

Syllabus	Doc E2E tests	Planning et Fonctionnement	Démarche centrée utilisateurs
Modalités de rendus et évaluations	Sujet	Des exemples de Vidéos	Cours
Back		Technologie Angular NodeJs	

Documentation E2E tests

Cette documentation est divisée en 3 parties :

- [Étapes démarrage playwright](#)
- [Étapes d'installation de playwright sur votre projet](#)
- [Cheat codes : documentation technique sur playwright.](#)

Pour vous aider, vous pouvez consulter les 2 repositories suivant :

- [Le starter de ps6](#) : qui contient la configuration de playwright et deux exemples de tests simples
- [La correction de ps6](#) : qui contient la configuration de playwright et un exemple plus avancé de test end to end (E2E).

Étapes démarrage playwright

Voici les étapes pour démarrer avec les tests e2e sur votre projet :

- Installation de playwright sur votre projet.
- Définition des critères d'importance de votre projet et définition de la stratégie de testing. En fonction des éléments importants de votre application, vous devez définir des niveaux d'importance pour chacun de vos scénarios de votre app. Une fois vos scénarios identifiés et priorisés, débutez l'écriture de vos tests en commençant par les scénarios les plus critiques. Vous itérez ensuite sur les scénarios suivants en fonction de leur importance et en séparant bien votre travail.
- Découverte de playwright avec la lecture :
 1. De la documentation cheat codes de playwright
 2. Des exemples de codes sur les repo du starter-ps6 et de la correction-ps6
- Écriture de vos premiers tests :
 1. Commencez par des exemples simples comme ceux proposés dans le starter-ps6. Lisez des éléments de page, cliquez sur des boutons etc...
 2. Itérez ensuite sur des scénarios e2e en définissant des fichiers fixtures pour chacun des composants (slide 32 du cours) et en écrivant des tests pour vos scénarios préalablement définis.

Intégration playwright

Les liens partagées ci-dessous concernant le démarrage de playwright proviennent du repository qui a fait office de base pour votre projet : <https://github.com/NablaT/starter-ps6-full-stack>

Clonez le pour mieux comprendre l'intégration et le premier exemple de test.

Note : Une seule personne du groupe a besoin de faire l'intégration de playwright mais la lecture de la doc d'intégration va vous permettre de mieux comprendre comment les tests fonctionnent dans votre projet. Pour intégrer playwright dans votre projet, vous pouvez suivre ces étapes :

Côté front-end :

1. Installer playwright en lançant dans le répertoire `front-end` la commande : `npm install --save-dev playwright @playwright/test`
2. À la racine de votre front-end (là où vous êtes), créer le fichier : `playwright.config.ts` avec le contenu suivant :

```
import { PlaywrightTestConfig } from '@playwright/test';

const config: PlaywrightTestConfig = {
  reporter: [['html', { open: 'always' }]],
  use: {
    headless: false,
    viewport: { width: 1280, height: 720 },
    ignoreHTTPSErrors: true,
    video: 'on-first-retry',
    screenshot: 'only-on-failure',
    launchOptions: {
      slowMo: 1000,
    }
  },
};

export default config;
```

3. Dans le `package.json`, ajoutez la commande : `"test:e2e": "npx playwright test"`

```
{
  "name": "front-end",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test",
    "test:e2e": "npx playwright test"
  },
  "private": true,
  "dependencies": {
```

4. Créez un dossier **e2e** dans le dossier front-end et, à l'intérieur, créez un nouveau dossier scénarios. Si vous avez déjà un dossier e2e, supprimez son contenu.
5. Dans le dossier e2e, créez un fichier de config: `e2e.config.ts` avec simplement l'export de l'url :

```
export const testUrl = 'http://localhost:4200';
```

6. Dans le dossier scénarios, créez un premier fichier de scénario de test. Par exemple : [initial-test-scenario.spec.ts avec son contenu](#)

```
import { test, expect } from '@playwright/test';
import { testUrl } from 'e2e/e2e.config';

// This file is here to test the playwright integration.
test.describe('Initial test display', () => {
  test('Basic test', async ({ page }) => {
    await page.goto(testUrl);
    // Let's try with something you don't have in your page.
    const pageTitle = await page.getByRole('heading', { name: 'AGreatHeadingNameYouDontHave' });
    // It should not be visible as you don't have it in your page.
    expect(pageTitle).not.toBeVisible();
    // Test case pass? Means the playwright setup is done! Congrats!
  });
});
```

