

# Développement front

MMI 3 – TP#5 S5

Danielo **JEAN-LOUIS**

## Limites – Jest / Test unitaire

- Nécessite une maintenance du développeur
  - Màj du code = màj des tests unitaires
- Ne permet pas de tout tester / vérifier
  - Le test manuel ne doit pas être remplacé par les tests unitaires

## Limites – Jest / Test unitaire

- Peut prendre beaucoup, beaucoup de temps à exécuter
  - Nécessite parfois un ordinateur puissant
- Ne fonctionne pas avec l'entièreté d'un projet
  - S'arrête là où les tests d'intégration commencent

# Nébuleuse de tests automatisés

- Tests unitaires
  - Testent une unité de code
- Test d'intégration
  - Testent le fonctionnement orchestré de portions de code → ex : un formulaire entier
- **Tests end-to-end**
  - Simulent les actions utilisateurs. Testent un parcours entier

# Test end-to-end

- Souvent abrégé en e2e test
- Simule le comportement d'un utilisateur pour une action spécifique avec toutes les étapes
  - Ex : de la connexion à l'achat d'un produit
- Plus complet qu'un test unitaire ou test d'intégration

Source(s) :

- <https://www.artofunittesting.com/definition-of-a-unit-test>

# Test end-to-end

- Possibilité infinie → Priorisation à établir
  - Définition de scénarios/plans de tests
- Plus récent que les tests unitaires
- Existe dans quasiment tous les langages
- Géré aussi bien par les développeurs que la QA

# Test – Cycle de vie

- Initialisation (setUp) : mise à en place de l'env
- Exécution : Exécution des commandes
- Vérification : Teste des assertions
- Désactivation (tearDown) : nettoyage de l'env



## Point technique – Rôle de QA

- Responsable de la qualité d'un produit
  - QA = Quality Assurance
- Peut avoir une expérience en développement
  - Préférable pour avoir un meilleur salaire
- Connaît le produit et ses points critiques
- Très recherché par le monde du travail pour un profil de développement

# Test unitaire vs Test e2e

Test unitaire	Test e2e
Gère une petite unité de code	Gère une partie d'une application
Utilise des API simulées	Utilise de réelles API*
S'exécute dans la console	S'exécute dans un vrai navigateur/logiciel

\*Pensez à avoir un environnement de test pour éviter d'afficher de fausses données aux clients finaux

# Playwright

- Outil de e2e test permettant de contrôler un navigateur
- Développé et maintenu par Microsoft
- Fonctionne avec n'importe quelle typologie de projet web (vanilla js, react, vue...)
- Gratuit et Open Source
- Populaire

Source(s) :

- <https://playwright.dev/>

# Playwright

- Installation avec npm
  - `npm init playwright@latest`
- Gère les langages Python, C#, Java, Typescript et Javascript
- Documentation détaillée (en anglais)
- Réutilise certaines fonctions et assertions vues avec Jest
  - `test()`, `toBeEmpty()`, `toHaveValue()`...

Source(s) :

- <https://playwright.dev/>

# Playwright

- Permet de gérer les sites web, les extensions navigateur et les API
- Peut fonctionner en mode headless
  - Fonctionnement sans interface utilisateur
- Gestion de navigateurs multiples
- Extension pour VS Code disponible (voir sources)
- Tests exécutés en parallèle → Plus rapide

## Source(s) :

- <https://playwright.dev/>
- <https://marketplace.visualstudio.com/items?itemName=ms-playwright.playwright>

# Playwright

- Gère quasiment toutes les actions que vous pouvez faire vous-même en tant qu'utilisateur :
  - Click
  - Survol
  - Entrée de données dans champ de texte
  - Upload de média
  - Accéder à un site (important pour commencer)
  - ...

Source(s) :

- <https://playwright.dev/>

# Playwright - Exemple

```
test.spec.js

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

test('basic test', async ({ page }) => {
  await page.goto('http://mon.url');
  await expect(page).toHaveTitle(/ma page/i)
});
```

Exemple de test

Source(s) :

- <https://playwright.dev/>

# Playwright - Exemple

Nom du test

```
test.spec.js

test("has title", async ({ page }) => {
  // [...]
});
```

Fixtures isolées. Représente une sandbox de plusieurs classes Playwright : page, browser, context...

