

Les tests End 2 End

Table des matières

I. Découverte des tests de bout en bout (en anglais End-To-End - E2E)	3
II. Exercice : Quiz	6
III. Mise en place d'un test de bout en bout avec cypress.js	7
IV. Exercice : Quiz	11
V. Essentiel	12
VI. Auto-évaluation	12
A. Exercice	12
B. Test	12
Solutions des exercices	13

I. Découverte des tests de bout en bout (en anglais End-To-End - E2E)

Durée : 1 h 30

Prérequis : Visual studio code, avoir des connaissances en javascript, installer Node.js et npm, comprendre l'utilisation du terminal

Environnement de travail : Un ordinateur

Contexte

Aujourd'hui, les sites Internet deviennent de plus en plus complexes et souvent interconnectés avec de multiples sous-systèmes, services réseau et backend. Si l'un d'entre eux échoue, toute une application peut se bloquer. C'est pourquoi un test de bout en bout doit être effectué à la fin de tous les autres processus de test.

Définition Test de bout en bout

Le test de bout en bout est le processus de test d'une application du début à la fin telle qu'elle serait utilisée par ses utilisateurs. Dans un contexte d'application web, cela passe par démarrer le navigateur, accéder à l'URL, utiliser l'application comme prévu et vérifier son comportement. Dans le contexte d'une application desktop, cela consiste à télécharger puis démarrer l'application, l'utiliser et vérifier également son comportement.

Enfin si vous testez une API, l'idée est de passer des appels comme le feraient les clients.

Quel que soit votre site, application ou logiciel, l'idée est de tester son fonctionnement final avec toutes ses dépendances : base de données, métriques, services externes, etc.

Ainsi 3 couches d'une application sont impliquées dans un test de bout en bout :

- La couche de présentation (ou UI)
- La couche des fonctionnalités de notre application (couche logique, métier)
- La couche de persistance

