

Développement front

MMI 3 – TP#2 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
- Géré aussi bien par les développeurs que la QA
- Plus récent que les tests unitaires
- Existe dans quasiment tous les langages

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

Point technique – U.T. 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

Cypress

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

Source(s) :

- <https://docs.cypress.io/guides/overview/why-cypress>

Cypress

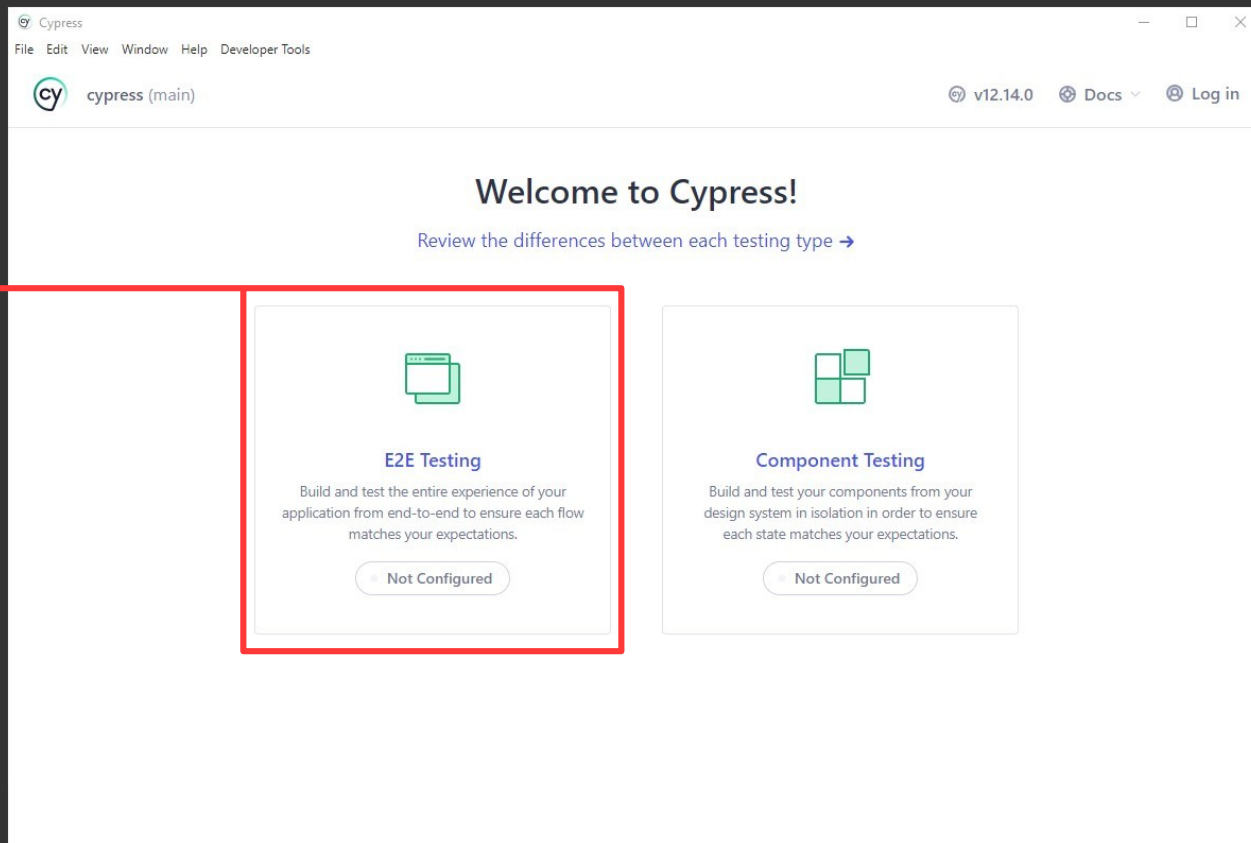
- Installation avec npm
 - `npm install -D cypress`
- Fonctionne avec le logiciel du même nom
 - Installé lors de l'installation de cypress via npm
- Utilise le langage de programmation javascript