Source(s) :

- <https://playwright.dev/>
- <https://playwright.dev/docs/api/class-fixtures>



# Pratiquons ! - Tests end-to-end (Partie 1)

Pré-requis :

- Avoir la ressource `ressources/playwright`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-5%2Fressources>

# Locators

- Permettent de récupérer un élément de page pour interagir avec ou le tester



```
// [...]  
page.getByTestId('cancel-button');
```

On récupère toutes les balises ayant le data-attribut “data-testid=’cancel-button’”

Source(s) :

- <https://playwright.dev/docs/locators#introduction>

# Locators

- Possibilité d'enchaîner les instructions suite à un locator

```
// [...]  
await page.getByTestId('cancel-button').first().click();
```

On récupère le premier élément trouvé puis on clique dessus

Note : Les actions et assertions sont asynchrones, le mot-clé “await” est **indispensable**

Source(s) :

- <https://playwright.dev/docs/locators#introduction>

# Locators

- Possibilité d'utiliser un sélecteur CSS pour récupérer un élément de la page

```
// [...]
```

```
page.locator('.ma-classe');
```

```
page.locator('[data-mon-attribut]');
```

```
page.locator('#mon-id');
```

Source(s) :

- <https://playwright.dev/docs/locators#introduction>

# Locators

- Récupèrent **tous** les éléments sur la page
  - Utiliser avec `.first()` / `.last()` si on veut récupérer le premier / dernier élément trouvé
  - Utiliser avec `.nth(nb)` si on veut récupérer le *énième* élément (on compte à partir de 0)
- Utilise la même syntaxe qu'en javascript ou CSS pour sélectionner les éléments (sélecteur `.locator()`)

## Source(s) :

- <https://playwright.dev/docs/api/class-locator#locator-first>
- <https://playwright.dev/docs/api/class-locator#locator-last>
- <https://playwright.dev/docs/api/class-locator#locator-nth>

# Locator - click()

- Effectue une action de clic sur un élément
- **Ne fonctionne que sur un élément unique**
  - `page.locator("selecteur").click()` → lève une **erreur s'il y a plusieurs éléments retournés**
- Ne fonctionne pas (par défaut) sur un élément non-visible
  - `.click({force : true})` → contournement

Source(s) :

- <https://playwright.dev/docs/api/class-locator#locator-click>

# Locator - click()

- Possibilité de maintenir une touche
  - `.click({ modifiers: ['Shift'] })` → Effectue un click + maj

Source(s) :

- <https://playwright.dev/docs/api/class-locator#locator-click>

# Assertion - toHaveURL()

- Assertion permettant de vérifier l'URL courante
- Accepte une chaîne de caractères ou une RegEx comme paramètre
- S'applique sur la classe Page

Source(s) :

- <https://playwright.dev/docs/next/api/class-pageassertions#page-assertions-to-have-url>



# Assertion - toHaveURL()

```
...  
  
// [...]  
await expect(page).toHaveURL('mon.url')
```

On vérifie que l'URL courante est égale à "mon.url"

Source(s) :

- <https://playwright.dev/docs/next/api/class-pageassertions#page-assertions-to-have-url>

# Assertion - expect()

- Crée une assertion
  - Condition à valider
  - Syntaxe basée sur Jest (voir sources)
- Plusieurs types d'assertions possibles :
  - Locator, pages, api, generic

Source(s) :

- <https://playwright.dev/docs/test-assertions>

# Assertion - expect()

- Tâche asynchrone
- Arrête le test si l'instruction échoue
  - Utilisation de “expect.soft()” pour éviter l'arrêt du test

## Source(s) :

- <https://playwright.dev/docs/test-assertions>
- <https://playwright.dev/docs/test-assertions#soft-assertions>

# Assertion - expect()


- Validation possibles :
  - Vrai ou Faux - `.toBeTruthy()` / `.toBeFalsy()`
  - Longueur - `.toHaveLength(42)`
  - Texte - `.toHaveText("mon texte")`
  - Capture d'écran : `.toHaveScreenshot()`
  - ...