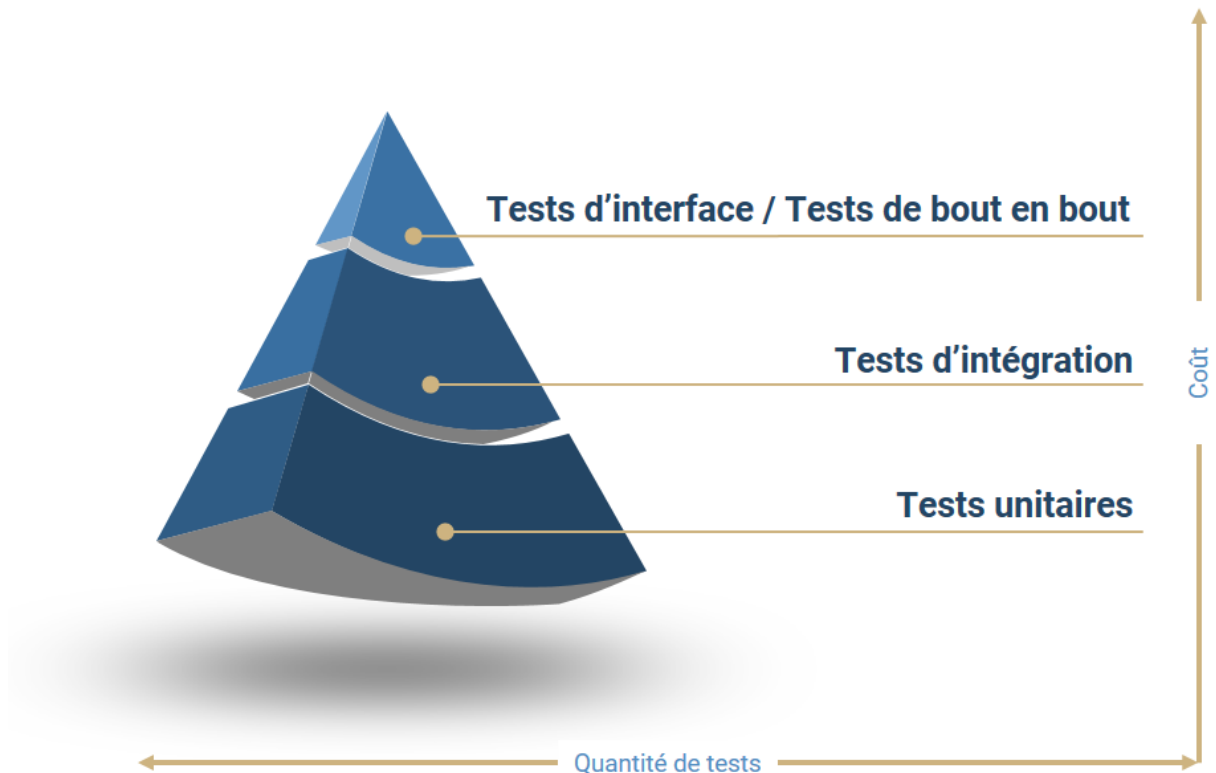
Le test de bout en bout

Bien que nombreux développeurs soient plus habitués aux tests unitaires, aux tests fonctionnels et aux tests d'intégrations, ces derniers permettent de valider seulement le code en cours d'écriture et non pas l'application finale.

C'est à cette étape que le test de bout en bout intervient. Il facilite l'écriture de scénarios utilisateurs réels de manière simple et universelle. Bien évidemment, il est tout à fait possible d'élaborer des scénarios d'utilisations complexes de l'application. Une fois ses scénarios automatisés, le gain de temps pour les équipes de tests est énorme et les tâches les plus répétitives se retrouvent fortement diminuées.

Les avantages des tests de bout en bout automatisés :

- Vérifier le bon fonctionnement de chaque couche de l'application
- Valider les modules en arrière-plan comme une base de données
- Limiter les risques de défaillance ou de panne dans le futur
- Diminuer les efforts des équipes de test sur des tâches manuelles répétitives
- Gagner du temps et réduire les coûts



Rappel

Les tests E2E sont la dernière étape du processus de testing. Ils consistent à se mettre à la place de l'utilisateur et à couvrir un maximum de cas d'utilisation d'une application ou d'un logiciel. Le processus de test de bout en bout est automatisable.

Procédure de base pour la mise en place de test de bout en bout :

- Se mettre à la place du client,
- Comprendre les *personas* des utilisateurs (un *personas* représente un certains types d'utilisateurs de l'application, une application est le plus souvent utilisés par différents *personas* possibles, par exemple chez Studi : *personas* étudiant, *personas* formateur),
- Viser une couverture de test de 100 %,
- Utiliser un bon outil de gestion des cas de test,
- Évitez qu'un cas de test dépendent d'un autre cas de test,
- Utilisez l'automatisation.

Les choses à éviter :

- Mon test exécute une seule condition de test particulière,
- Il ne couvre qu'une petite partie d'une fonctionnalité,
- Il ne teste qu'un seul *personas*,
- Il ne se limite qu'au cahier des charges,
- Il ne met pas en place de système de classification des données remontées par les tests.

Exemple **Scénario site e-commerce**

Exemple de scénario site e-commerce : « User se rend sur le site e-commerce Z, recherche le produit W, l'ajoute à son panier et finalise sa commande en payant par CB ». Mais il peut aussi être bien plus complexe afin de chercher à couvrir un maximum de possibilités :

