TP1 – Introduction à Playwright

Objectifs

- Comprendre les principes de base des tests E2E (End-to-End)
- Installer et configurer Playwright
- Découvrir la structure d'un projet Playwright
- Utiliser le mode enregistrement de Playwright (Codegen)
- Exécuter et interpréter un test automatisé

1. Introduction aux tests E2E

1.1 Définition

Un test E2E (End-to-End) vérifie le bon fonctionnement complet d'un parcours utilisateur dans une application, depuis l'interface jusqu'aux interactions serveur éventuelles.
L'objectif est de garantir que toutes les couches du système fonctionnent ensemble.

1.2 Pourquoi Playwright ?

Playwright est un framework d'automatisation développé par Microsoft, permettant de :

- piloter les navigateurs Chromium, Firefox et WebKit ;
- effectuer des tests rapides, fiables et parallélisés ;

- bénéficier d’un mode record (Codegen) et d’un support TypeScript natif.

Lien -> https://playwright.dev/

2. Installation et création d’un projet Playwright

2.1 Prérequis

- Node.js version ≥ 18
- Un éditeur de code (VS Code recommandé)

2.2 Initialisation du projet

Créer un dossier de travail et initialiser Playwright :

```
mkdir tp1-playwright
cd tp1-playwright
npm init playwright@latest
```

2.3 Structure générée

Le projet contient :

- `playwright.config.ts` : configuration globale (navigateurs, timeout, etc.)
- `tests/example.spec.ts` : exemple de test
- `package.json` : dépendances et scripts

3. Premier test Playwright

3.1 Le script du test

Créer un fichier `tests/todo.spec.ts` et y copier le test suivant :

```
import { test, expect } from '@playwright/test';

test('ajouter une tâche TODO', async ({ page }) => {
  await page.goto('https://demo.playwright.dev/todomvc');
  await page.getByPlaceholder('What needs to be done?').fill('Implémenter un test E2E');
  await page.keyboard.press('Enter');
  await expect(page.getByText('Implémenter un test E2E')).toBeVisible();
});
```

3.2 Exécution du test

Exécuter le test :

```
npx playwright test tests/todo.spec.ts
```

Consulter le rapport:

```
npx playwright show-report
```

Exécution en mode visible (headed) :

```
npx playwright test tests/todo.spec.ts --headed
```

Trop rapide pour voir ? => Ajouter pause() dans le test :

```
await page.pause();
```

Resume pour continuer

Filtrer par projet:

```
npx playwright test tests/todo.spec.ts --headed --project=chromium
```

4. Les sélecteurs Playwright

4.1 Rôle des sélecteurs

Les sélecteurs permettent à Playwright d'identifier les éléments du DOM sur lesquels interagir (clic, saisie, vérification…).

4.2 Types de sélecteurs

Type	Exemple	Description
CSS	`page.locator('button.submit')`	Sélecteur standard CSS
Text	`page.getByText('Se connecter')`	Recherche par texte visible
Role (ARIA)	`page.getByRole('button', { name: 'Envoyer' })`	Recherche par rôle d'accessibilité
Label	`page.getByLabel('Email')`	Associe un champ à son label
Placeholder	`page.getByPlaceholder('Mot de passe')`	Recherche un champ par placeholder
Test ID	`page.getByTestId('user-card')`	Utilise un attribut `data-testid`

4.3 Bonnes pratiques

- **Privilégier `getByRole` et `getByLabel`** : plus robustes et accessibles.
- Éviter les sélecteurs CSS ou XPath trop complexes.
- Ajouter des `data-testid` dans le code source pour cibler des éléments spécifiques.

5. Découverte du mode Record (Codegen)

5.1 Lancer Codegen

Playwright propose un outil d'enregistrement automatique du code :

```
npx playwright codegen https://demo.playwright.dev/todomvc
```

Une fenêtre de navigateur s'ouvre :

- Les actions effectuées (clics, saisies, assertions) sont traduites en code TypeScript.
- Le code généré apparaît en temps réel dans la console.

5.2 Copier le code généré

Copier le code produit par Codegen et le placer dans un nouveau fichier :
`tests/generated.spec.ts`

Exemple de résultat :

```
import { test, expect } from '@playwright/test';

test('scenario enregistré', async ({ page }) => {
  await page.goto('https://demo.playwright.dev/todomvc');
  await page.getByPlaceholder('What needs to be done?').click();
  await page.getByPlaceholder('What needs to be done?').fill('Acheter du lait');
  await page.keyboard.press('Enter');
  await expect(page.getByText('Acheter du lait')).toBeVisible();
});
```

Exécuter ensuite ce test pour vérifier son bon fonctionnement.

6. Refactorisation du code généré

Le code produit automatiquement par Codegen doit être amélioré :

- Nettoyer les lignes inutiles
- Remplacer les sélecteurs CSS par des sélecteurs sémantiques (`getByRole` , `getByText`)
- Ajouter des commentaires
- Renommer les tests de manière explicite

Exemple :

```
test('ajout d'une tâche', async ({ page }) => {
  await page.goto('https://demo.playwright.dev/todomvc');
  await page.getByRole('textbox', { name: 'What needs to be done?' }).fill('Faire les courses');
  await page.keyboard.press('Enter');
  await expect(page.getByText('Faire les courses')).toBeVisible();
});
```

7. Exercice pratique

Créer un nouveau test intitulé `tests/multiple-tasks.spec.ts` avec le scénario suivant :

1. Ouvrir la page TodoMVC
2. Ajouter deux tâches : "Acheter du pain" et "Aller courir"
3. Supprimer la tâche "Acheter du pain"
4. Vérifier que :
    - "Aller courir" est toujours visible
    - "Acheter du pain" a disparu

Exécuter le test en mode `--headed` pour visualiser les étapes.

8. Conclusion

À la fin de ce TP, vous devez être capables de :

- Initialiser un projet Playwright
- Lancer un test automatisé simple
- Utiliser l’outil Codegen pour générer du code
- Nettoyer et exécuter des scénarios d’automatisation basiques