Source(s) :

- <https://docs.cypress.io/guides/overview/why-cypress>

Cypress

Choisissez E2E Testing

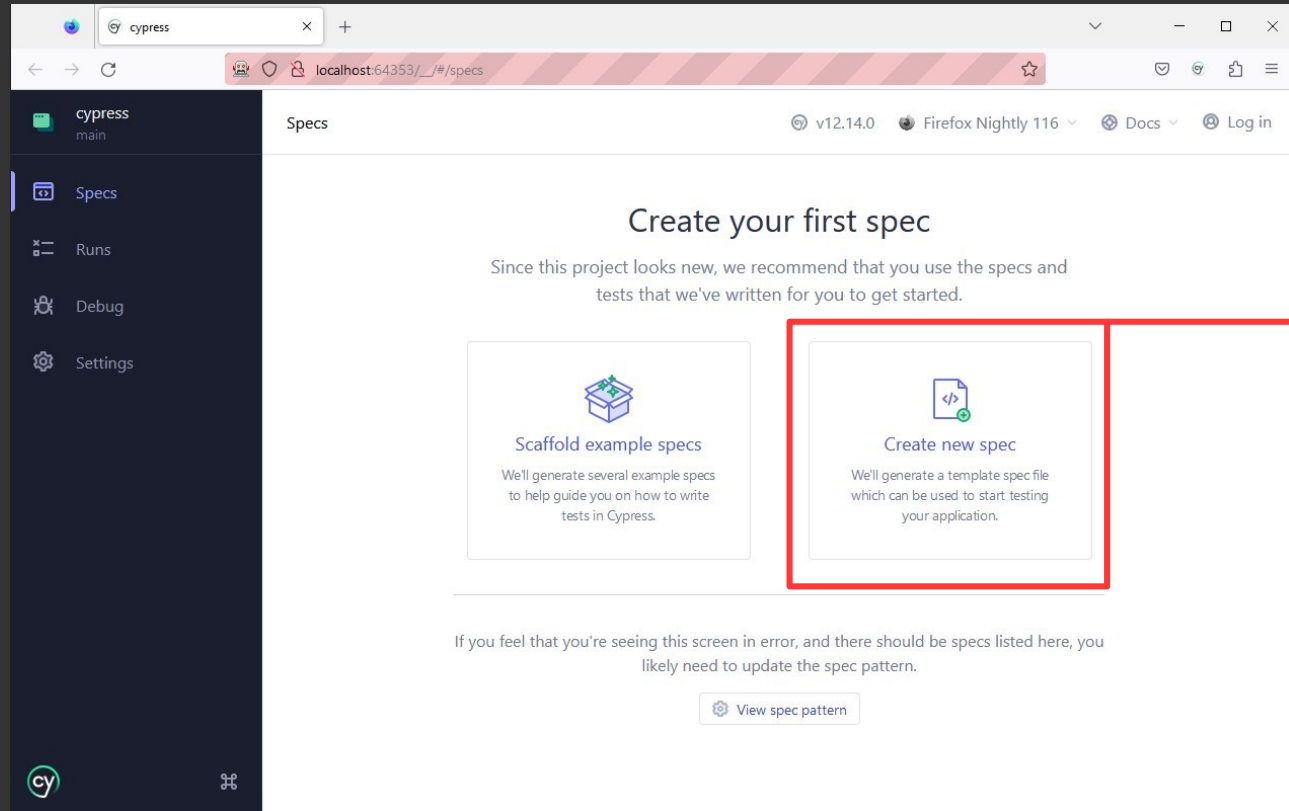


Ecran d'accueil de cypress (premier lancement)

Source(s) :