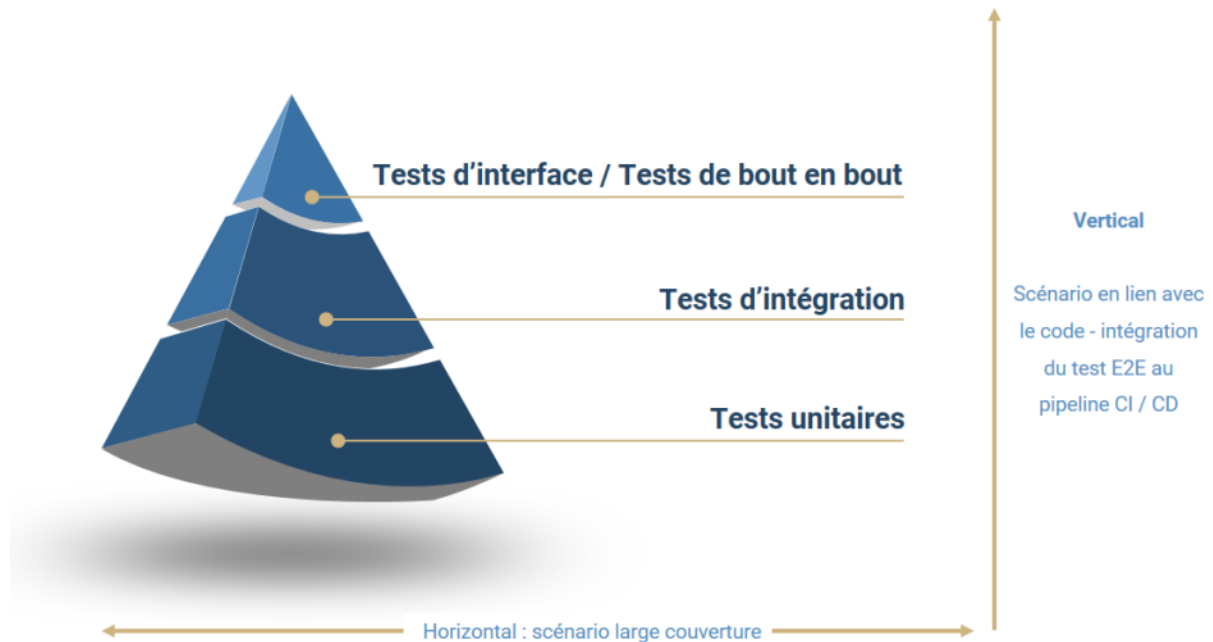
- User se connecte sur le site Z.
- Il recherche le produit W.
- Il change la quantité.
- Il l'ajoute au panier.
- Il rejoint son panier.
- Il supprime l'article (on peut supposer une erreur de l'utilisateur).
- Il retourne sur l'article W.
- Il l'ajoute à nouveau au panier.
- Il retourne sur le panier.
- User choisit livraison en magasin.
- User choisit un magasin.
- Il change de magasin.
- Il règle une partie par carte.
- Il règle une partie avec un bon cadeau.

Les tests de bout en bout verticaux et horizontaux

Un test E2E peut être de deux types : vertical ou horizontal. Plus connus et plus utilisés, la méthode de test horizontal sert à tester l'application en se plaçant du point de vue d'un utilisateur. Cette méthode consiste à tester en suivant le scénario établi afin de confirmer ou non que le système fonctionne comme prévu.

L'approche verticale fait intervenir le concept de couches, autrement dit les tests se déroulent dans un ordre hiérarchique. Chaque composant d'une application est testé du début à la fin pour en assurer la qualité. Le test vertical sert le plus souvent à tester des composants sensibles d'un système complexe qui ne concernent généralement ni les interfaces, ni les utilisateurs. Même si la méthode verticale teste un composant ou morceau de code plus précis, les scénarios de tests doivent rester le plus large possible.

L'objectif étant de trouver un juste équilibre entre l'utilisation des deux méthodes afin qu'elles se complètent.



Rappel

La création d'un test de bout en bout consiste à créer un scénario couvrant le maximum de possibilités d'utilisation d'une application ou d'un logiciel.

Il existe deux méthodes d'utilisation de ses scénarios : la méthode horizontale et la méthode verticale.

Exercice : Quiz

[solution n°1 p.15]

Question 1

Un scénario dépendant d'un autre scénario est un problème.

- ☐ Vrai
- ☐ Faux

Question 2

Dans le cadre de test vertical, le scénario doit être très précis.