7. Lancez les tests : **npm run test:e2e** et vous devriez voir :

```
> starter-quiz-two@0.0.0 test:e2e
> npx playwright test

Running 1 test using 1 worker

✓ 1 e2e/scenarios/initial-test.spec.ts:6:7 › Initial test display › Basic test (481ms)

1 passed (1.0s)
```

8. Pour l'écriture de vos fixtures par composant, vous aurez besoin de la classe **E2EComponentFixture**. Créer un fichier **e2e-component.fixture.ts** dans le dossier e2e [avec ce contenu](#) :

```
import { Page } from '@playwright/test';

export class E2EComponentFixture {
  protected page: Page;

  constructor(page: Page) {
    this.page = page;
  }
}
```

9. Vous pouvez maintenant écrire vos e2e tests !

10. Vous pouvez rajouter dans le .gitignore les dossiers qui contiendront les rapports de tests pour ne pas les avoir sur votre repository git.

E2E folder

/playwright-report

/test-results

1. Vous pouvez mettre à jour vos **README.md** côté front-end et back-end pour documenter les commandes:

Run the end to end tests

Before running the tests, you need to run your front-end and back-end:

- 1) Run your back-end: `npm run start:e2e`
- 2) Run your front-end: `npm run start`
- 3) Run the tests: `npm run test:e2e`

Côté back-end :

1. Installer les dépendances suivantes :

npm install --save-dev cross-env rimraf mkdirp

2. Ouvrir le fichier package.json et rajouter la commande:

"start:e2e": "rimraf ./database/e2e && mkdirp ./database/e2e && cross-env DB_FOLDER=e2e/ node app/index.js"

```
{
  "name": "backend-ps6-starter-quiz",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node app/index.js",
    "dev": "./node_modules/.bin/nodemon app/index.js",
    "lint": "./node_modules/.bin/eslint -c .eslintrc.js app/",
    "lint:fix": "./node_modules/.bin/eslint -c .eslintrc.js app/ --fix",
    "start:e2e": "rimraf ./database/e2e && mkdirp ./database/e2e && cross-env DB_FOLDER=e2e/ node app/index.js"
  },
  "precommit": [
```

3. Dans le fichier **app/utills/base-model.js**, modifier la définition de `this.filePath` ainsi :

this.filePath = `\${__dirname}/../database/\${process.env.DB_FOLDER ?? ''}\${this.name.toLowerCase()}.data.json`

```
module.exports = class BaseModel {
  constructor(name, schema) {
    if (!name) throw new Error('You must provide a name in constructor of BaseModel')
    if (!schema) throw new Error('You must provide a schema in constructor of BaseModel')
    this.schema = Joi.object().keys({ ...schema, id: Joi.number().required() })
    this.items = []
    this.name = name;
    this.filePath = `${__dirname}/../database/${process.env.DB_FOLDER ?? ''}${this.name.toLowerCase()}.data.json`
    this.load()
  }
  load() {
```

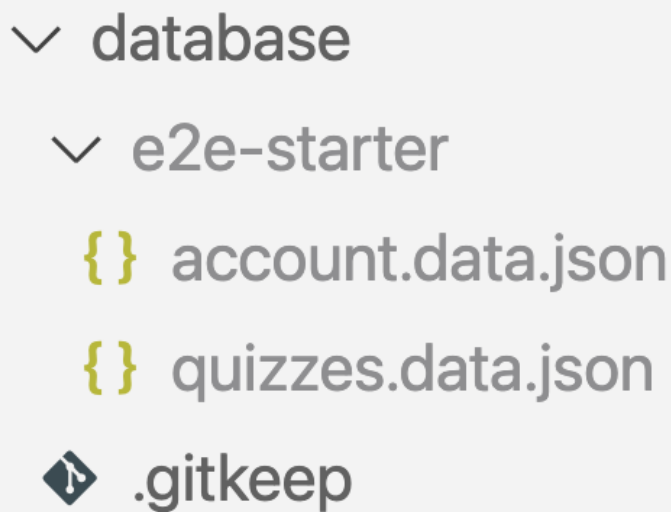
4. Lancer votre serveur pour vos tests avec la commande : **npm run start:e2e**

Démarrer les tests avec des données :

Si vous souhaitez démarrer vos tests avec des données au démarrage de votre serveur pour les utiliser dans vos tests, vous suivre les étapes suivantes :