- <https://docs.cypress.io/guides/overview/why-cypress>

Cypress



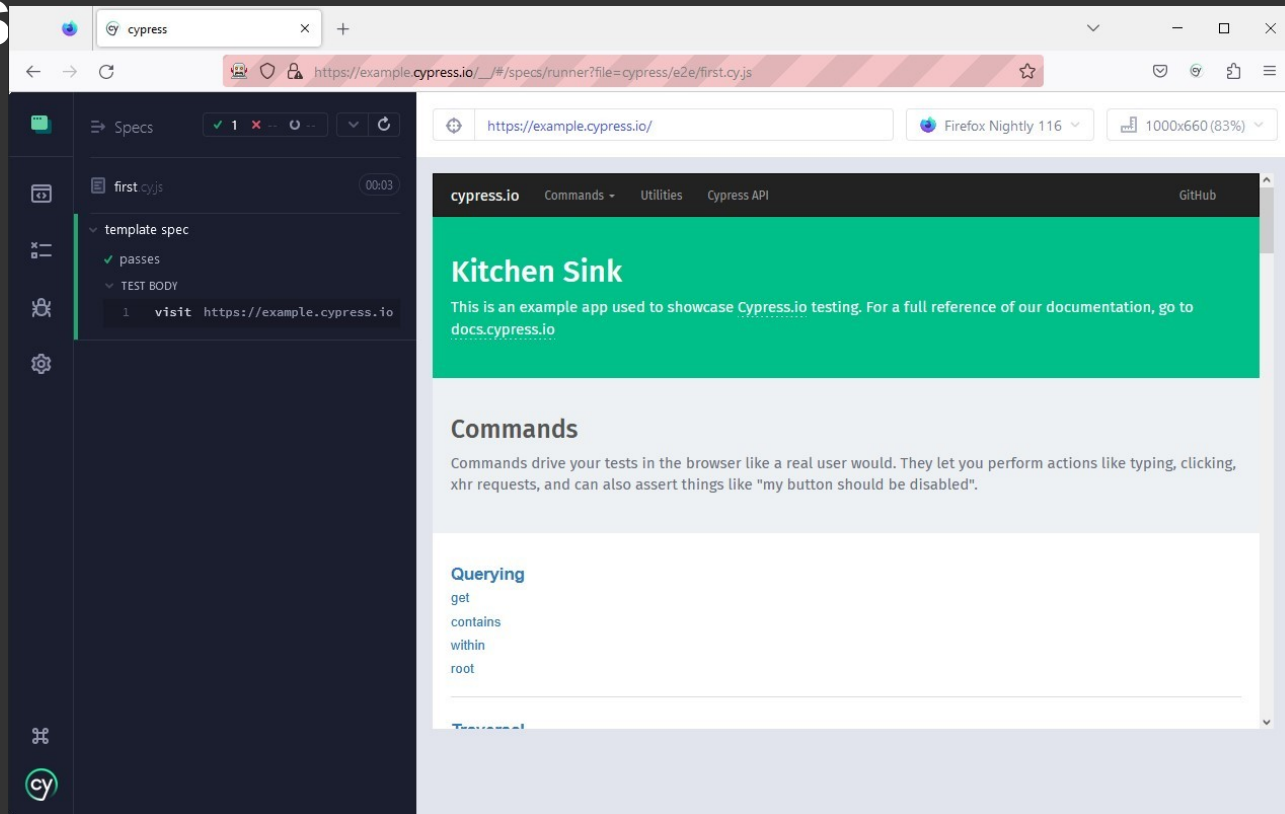
Créez votre premier test

Ecran de tests (ici aucun test n'a encore été écrit)

Source(s) :