- ☐ Vrai
- ☐ Faux

Question 3

Dans le cadre de test horizontal, le scénario peut être utilisé sur différentes applications.

- ☐ Vrai
- ☐ Faux

Question 4

Faire des scénarios spécifiques à un *personas* est une pratique à éviter.

- ☐ Vrai
- ☐ Faux

Question 5

Les tests E2E automatisés peuvent-ils s'intégrer au pipeline CI/CD ?

- ☐ Vrai
- ☐ Faux

III. Mise en place d'un test de bout en bout avec cypress.js

Définition Présentation Cypress

Cypress est un framework JS de tests end-to end. Il s'agit d'un outil open source qui permet de tester facilement les applications.

Méthode Installation de Cypress

Pré-requis :

- Installer VS code¹, ou autre éditeur de texte
- Installer Node.js²
- Installer npm³

Pour commencer il faut créer un nouveau dossier et dans ce dossier initialiser npm.

Utiliser les lignes de commande les unes après les autres, taper entrée pour les exécuter, bien attendre que chaque processus soit terminé pour passer au suivant.

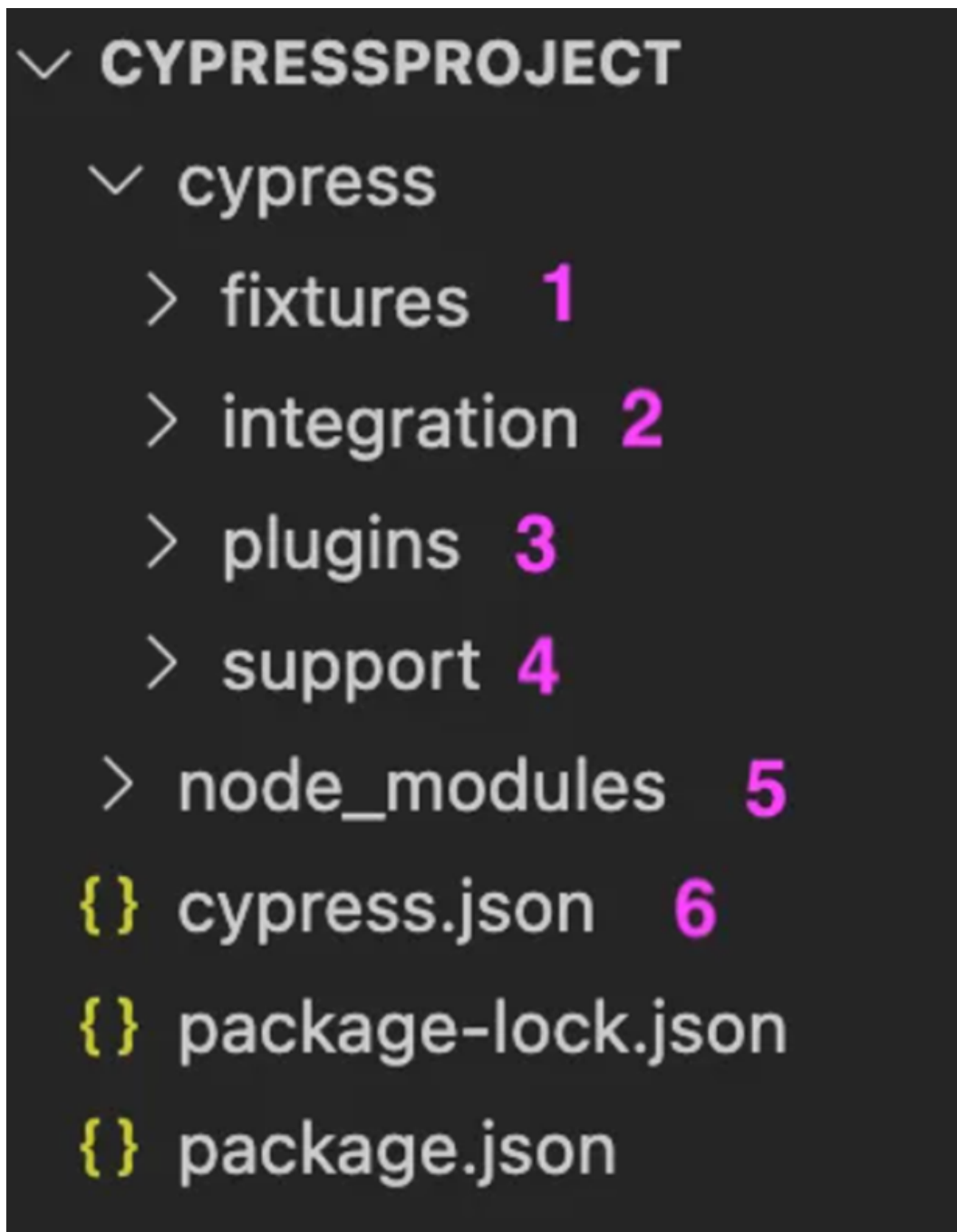
```
1 <code>
2 mkdir cypressproject //crée un dossier
3 cd cypress //entre dans le dossier
4 npm init -y //initialise un projet npm
5 npm install har-validator // permet de corriger une erreur d'installation de cypress
6 npm i cypress --save-dev // installation de cypress
7 node_modules/.bin/cypress open // lance cypress et créer les dossiers de base, garder cette
fenêtre ouverte, si besoin refaite la commande pour la relancer
8 code . // ouvre le dossier dans VS code ou ouvrez votre dossier avec un éditeur
9 </code>
```