Source(s) :

- <https://playwright.dev/docs/test-assertions>

# Assertion - négation

- Propriété “not”
- Permet de tester l'inverse d'une assertion

A code editor window with a purple border and a dark purple background. The title bar shows three small circles on the left and the filename 'negating-matchers.spec.js' on the right. The code inside is 'await expect(locator).not.toContainText('MMI');' with syntax highlighting: 'await' is pink, 'expect' is green, 'locator' is white, '.not' is blue, 'toContainText' is green, and 'MMI' is yellow.

```
negating-matchers.spec.js

await expect(locator).not.toContainText('MMI');
```

On teste si notre locator ne possède pas le texte “MMI”

Source(s) :

- <https://playwright.dev/docs/test-assertions>
- <https://playwright.dev/docs/test-assertions#negating-matchers>

# Pratiquons ! - Tests end-to-end (Partie 2)

Pré-requis :

- Avoir la ressource `ressources/playwright`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-5%2Fressources>

# Conventions et bonnes pratiques

- Extension de fichiers .spec.js ou .test.js
- **Le nom d'un test doit être explicite**
  - Commencer l'intitulé d'un test par "should"
- Grouper les tests par catégorie / page
  - Fonction "test.describe()" (comme jest)
- **Tester uniquement son code**
  - Le code tiers ne doit pas être testé

Source(s) :

- <https://playwright.dev/docs/best-practices>

# Conventions et bonnes pratiques

- Définir les tests les plus pertinents
  - Vous n'avez pas forcément les moyens ni les besoins de tout tester
- Éviter d'utiliser des classes ou id comme sélecteur
  - Ils peuvent changer et casser vos tests



# Formulaires

- Possibilité de manipuler les éléments de formulaire
  - Input
  - Radio button
  - Select
  - ...

# Formulaires

input.spec.js

```
// [...]
```

```
await page.getByRole('textbox', { name: 'Nom' }).fill('hello');
```

On cherche l'input avec le label "Nom" et on écrit le texte "hello" dedans  
(On peut remplacer `.getByRole()` par `.locator()`)

Source(s) :

- <https://playwright.dev/docs/api/class-locator#locator-get-by-role>
- <https://playwright.dev/docs/api/class-locator#locator-fill>

## Action - fill()

- Permet d'écrire du texte dans un élément éligible
- S'applique sur un élément du DOM
- Possibilité d'écrire comme un être humain
  - Paramètre “delay”
- Ne pas utiliser “type()”, méthode désuète

Source(s) :

- <https://playwright.dev/docs/api/class-locator#locator-fill>

# Pratiquons ! - Tests end-to-end (Partie 3/4)

Pré-requis :

- Avoir la ressource `ressources/playwright`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-5%2Fressources>

# Sélecteurs

- Évitez les sélecteurs mutables et polyvalents :
  - "class", "id", "balise"...
- Préférez les data-attributes
  - Avantages :
    - Mono-role
    - Permettent de passer des données
    - Sélecteurs “immutables”

# Point technique – Sélecteur de texte

- Fausse bonne idée
  - Les textes sont amenés à changer surtout s'ils sont dynamiques (articles, commentaires...)
- Sélection par texte **si et seulement si** le texte a son importance
- Possibilité d'avoir plus de souplesse avec les RegEx

# Sélecteurs - data-testid

- Convention issue de l'outil testing-library
  - Autre outil de tests automatisés
- Nom de data-attr explicite pour désigner un élément qui va être manipulé par un outil de test

Pas obligatoire, mais je vous conseille de l'utiliser ou de définir une convention

Source(s) :

- <https://testing-library.com/>

# Sélecteurs - data-testid



```
<a href="" data-testid="link-account">mon lien</a>
```

Utilisation du data-attribut "data-testid" pour cibler l'élément dans l'outil de test



# Sélecteurs - data-testid



data-testid.spec.js

```
// [...]
```

```
await page.getByTestId('link-account').click();
```

Playwright propose une méthode dédiée pour cibler par data-testid.  
Il est possible de cibler ces éléments via la méthode locator()

# Sélecteurs - data-attr

- A préférer pour sélectionner un élément dans le javascript
- Prévient les effets de bord en cas de changement de valeur pour les attributs : id, class...
- Une même balise peut avoir plusieurs data-attributes **uniques**

Source(s) :

- [https://developer.mozilla.org/fr/docs/Learn/HTML/Howto/Use\\_data\\_attributes](https://developer.mozilla.org/fr/docs/Learn/HTML/Howto/Use_data_attributes)

# Pratiquons ! - Tests end-to-end (Partie 5)

Pré-requis :

- Avoir la ressource `ressources/playwright`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-5%2Fressources>

# Upload de média

- Playwright utilise un réel fichier
- Utilisation de la méthode `.setInputFiles()`

Source(s) :

- <https://playwright.dev/docs/input#upload-files>

# Upload de média



upload.spec.js

```
// [...]  
await page.getByLabel('Photo').setInputFiles(  
  './tests/test.tmp.gif'  
);
```

On choisit le fichier "test.tmp.gif" pour le champ ayant le label "Photo".  
Le chemin est relatif à la racine du projet pas au fichier de test.

