Installation & Utilisation de Playwright

Installation

npm init playwright@latest
par défaut typescript
le nom du dossier de test
ajouter github workflow
et le brower

run le test d'exemple :

```
PS D:\Projet Perso\QuestBoard\QuestBoard> npx playwright tes

Running 6 tests using 4 workers
  6 passed (24.4s)

To open last HTML report run:

  npx playwright show-report
```

ensuite pour voir les résultats en détails il y'a un html report qui va être généré,
il faut entrer la commande `npx playwright show-report` .
On peut lancer un test en UI Mode qui va permettre une meilleur expérience avec plus de fonctionnalités.

Update

`npm install -D @playwright/test@latest`
`npx playwright install --with-deps`

pour aussi voir la version utilisé : `npx playwright --version`

Ecrire des tests

Les tests Playwright servent à faire des actions et s'assuré de l'état.
Doc Playwright

Locator :

Pour charger la page : `await page.goto('https://playwright.dev/');`
Interaction : `await page.getByRole('link', { name: 'Get started' }).click();`
Pour les actions basiques sur la doc tout est référencé.

Assertions : c'est une forme de expect function pour tester des conditions, il y'a aussi des async matchers qui vont attendre jusqu'à la condition excepté soit réalisé. exemple : `await expect(page).toHaveTitle(/Playwright/);` .

Test Isolation : Les pages sont isolés des tests à cause du contexte du navigateur qui est équivalent à un nouveau profil navigateur où tous les tests ont un environnement frais même en faisant plusieurs tests dans un seul navigateur. exemple :
`test('example test', async ({ page }) => {`
`// "page" belongs to an isolated BrowserContext, created for this specific test.`
`});` .

Test Hooks : ça sert à regrouper des test pour eviter de faire plusieurs blocs c'est pratique et il faut l'utiliser. Exemple :
`test.describe('navigation', () => {`
`test.beforeEach(async ({ page }) => {`
`// Go to the starting url before each test.`
`await page.goto('https://playwright.dev/');`
`});` .

Générer des tests :

`codegen` : Il va regarder les page rendered et trouver le locator recommandé, priorisé les rôles, le text et les tests id locators.

Avec le text generator on peut :

- Faire des actions comme click ou seulement interagir avec la page
- Assertions en cliquant sur un des icones de la toolbar et cliquer sur un élément de la page pour l'asserter contre on peut choisir :
  - - `'assert visibility'` to assert that an element is visible
  - `'assert text'` to assert that an element contains specific text
  - `'assert value'` to assert that an element has a specific value

Quand on a finit d'interagir avec la page on appuis sur le bouton record pour arrêter l'enregistrement et utiliser le bouton copy pour généré le code à l'éditeur
Utiliser le bouton clear pour clear le code et recommencer à record et une fois finit on ferme Playwright inspector window.

Ce que je dois test sur mon QuestBoard :

- Ouverture du site (si il s'ouvre bien, par exemple accès à la page d'accueil)
- Création d'un objectif (regarder si il est bien ajouté, si la barre de progression est mise à jour etc..)
- Edition d'un objectif
- Validation d'un objectif
- Suppression d'un objectif

1. **Ouverture du site & Titre du site**

```
test('has title', async ({ page }) => {
  await page.goto('/');

  // Expect a title "to contain" a substring.
  await expect(page).toHaveTitle(/QuestBoard/);
});
```

2. **Création d'un objectif**

```
test("fill the input and valid input", async ({ page }) => {
  test.setTimeout(60000);
  await page.goto("/");
  await page.getByTestId("input-quetes").fill("example value");
  await page.getByRole("button", { name: /ajouter/i }).click();
  await page.waitForTimeout(500); // obliger car fail 1/2 pour firefox
  // await expect(page.getByTestId("list-quetes")).toHaveValue("example value");
  await expect(page.getByText("example value")).toBeVisible(); // le mieux aurait été de vérifier avec le testid
  // il faut qu'il y a seulement un example value sinon il ne sait pas lequel choisir pour verifier
});
```

3. **Edition d'un objectif :**

```
test("edit a quest", async ({ page }) => {
  await page.goto("/");
  await page
    .getByTestId(/^btn-edit-/)
    .first()
    .click();
  await page.getByTestId("textfield-edit").fill("edited");
  await page
    .getByTestId(/^btn-sauvegarder/)
    .first()
    .click();
  await expect(page.getByText("edited")).toBeVisible();
});
```

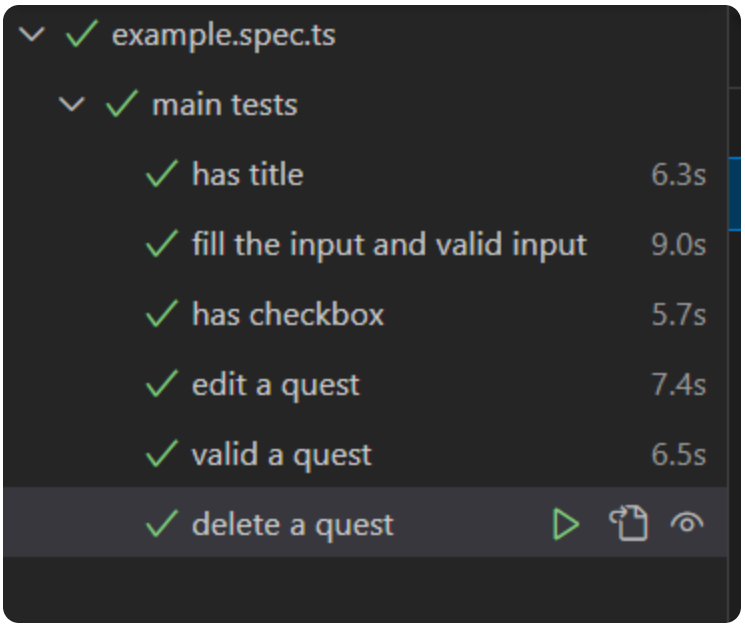
4. **Validation d'un objectif**

```
// je voulais faire le test avec les ids mais vue que ceux de firebase sont aléatoires et trop long pas possible
test("valid a quest", async ({ page }) => {
  await page.goto("/");
  await page.getByRole("checkbox").nth(0).check();
  await page.getByTestId("btn-valid").click();
  await expect(page.getByText("edited")).not.toBeVisible(); // edited car il valide la première quête donc celle modifié en edited
});
```

5. **Suppression d'un objectif**

```
// je voulais faire le test avec les ids mais vue que ceux de firebase sont aléatoires et trop long pas possible
test("delete a quest", async ({ page }) => {
  await page.goto("/");
  await expect(page.getByText("example value")).toBeVisible();
  await page.getByTestId("btn-close").nth(0).click();
  await expect(page.getByText("example value")).not.toBeVisible;
});
```

Fin du test


example.spec.ts
main tests
has title 6.3s
fill the input and valid input 9.0s
has checkbox 5.7s
edit a quest 7.4s
valid a quest 6.5s
delete a quest