1 <https://code.visualstudio.com/download>

2 <https://nodejs.org/en/download/>

3 <https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>

En ouvrant visual code studio, on peut facilement visualiser la structure de notre application.



1. fixtures : ce dossier contient les données statiques et réutilisables tout au long du projet.
2. integration : dans ce dossier, on peut créer plusieurs dossiers et fichiers de test selon l'exigence de notre projet.
3. plugins : il contient les fichiers qui permettent de modifier le comportement interne de cypress.
4. support : sous le dossier de support, nous avons les fichiers qui nous aident à fournir des méthodes standard ou réutilisables.
5. node_modules : ce dossier contient tous les packages npm que nous avons installés. Nous ne modifierons aucun fichier à l'intérieur de ce dossier.
6. cypress.json : nous pouvons ajouter plusieurs configurations dans notre fichier cypress.json. Par exemple, nous pouvons ajouter des variables d'environnement, une URL de base, des délais d'attente, etc.

Complément**Méthode** Mise en ligne d'un site de test

On vous propose de tester un formulaire présent sur un site web. Pour cela il nous faut un site web qui tourne sur un serveur ou localhost. Créer à la racine du cypressproject le fichier index.html qui contiendra notre formulaire.

```
1 <code>
2 <!DOCTYPE html>
3 <html lang="en">
4   <head>
5     <meta charset="UTF-8" />
6     <title>Cypress tutorial for beginners</title>
7   </head>
8   <body>
9     <main>
10      <form>
11        <div>
12          <label for="name">Name</label>
13          <input type="name" required name="name" id="name" />
14        </div>
15        <div>
16          <label for="email">Email</label>
17          <input type="email" required name="email" id="email" />
18        </div>
19        <div>
20          <label for="message">Your message</label>
21          <textarea id="message" name="message" required></textarea>
22        </div>
23        <div>
24          <button type="submit">SEND</button>
25        </div>
26      </form>
27    </main>
28  </body>
29  <script src="form.js"></script>
30 </html>
31 </code>
```

Créer le fichier form.js qui contiendra le code qui permettra la gestion de l'envoi du formulaire.

```
1 <code>
2 form.addEventListener("submit", event => {
3   console.log('envoyé')
4   event.preventDefault();
5 });
6 </code>
```

Enfin retournez dans votre terminal à la racine de votre dossier et taper.

```
1 <code>
2 npx serve // lance notre projet sur serveur interne
3 </code>
```

Voici un visuel du rendu

A screenshot of a web browser window. The address bar shows 'localhost:3000'. The page contains a form with three input fields: 'Name', 'Email', and 'Your message'. Below the 'Your message' field is a 'SEND' button.