Source(s) :

- <https://playwright.dev/docs/input#upload-files>

# Pratiquons ! - Tests end-to-end (Partie 6)

Pré-requis :

- Avoir la ressource `ressources/playwright`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-5%2Fressources>

# Tests asynchrones

- Playwright réalise de **réels** appels d'API
  - Possibilité de surcharger les réponses (usage de mocks)
- Préférer l'attente de réponse pour éviter une erreur si l'API est trop lente
- Possibilité de mocker des HTTP Archive

Source(s) :

- <https://playwright.dev/docs/mock>

# Mock

- Doit être effectué avant l'appel de l'API à la page
  - Sinon, ça ne fonctionnera pas
- Gestion de tout type de requêtes HTTP

Source(s) :

- <https://playwright.dev/docs/mock>



# Mock

- Permet :
  - D'ajouter des données à la réelle requête
  - Modifier les headers de la requête (sans toucher son corps)

Source(s) :

- <https://playwright.dev/docs/mock>

# Mock API



```
mock.spec.js

await page.route("/api", async (route) => {
  const json = [{ formation: "MMI", id: 42 }];
  await route.fulfill({ json });
});
```

Toute requête d'API contenant "/api" retourneront autre chose que la vraie API

Source(s) :

- <https://playwright.dev/docs/mock#modify-api-responses>

# Mock API

```
mock.spec.js

await page.route("/api", async (route) => {
  if(route.request().method().toLowerCase() === "get") {
    const json = [{ name: "Pikachu", id: 25 }];
    await route.fulfill({ json });
  } else {
    await route.fallback();
  }
});
```

Il est possible de vérifier le type de requête et de continuer normalement si la condition n'est pas remplie

Source(s) :

- <https://playwright.dev/docs/mock#modify-api-responses>
- <https://playwright.dev/docs/api/class-route>

# Mock API

mock.spec.js

```
await route.fulfill({  
  status: 404,  
  contentType: 'application/json',  
  body: JSON.stringify({ error: "Something went wrong" })  
});
```

Playwright permet également de changer le code de la réponse. Ici 404.

Source(s) :

- <https://playwright.dev/docs/mock#modify-api-responses>
- <https://playwright.dev/docs/api/class-route>

# Mock API

mock.spec.js

```
await page.waitForResponse("/api");
```

Si votre API prend du temps, il est possible d'attendre son retour.  
Il est possible d'attendre plus longtemps en passant un paramètre.

Source(s) :

- <https://playwright.dev/docs/api/class-page#page-wait-for-response>

# Pratiquons ! - Tests end-to-end (Partie 7)

Pré-requis :

- Avoir la ressource `ressources/playwright`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-5%2Fressources>

# Taille de fenêtre

- Utilisation de la méthode `setViewportSize()`
- Permet de changer la taille de la fenêtre
  - Idéal pour tester le responsive

Source(s) :

- <https://playwright.dev/docs/api/class-page#page-set-viewport-size>

# Capture d'écran - screenshot()

- Utilisation de la méthode screenshot()
- Prend une capture d'écran complète / partielle
  - Possibilité d'en prendre plusieurs au sein du même test
  - Possibilité de rogner la capture

Source(s) :

- <https://playwright.dev/docs/api/class-page#page-screenshot>



# Capture d'écran - screenshot()

screenshot.spec.js

```
await page.screenshot({ path: "screenshot.png" });
```

On prend une capture nommée "screenshot.png"

Note : le chemin a pour base la racine du projet

Note 2 : Ne commitez pas les captures d'écran

Source(s) :

- <https://playwright.dev/docs/api/class-page#page-screenshot>

# Invocation de fonction

- Utilisation de la méthode `evaluate()`
- Appelle une fonction sur l'élément précédemment récupéré
  - Modification du CSS, par exemple

Source(s) :

- <https://playwright.dev/docs/evaluating>

# Invocation de fonction



```
await page.locator("input").evaluate((item) => {  
    item.classList.add("my-class");  
});
```

On ajoute une classe sur l'élément qui a été récupéré

Source(s) :

- <https://playwright.dev/docs/evaluating>

# Pratiquons ! - Tests end-to-end (Partie 8)

Pré-requis :

- Avoir la ressource `ressources/playwright`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-5%2Fressources>

# playwright.config.js

- Fichier de configuration de Playwright
  - Crée lors de l'initialisation du projet
- Permet de configurer :
  - URL d'entrée (le domaine d'où commence tous les tests)
  - Taille par défaut du navigateur
  - ...