1. Créer un dossier e2e-starter dans votre dossier back-end/database. Mettez à l'intérieur vos fichiers avec vos

données :



```
database
└─ e2e-starter
   ├── account.data.json
   ├── quizzes.data.json
   └─ .gitkeep
```

2. Installer la dépendance copyfiles: **npm install --save-dev copyfiles**
3. Mettez à jour votre **package.json** dans votre backend pour ajouter la copie des fichiers :

```
"start:e2e": "rimraf ./database/e2e && copyfiles -f ./database/e2e-starter/* ./database/e2e  
./database/e2e && cross-env DB_FOLDER=e2e/ node app/index.js"
```

Doc copyfiles : <https://www.npmjs.com/package/copyfiles>

Cheat codes

Documentation : <https://playwright.dev/docs/locators#locate-by-css-or-xpath>

1. **Locate by role** : Localiser un élément en fonction de son rôle, tel que perçu par les utilisateurs et les technologies d'assistance.

Exemple de code pour localiser par rôle :

```
...
```

```
await page.getByRole('button', { name: 'Sign in' }).click();
```

```
...
```

Doc: [Locators | Playwright](#)

2. **Locate by label:** Localiser un contrôle de formulaire en fonction de son libellé associé.

Exemple de code pour localiser par libellé :

```
...
```

```
await page.getByLabel('Password').fill('secret');
```

```
...
```

Doc: [Locators](#) | [Playwright](#)

3. **Locate by placeholder:** Localiser un élément de formulaire en fonction de son texte de placeholder.

Exemple de code pour localiser par placeholder :

```
...
```

```
await page.getByPlaceholder("name@example.com").fill("playwright@microsoft.com");
```

```
...
```

Doc: [Locators](#) | [Playwright](#)

4. **Locate by text:** Trouver un élément en fonction du texte qu'il contient.

Exemple de code pour localiser par texte :

```
...
```

```
await expect(page.getByText('Welcome, John')).toBeVisible();
```

```
...
```

Doc: [Locators](#) | [Playwright](#)

5. **Locate by alt text:** Localiser une image en fonction de son texte alternatif.

Exemple de code pour localiser par texte alternatif :

```
...
```

```
await page.getByAltText('playwright logo').click();
```

```
...
```

Doc: [Locators](#) | [Playwright](#)

6. **Locate by title:** Localiser un élément en fonction de son attribut de titre.

Exemple de code pour localiser par titre :

```
...
```

```
await expect(page.getByTitle('Issues count')).toHaveText('25 issues');
```

```
...
```

Doc: [Locators](#) | [Playwright](#)

7. **Locate by test id:** Localiser un élément en fonction de son identifiant de test personnalisé.

Exemple de code pour localiser par identifiant de test :

```
...

await page.getByTestId('directions').click();

...
```

8. **Locate by CSS or XPath:** Localiser un élément en utilisant des sélecteurs CSS ou XPath.

Exemple de code pour localiser par CSS ou XPath :

```
...

await page.locator('css=button').click();

...
```

Docs: [Locators](#) | [Playwright](#)

Autres éléments moins utile dans le contexte de ps6 (sauf cas spécifique) à découvrir dans la doc : [Locate by alt text](#) | [Locate by test id](#)

Mode debug playwright:

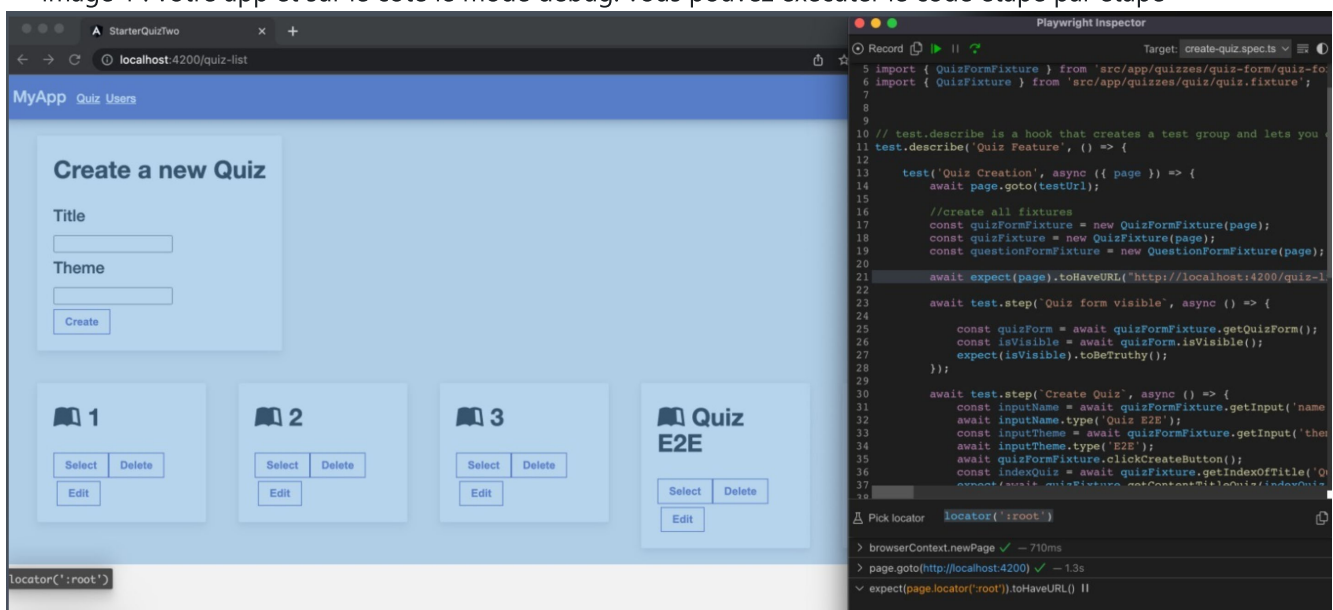
Vous pouvez utiliser le paramètre "debug" pour lancer les E2E en mode debug ce qui vous permettra de voir ce qui se passe étape par étape et d'accéder à la fonction "Pick locator" qui vous aidera à écrire vos fixtures. Exemple des commandes pour lancer en debug :

`npm run test:e2e -- --debug` (Lancer tout les tests)

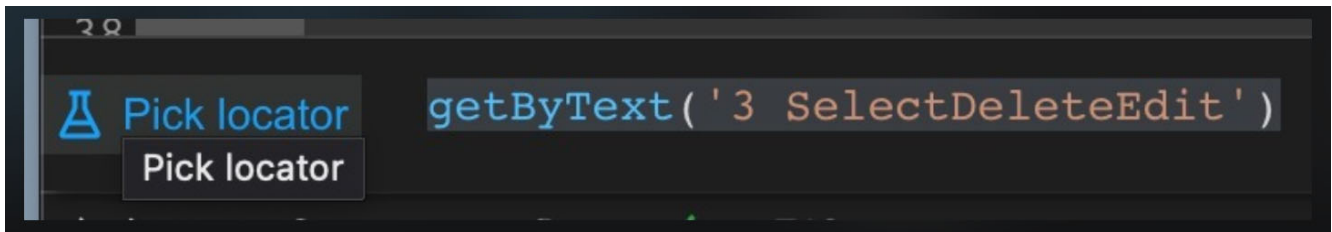
`npm run test:e2e -- --debug home-page-display.spec.ts` (Lancer un seul test)

Illustrations :