- <https://docs.cypress.io/guides/overview/why-cypress>

Cypress

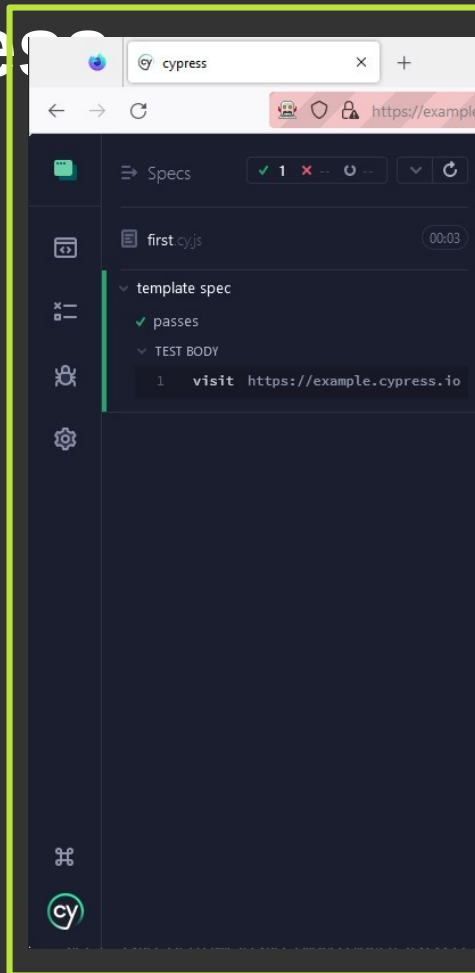


Écran d'exécution de tests

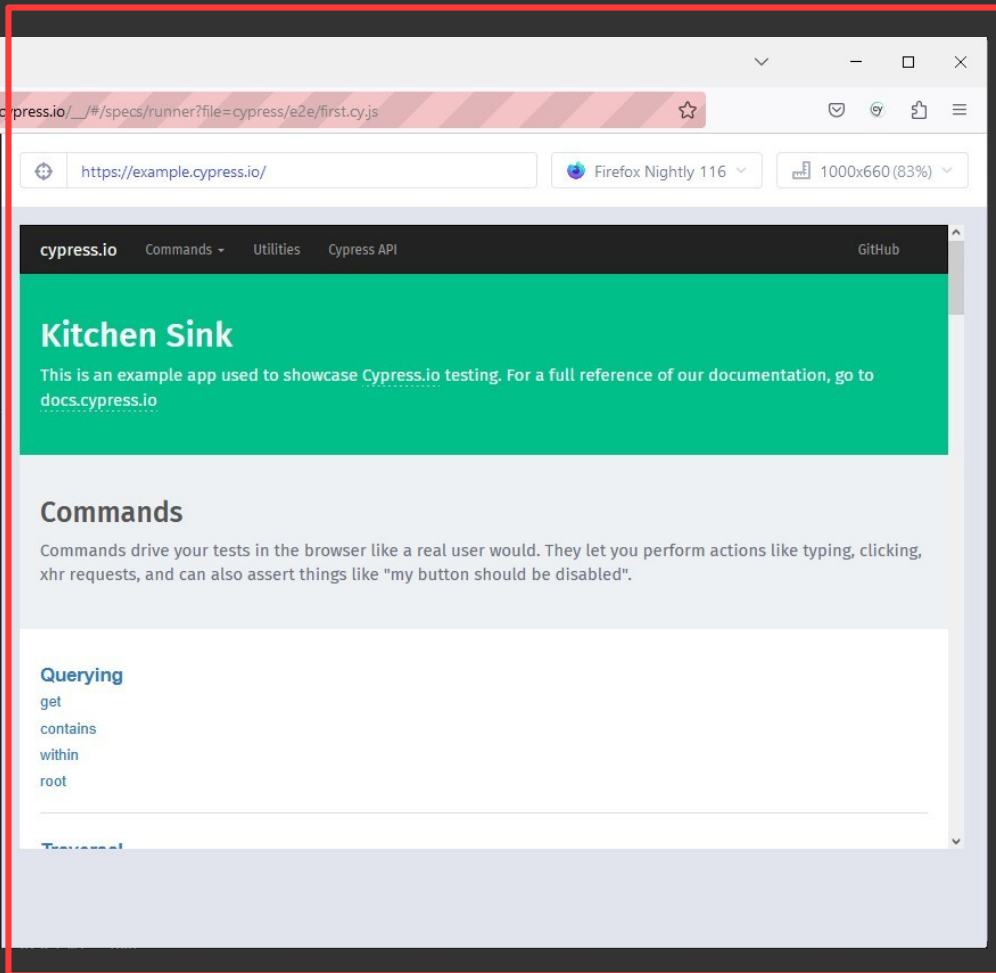
Source(s) :