Source(s) :

- <https://playwright.dev/docs/test-configuration>
- <https://playwright.dev/docs/test-use-options>

# Ouverture de nouvel onglet

- Playwright ouvre un nouvel onglet en cas de `target="_blank"` mais...
  - ...L'objet Page ne change pas d'onglet
- Nécessite de prévenir Playwright de l'ouverture d'un nouvel onglet

Source(s) :

- <https://playwright.dev/docs/pages#handling-new-pages>

# Ouverture de nouvel onglet

```
target-blank.spec.js

test("redirection", async ({ page, context }) => {
  await page.goto("new-tab.com");
  await page.getByRole("link").first().click();
  const pagePromise = context.waitForEvent('page');
  // Stores the new page. Page instance
  const newPage = await pagePromise;
  // [...]
});
```

On transfère la page ouverte dans un nouvel onglet dans la variable "newPage"

Source(s) :

- <https://playwright.dev/docs/pages#handling-new-pages>

# Générer les tests

- Permet de générer le code des actions de tests
  - Et certaines assertions
  - Sélectionne le locator le plus pertinent et unique
- Utilisation de la commande
  - `npx playwright codegen` (ou “open”)

Source(s) :

- <https://playwright.dev/docs/codegen-intro>



# Générer les tests

- Utile quand on débute les tests
- Compatible avec VS Code et l'extension Playwright pour VS Code (voir source)
  - Génère des fichiers Typescript

## Source(s) :

- <https://playwright.dev/docs/codegen-intro>
- <https://marketplace.visualstudio.com/items?itemName=ms-playwright.playwright>

# Pratiquons ! - Tests end-to-end (Partie 9)

Pré-requis :

- Avoir la ressource `ressources/playwright`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-5%2Fressources>

# Taguer ses tests

- Système d'annotation
- Permet de filtrer les tests à lancer
- Un test peut avoir plusieurs tags
- Un tag peut s'appliquer à un groupe de tests

Source(s) :

- <https://playwright.dev/docs/test-annotations#tag-tests>

# Taguer ses tests

```
tag.spec.js

test("Should have a tag",
  { tag: '@report' },
  async ({ page }) => {
    // [...]
  })
```

Notre test a le tag "@report", nous pourrions choisir d'exécuter uniquement ce tag

Source(s) :

- <https://playwright.dev/docs/test-annotations#tag-tests>

# Initialisation / Désactivation

- setUp / tearDown en anglais
- Permet d'effectuer une action avant/après chaque test
  - Limite la répétition de code
- Utilisation des méthodes test.beforeEach() et test.afterEach() avec Playwright

## Source(s) :

- <https://playwright.dev/docs/api/class-test#test-before-each>
- <https://playwright.dev/docs/api/class-test#test-after-each>

# Initialisation / Désactivation

setup.spec.js

```
test.beforeEach(async ({ page }) => {  
  await page.goto('https://my.start.url/');  
});
```

Cette instruction ouvrira notre url avant **chaque test présent dans le même fichier**, plus besoin de répéter l'instruction.

Source(s) :

- <https://playwright.dev/docs/api/class-test#test-before-each>
- <https://playwright.dev/docs/api/class-test#test-after-each>

# Rapport

- Permet d'avoir une vue d'ensemble des tests
- Contenu généré dans le dossier "playwright-report"
  - Préférer sa consultation depuis un serveur
    - Commande "npx playwright show-report"
- Actualisé après chaque test

Source(s) :

- <https://playwright.dev/docs/test-reporters>

# Rapport

- Plusieurs formats de données disponibles
- N'affiche que les résultats des derniers
- Automatiquement généré si les tests sont exécutés en mode headless et lèvent une erreur

Source(s) :

- <https://playwright.dev/docs/test-reporters>



# Concurrents (liste non exhaustive)

- Selenium (à éviter)
- Cypress
- Katalon (basé sur Selenium)
- Nightwatchjs
- Puppeteer

## Source(s) :

- <https://nightwatchjs.org/>
- <https://www.cypress.io/>
- <https://pptr.dev/>
- <https://www.selenium.dev/>

**Questions ?**