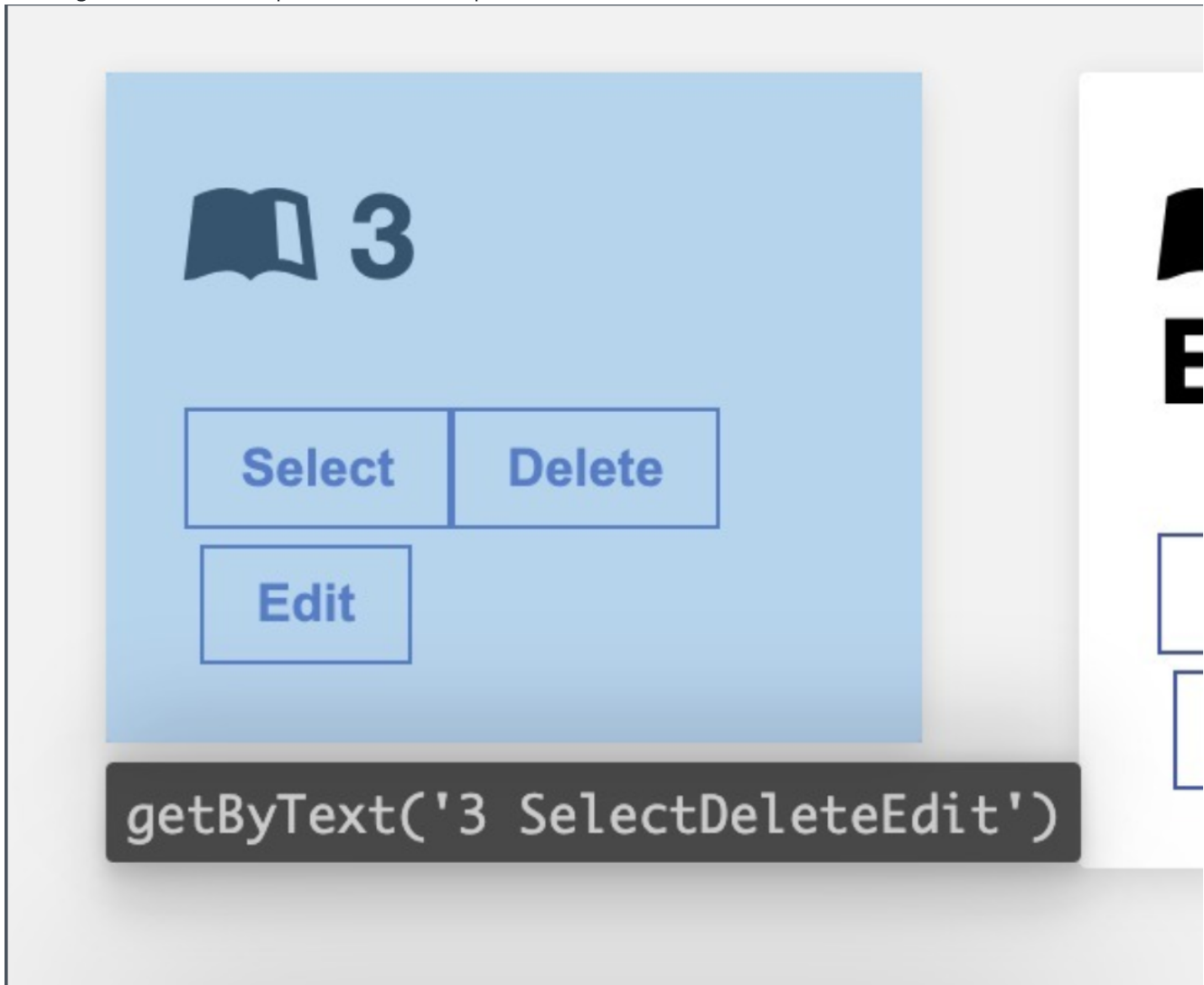
- image 1 : votre app et sur le coté le mode debug: vous pouvez exécuter le code étape par étape



- image 2: vous pouvez utiliser la fonction "pick locator" pour récupérer un élément de votre page que vous n'arrivez pas à accéder



- image 3: il suffit de cliquer sur l'élément que vous souhaitez sélectionner.



Les promesses : Promise

Quand vous faites :

```
const pageTitle = await page.getByRole('heading', { name: 'AGreatHeadingNameYouDontHave' });
```

La récupération du `pageTitle` n'est pas instantanée, on doit attendre la réponse (le temps que la page s'affiche ou que l'élément apparaisse etc..). On doit donc attendre, ça nous rappelle une notion qu'on a bien vu qui sont les observables ! Ici, une promesse c'est différent des observables, c'est one-shot : on attend et on reçoit la réponse (échec ou succès) et la promesse terminée. Il n'y a pas de notions de souscriptions comme pour les observables où un subscribe peut-être appelé plusieurs fois.

Pensez à vos `await` lorsque vous utilisez les fonctions de playwright car ces fonctions retourne des `Promise`. `await` permet d'attendre la réponse de la `Promise` avant de continuer l'exécution de la fonction, avant de passer à la ligne d'après. Dernière chose, si vous faites un `await` dans une fonction, votre fonction doit être flaggée comme `async` pour que Javascript comprenne comment traiter son contenu. Exemple avec cette

fonction dans la correction: <https://github.com/NablaT/ps6-correction-td1-td2-v2/blob/master/frontend/src/app/quizzes/quiz/quiz.fixture.ts#L12>

```
async getContentTitleQuiz(index) {  
  const title = await this.getTitleQuiz(index);  
  return title.textContent();  
}
```

✉ [Contacter l'assistance du site](#) ↗

Connecté sous le nom « [duong Thi Thanh Tu](#) » ([Déconnexion](#))

[Résumé de conservation de données](#)

[Obtenir l'app mobile](#)

Fourni par [Moodle](#)