contains(@class, 'result')]	Recherche tous les éléments input dont l'attribut 'class' contient la valeur 'result'
//input[starts-with(@class, 'result')]	Recherche tous les éléments input dont l'attribut 'class' commence par la valeur 'result'
//input[not(contains(@class, 'hidden'))]	Recherche tous les éléments input dont l'attribut 'class' n'a pas la valeur 'hidden'
//img[@width<20][@height<20] //img[@width<20 and @height<20]	Condition AND
//input[@name='q' or @id='input']	Condition OR
//*[text()='Search']	Recherche tous les éléments dont le texte est égal à 'Search'
//*[contains(text(),'Search')]	Recherche tous les éléments dont le texte contient 'Search'
//div[contains(.,'Search')]	Recherche tous les éléments de type div et leurs descendants dont le texte contient 'Search'

https://www.w3schools.com/xml/xpath_syntax.asp

Commandes utiles

Pour Playwright

- page.url()
- page.goto("https://example.com")
- browser.close()
- page.reload()
- page.title()
- page.screenshot(path="screenshot.png")
- page.set_viewport_size({"width": 1280, "height": 720})

Pour les éléments web

- locator.clear()
- locator.click()
- page.locator("css_selector").all()
- locator.fill("your text")
- locator.bounding_box()
- locator.get_attribute("attribute")
- locator.text_content()
- locator.is_visible()

Exercice 1 : Utilisation codegen et localisateurs

0.1. Lancer playwright codegen

0.2. Réaliser le test manuel sur orangehrm (https://opensource-demo.orangehrmlive.com)

Login → Création Employé → Saisie du détail → Recherche Emplyé par nom → Modification des coordonnées → Recherche Employé par matricule → Suppression de l'employé → Logout

0.4. Analyser le code généré et modifier les localisateurs non pertinents.

0.5. Intégrer Login et Logout dans un test et l'exécuter.

0.6. Analyser les résultats et corriger les erreurs

Utiliser le code de l'exemple https://playwright.dev/python/docs/intro comme modèle

3. Les assertions

Les assertions dans Playwright

Les assertions Playwright permettent de vérifier si des conditions spécifiques sont remplies dans l'application testée. Elles sont essentielles pour confirmer que l'application se comporte comme prévu lors des tests.

Types d'assertions

Assertions de base

```
# Vérifier si une valeur est vraie
expect(success).to_be_truthy()
# Vérifier l'égalité
expect(result).to_equal(expected_value)
# Vérifier si une chaîne contient une sous-chaîne
expect(text).to_contain("expected substring")
```

Assertions de base

```
Assertions spécifiques au web
# Vérifier le titre de la page
expect(page).to_have_title("Expected Title")
# Vérifier la visibilité d'un élément
expect(page.locator("button#submit")).to_be_visible()
# Vérifier le texte d'un élément
expect(page.locator("h1")).to_have_text("Welcome")
# Vérifier l'URL de la page
expect(page).to_have_url("https://example.com/dashboard")
```

Assertions sur les localisateurs

```
# Vérifier si une case à cocher est cochée
expect(page.locator("#terms")).to_be_checked()
# Vérifier si un élément est activé
expect(page.locator("button#submit")).to_be_enabled()
# Vérifier le nombre d'éléments
expect(page.locator("li.item")).to_have_count(5)
# Vérifier la valeur d'un champ de saisie
expect(page.locator("input#email")).to_have_value("user@example.com")
```

Les assertions courantes

- **toBe()** : Vérifie l'égalité stricte pour les valeurs primitives
- **toEqual()** : Effectue une comparaison profonde pour les objets et les tableaux
- **toContain()** : Vérifie si une chaîne ou un tableau contient une valeur spécifique
- **toBeTruthy()** : Vérifie si une valeur est considérée comme vraie
- **toHaveText()** : Vérifie le texte d'un élément web
- **toHaveTitle()** : Vérifie le titre de la page
- **toBeVisible()** : Vérifie si un élément est visible sur la page
- **toHaveCount()** : Vérifie le nombre d'éléments correspondant à un sélecteur
- **toHaveAttribute()** : Vérifie si un élément possède un attribut spécifique
- **toHaveValue()** : Vérifie la valeur d'un champ de formulaire