- <https://docs.cypress.io/guides/overview/why-cypress>

Cypress



Liste de commandes



Application

Cypress

- Documentation détaillée (en anglais)
- Pointilleux dans sa structure de travail
 - Tous vos tests doivent être dans le dossier cypress/ (créé à la racine de votre projet) et avoir une extension en ".cy.js"
- Permet de remonter l'exécution des actions
 - Idéal pour le debug

Source(s) :

- <https://docs.cypress.io/guides/overview/why-cypress>

Cypress

- **Ne permet pas nativement de gérer plusieurs domaines dans le même test**
 - Un test = un site (par défaut)
- Extensible via des plugins
 - Possibilité de créer ses propres extensions
- Réutilise certaines fonctions vues avec Jest
 - `describe()`, `test()`

Source(s) :

- <https://docs.cypress.io/guides/overview/why-cypress>

Cypress

- 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://docs.cypress.io/guides/overview/why-cypress>

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

Pré-requis :

- Avoir la ressource ressources/cypress

A télécharger ici :

<https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-2/ressources>

Cypress

- Syntaxe haut niveau
 - Le nom des méthodes est proche d'une phrase



```
// [...]  
cy.get("a")
```


On récupère toutes les balises <a> trouvées sur la page

Source(s) :

- <https://docs.cypress.io/api/table-of-contents>

Cypress

- Possibilité d'enchaîner les instructions



```
// [...]  
cy.get("a").first().click()
```

On récupère la **seule** balise <a> trouvée dans la page dans la page puis on clique dessus

Source(s) :

- <https://docs.cypress.io/api/table-of-contents>

Cypress

- Vérification des résultats (méthode `.should()`)

```
cy.get("a").first().click()  
cy.location('href').should(  
  "eq",  
  "ma-page.html"  
);
```

Après avoir cliqué, on s'assure qu'on atterrit sur la bonne page

Source(s) :

- <https://docs.cypress.io/api/table-of-contents>

Cypress - get()

- Récupère **tous** les éléments sur la page
 - Utiliser avec `.first()` si on veut récupérer le premier élément trouvé
 - Utiliser avec `.eq(nb)` si on veut récupérer le *énième* élément
- Utilise la même syntaxe qu'en javascript ou CSS pour sélectionner les éléments

Source(s) :

- <https://docs.cypress.io/api/commands/get>
- <https://docs.cypress.io/api/commands/first>
- <https://docs.cypress.io/api/commands/eq>

Cypress - get()

```
// [...]  
cy.get("li")
```

← On récupère toutes les balises

On récupère la première balise →

```
// [...]  
cy.get("li").first()
```

Source(s) :

- <https://docs.cypress.io/api/commands/get>
- <https://docs.cypress.io/api/commands/first>
- <https://docs.cypress.io/api/commands/eq>

Cypress - click()

- Effectue une action de clic sur un élément
- **Ne fonctionne que sur un élément unique**
 - `cy.get("selecteur").click()` → **interdit**
- Ne fonctionne pas (par défaut) sur un élément non-visible
 - `.click({force : true})` → contournement

Source(s) :

- <https://docs.cypress.io/api/commands/click>

Cypress - location(param)

- Représente la page active
- Contient toutes les propriétés de l'objet `window.location` de javascript
 - Lien de la page → `.location("href")`
 - Paramètre d'URL → `.location("search")`
 - ...