Enfin ajouter l'adresse à votre fichier cypress.json.

```
1 <code>
2 {
3   "baseUrl": "http://localhost:3000"
4 }
5 </code>
```

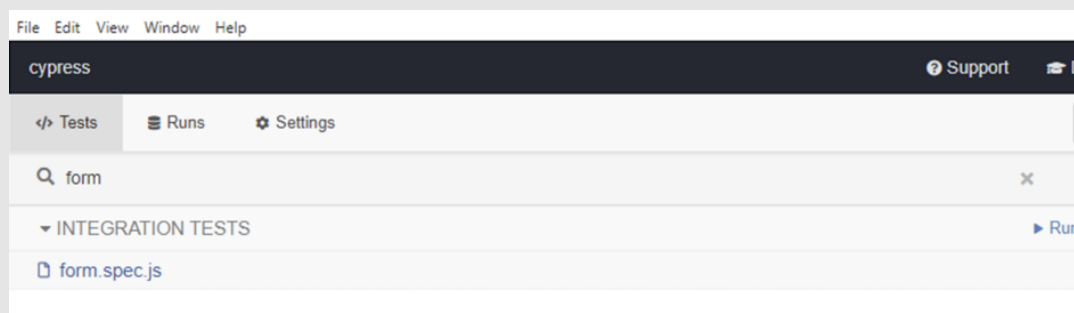
Méthode Création d'un premier test

Les tests doivent toujours être créés dans le dossier cypress/integration. Bien entendu cypress.js utilise ses propres objets et ses propres méthodes, pour aller plus loin dans l'utilisation et les concepts derrière tout ce code la seule solution est de consulter la documentation technique.¹

Créons notre premier test form.spec.js ce test viendra seulement cibler l'élément <form> de notre code HTML, cela n'a pas un grand intérêt en soi mais c'est la base qui vous permettra de réaliser les scénarios les plus complexes très rapidement.

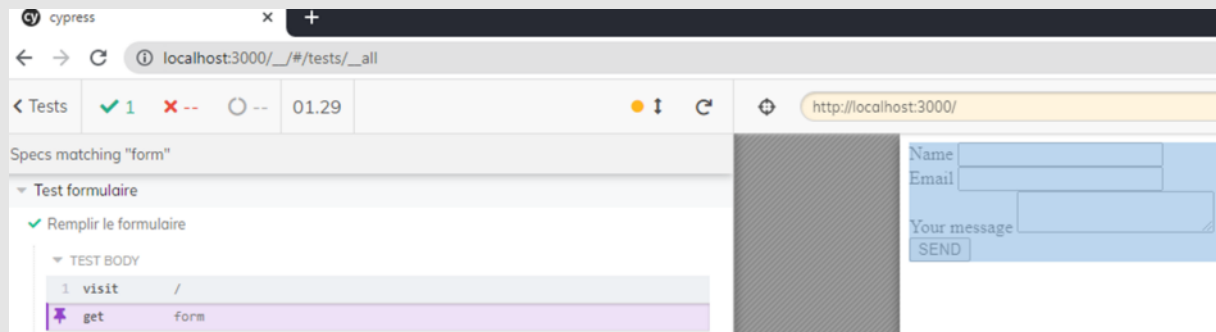
```
1 <code>
2 describe("Test formulaire", () => { // initie le test + description
3   it("Remplir le formulaire", () => { // déclare l'action
4     cy.visit("/"); // url a rechercher
5     cy.get("form"); // une des méthodes permettant de sélectionner un object du DOM
6   });
7 });
8 </code>
```

Retourner dans la fenêtre cypress et rechercher votre test, puis lancer le à l'aide de RUN.