Exercice 2 : Ajouter les assertions

0.1. Reprendre le code généré dans l'exercice 1

0.2. Compléter les tests avec : Création Employé → Saisie du détail → Recherche Emplyé par nom

0.4. Ajouter les assertions pertinentes

0.5. Rendre le code modulaire en créant une librairie pour OrangeHRM

0.6. Exécuter le test, analyser les résultats et corriger les erreurs

4. Utilisations des données

Fixture Pytest

L’utilisation des données (CSV ou autre) dans Playwright utilise le mécanisme de fixture dans pytest.

Les fixtures dans pytest sont des fonctions qui permettent de préparer l'environnement de test et de fournir des données ou des objets réutilisables pour les tests.

Les fixtures peuvent avoir différentes portées (scope) : session / module / function

Les fixtures peuvent aussi être utilisées pour automatiser les actions de logins et de logout pour chaque test (avec le fichier conftest.py)

Fixture Pytest : exemples fichier CSV

 # Exécution d'un test pour chaque ligne

```
import pytest
import csv

with open('logins.csv', 'r') as f:
    reader = csv.DictReader(f)
    data = list(reader)

@pytest.mark.parametrize("row", data)
def test_eval(row):
    print (row)
```

 # Boucle automatique sur toutes les lignes

```
import pytest
import csv

@pytest.fixture
def test_data():
    with open('test_data.csv', 'r') as f:
        reader = csv.DictReader(f)
        data = list(reader)
    return data

def test_example(page: Page, test_data):
    for row in test_data:
                print(row)
```

Fixture Pytest : utilisation conftest.py

conftest.py

```
import pytest
from playwright.sync_api import Browser, BrowserContext, expect


@pytest.fixture(scope="module")
def authenticated_context(browser: Browser):
    context = browser.new_context()
    page = context.new_page()

    # SETUP : Login
    page.goto("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")
    page.get_by_role("textbox", name="Username").fill("admin")
    page.get_by_role("textbox", name="Password").fill("admin123")
    page.get_by_role("button", name="Login").click()
    expect (page.locator("p.oxd-userdropdown-
name")).to_have_text("Astrogildo Silvasauro")

    yield context  # Exécution des tests

    # TEARDOWN : Logout
    page.locator("p.oxd-userdropdown-name").click()
    page.get_by_role("menuitem", name="Logout").click()
    context.close()
```

 # dans chaque test

```
import re
from playwright.sync_api import Page, expect

def test_has_title(page: Page, authenticated_context):
        ajout_employee()
```

Exercice 2 : Ajouter les assertions

0.1. Reprendre le code généré dans l'exercice 2

0.2. Compléter les tests avec : Création Employé → Saisie du détail → Recherche Emplyé par nom

0.4. Déplacer le login / logout dans un fichier conftest.py

0.5. Créer 20 employés à partir d'un fichier CSV (utiliser Faker pour générer les noms prénoms automatiquement)

0.6. Exécuter le test, analyser les résultats et corriger les erreurs

6. Cas pratique

Cas pratique : taiga.io

Reprendre le projet créé dans le TD Postman sur le site taiga.io (https://tree.taiga.io)

Automatiser le scénario suivant :

1. Se connecter à l'application
2. Sélectionner le projet
3. Afficher la liste des anomalies (issues) du projet
4. Créer une anomalie : Subject / Description / Type / Severity / Priority
5. Créer une liste d'anomalie avec une variation sur le type, la sévérité et la priorité
6. Appliquer le filtre Type = Bug et Severity = Critical
7. Rechercher une anomalie par la référence et la supprimer

Merci

→

onepoint.
beyond the obvious