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/window/location>
- <https://docs.cypress.io/api/commands/location>

Cypress - should()

- Crée une assertion
 - Une condition à valider
 - Syntaxe basée sur chai (voir sources)
- Validation possibles :
 - Vrai ou Faux - `.should('be.true')` / `.should('be.false')`
 - Longueur - `.should('have.lengthOf', 42)`
 - Visibilité - `.should("be.visible")`
 - ...

Source(s) :

- <https://docs.cypress.io/api/commands/should>
- <https://docs.cypress.io/guides/references/assertions#Chai>

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

Pré-requis :

- Avoir la ressource ressources/cypress

A télécharger ici :

<https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-2/ressources>

Cypress - get()

- Note : **dans certains cas**, `.get()` ne retourne qu'un seul élément au lieu d'un tableau. Pour éviter les erreurs, combinez toujours avec `.first()`, `.eq()` ou `.last()`

Source(s) :

- <https://docs.cypress.io/api/commands/get>
- <https://docs.cypress.io/api/commands/first>
- <https://docs.cypress.io/api/commands/eq>

Conventions et bonnes pratiques

- Extension de fichiers `.cy.js`
- **Le nom d'un test doit être explicite**
- Grouper les tests par catégorie / page
 - Fonction “describe()” (comme jest)
- Définir les valeurs partagées grâce au fichier `cypress.config.js`

Source(s) :

- <https://docs.cypress.io/guides/references/best-practices>

Conventions et bonnes pratiques

- Mettre vos tests dans le dossier cypress/e2e
 - Vous pouvez créer des sous-dossiers dedans
- Définir les tests les plus pertinents
 - Vous n'avez pas forcément les moyens ni les besoins de tout tester

Cypress – Formulaires

- Permet de remplir un formulaire
 - Input
 - Radio button
 - Select
 - ...

Source(s) :

- <https://docs.cypress.io/api/commands/should>
- <https://docs.cypress.io/guides/references/assertions#Chai>

Cypress – Formulaires

```
...  
// [...]  
cy.get("#prenom").first().type("hello")
```

On cherche l'input avec l'id "prenom" et on écrit le texte "hello" dedans

Source(s) :

- <https://docs.cypress.io/api/commands/should>
- <https://docs.cypress.io/guides/references/assertions#Chai>

Cypress - type()

- Permet d'écrire du texte dans un élément
 - Un champ de préférence
- Permet d'enregistrer des actions clavier (voir doc)
- Ne jamais ajouter une action après la commande

Source(s) :

- https://docs.cypress.io/api/commands/type#docusaurus_skipToContent_fallback

Cypress - type()



```
// [...]  
cy.get("#prenom")  
  .first()  
  .type("hello")  
  .should('have.class', 'active')
```



Correct

Il n'y a pas de commande
après .type() →



Interdit

← Il y a une commande
après .type()



```
// [...]  
cy.get("#prenom")  
  .first()  
  .type("hello")  
cy.get("#prenom").should('have.class', 'active')
```

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

Pré-requis :

- Avoir la ressource ressources/cypress

A télécharger ici :

<https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-2/ressources>

Sélecteurs

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

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 à son importance

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-attribute "data-testid" pour cibler l'élément dans l'outil de test

Sélecteurs - data-testid

```
cy.get("[data-testid='link-account']").first().click()
```

Ciblage de l'élément dans cypress

Sélecteurs - data-attr

- A préférer pour sélectionner un élément dans le javascript
- Préviend 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

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

Pré-requis :

- Avoir la ressource ressources/cypress

A télécharger ici :

<https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-2/ressources>

Upload de média

- Utilisation de mock
 - Doivent être contenus dans le dossier `cypress/fixtures/`
- Cypress réalise réellement l'upload
- Utilisation de la méthode `.selectFile()`

Source(s) :

- https://docs.cypress.io/api/commands/selectfile#docusaurus_skipToContent_fallback

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

Pré-requis :

- Avoir la ressource ressources/cypress

A télécharger ici :

<https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-2/ressources>

Tests asynchrones

- Cypress réalise de réels appels d'API
 - Possibilité de surcharger les réponses (usage de fixtures)
- Utilisation de la méthode `.wait()` pour attendre le retour d'API
 - Sinon Cypress échouera si la ligne suivante dépend de la réponse d'API

Source(s) :

- <https://docs.cypress.io/api/commands/wait>
- <https://docs.cypress.io/api/commands/intercept>
- <https://learn.cypress.io/advanced-cypress-concepts/intercepting-network-requests>

Tests asynchrones

- Combinaison de `.wait()` avec `.intercept()` pour savoir quels retours Cypress doit attendre
- La méthode `.wait()` ne doit pas servir à attendre arbitrairement une tâche
- Mettez toujours vos `.intercept()` avant le premier `.visit()`

Source(s) :

- <https://docs.cypress.io/api/commands/wait>
- <https://docs.cypress.io/api/commands/intercept>
- <https://learn.cypress.io/advanced-cypress-concepts/intercepting-network-requests>

Tests asynchrones

```
cy.intercept(  
  'GET',  
  'https://calendrier.api.gouv.fr/jours-feries/*'  
)  
.as('getNationalHolidays')
```

On indique à Cypress qu'on souhaite attendre tous les appels de type GET vers l'url définie. La méthode `.as()` permet de définir un alias (au nom arbitraire) qui sera utilisé dans la méthode `.wait()`

Tests asynchrones

```
cy.intercept("/jours-feries/*")  
    .as("getNationalHolidays");  
// [...]  
cy.wait("@getNationalHolidays", {  
    timeout: 15000  
});
```

On demande à Cypress d'attendre le retour de toute requête contenant `"/jours-feries/*"` avant de passer à la ligne suivante. Le paramètre `"timeout"` (facultatif) permet d'attendre plus longtemps si le serveur prend du temps.

Point technique – Cache serveur

- Code serveur 304
 - Indique que le résultat de la requête provient du navigateur et non du serveur
- Economie de bande passante
- Considéré comme étant une non-requête pour Cypress. Voir partie 7 du TP

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

Pré-requis :

- Avoir la ressource ressources/cypress

A télécharger ici :

<https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-2/ressources>

Tests asynchrones - intercept()

- Retourne une promesse
 - Possibilité de récupérer la réponse du serveur et la réutiliser dans le test

```
cy.intercept("/jours-feries/*")  
  .as("getNationalHolidays");  
// [...]  
cy.wait("@getNationalHolidays").then(res => {  
  console.log(res);  
  // [...]  
});
```

← On récupère la réponse de l'API dans la variable "res".

Note : par défaut, vous ne pouvez pas remplacer cette promesse par async/await.

cypress.config.js

- Fichier de configuration de Cypress
 - Crée lors de l'initialisation du projet
- Permet de configurer :
 - URL d'entrée (le domaine d'où commence tous les tests)
 - Variables partagées
 - ...
 - Surcharger le comportement par défaut

Source(s) :

- <https://docs.cypress.io/guides/references/configuration>

Cypress – En vrac

- Possibilité de faire des captures d'écran page entière ou partie spécifique
 - `cy.screenshot()`
- Création de commandes personnalisées pour réunir et partager des commandes répétées
- Changement de taille de fenêtre
 - `cy.viewport()`
- ...

Source(s) :

- <https://docs.cypress.io/api/commands/screenshot>
- https://docs.cypress.io/api/cypress-api/custom-commands#docusaurus_skipToContent_fallback
- <https://example.cypress.io/commands/viewport>

Cypress - within()

- Permet de scoper les sélecteurs
 - Chercher dans un sous-élément de la page

```
cy.get('[data-testid="list-but"]')
  .within(() => {
    // Dans within Cypress va chercher uniquement
    // dans la balise ayant le data-attribute test-id
    // avec la valeur "list-but"
    return cy.get('[data-testid="but-mmi"]').should('have.class', 'active')
  })
```

Source(s) :

- <https://docs.cypress.io/api/commands/within>

Questions ?