¹ <https://docs.cypress.io/guides/core-concepts/introduction-to-cypress#Cypress-can-be-simple-sometimes>

La fenêtre qui s'ouvre permet de visualiser votre test. Les éléments de gauche sont simplement les blocs étapes .it, .visit et .get présents dans votre fichier de test.



Lorsque l'on clique sur le dernier bloc de notre test, celui-ci cible bien notre élément HTML <form> on peut donc conclure que notre test est fonctionnel.

Exercice : Quiz

[solution n°2 p.15]

Question 1

Lesquels de ses noms de fichiers désignent un fichier de test ?

- ☐ button.json.js
- ☐ form.cyp.js
- ☐ button.spec.js
- ☐ form_spec.js

Question 2

Laquelle de ses lignes de commande relance la fenêtre cypress ?

- ☐ npm cypress
- ☐ npm run cypress
- ☐ npx serve
- ☐ node_modules/.bin/cypress open

Question 3

Quelle commande du terminal permet de créer un dossier ?

- ☐ mkdir
- ☐ npm
- ☐ init
- ☐ run

Question 4

La commande `.get().type()` cypress permet d'écrire seulement dans un input de type « *text* » ?

- ☐ Vrai
- ☐ Faux

Question 5

Cypress permet de cibler seulement les sites du localhost ?

- ☐ Vrai
- ☐ Faux

Question 6

La commande `cy.get('#username').type('bob@burgers.com')` est une commande valide ?

- ☐ Vrai
- ☐ Faux

V. Essentiel

Les tests de bout en bout sont la dernière étape du processus de testing. Ils consistent à se mettre à la place de l'utilisateur et à couvrir un maximum de cas d'utilisation d'une application ou d'un logiciel.

La création d'un test de bout en bout consiste à créer un scénario couvrant le maximum de possibilités d'utilisation d'une application ou d'un logiciel. Il existe deux méthodes d'utilisation de ses scénarios : la méthode horizontale et la méthode verticale.

Le processus de test de bout en bout est rendu possible par l'utilisation de framework de test compatible avec le langage de notre application comme Cypress ou Sélénium.

VI. Auto-évaluation

A. Exercice

Question

[solution n°3 p.17]

À l'aide de la documentation technique de cypress.js et de vos recherches sur les méthodes `.get()` et `.type()` compléter le test vu dans la partie 2 de ce cours afin qu'il remplisse automatiquement le formulaire et le soumette.

B. Test

Exercice 1 : Quiz

[solution n°4 p.17]

Question 1

Le test de bout en bout intervient à quel moment ?

- ☐ Après le build
- ☐ Après les tests unitaires
- ☐ Après le monitoring
- ☐ Après les tests d'intégrations

Question 2

1 <https://www.google.com/search?q=bob%40burgers.com&oq=bob%40burgers.com&aqs=chrome.0.69i59j69i58.1159j0j7&sourceid=chrome&ie=UTF-8>

Laquelle de ses couches n'est pas testée par les tests E2E ?

- ☐ La couche UI
- ☐ La couche logique
- ☐ La couche éditique
- ☐ La couche de persistance

Question 3

Que nous permet l'automatisation des test E2E ?

- ☐ Diminuer les coûts
- ☐ Diminuer les tests
- ☐ Augmenter les coûts
- ☐ Augmenter les tests

Question 4

Le test End-to-End valide un certain morceau de code ?

- ☐ Vrai
- ☐ Faux

Question 5

Les tests E2E doivent-ils rester le plus simple possible pour éviter les écritures complexes ?

- ☐ Vrai
- ☐ Faux


Solutions des exercices

Exercice p. 6 Solution n°1**Question 1**

Un scénario dépendant d'un autre scénario est un problème.

☒ Vrai

☐ Faux


 Si un scénario dépendant d'un autre scénario est à l'origine d'un bug, il pourra s'avérer compliqué à reproduire.

Question 2

Dans le cadre de test vertical, le scénario doit être très précis.

☐ Vrai

☒ Faux


 Le scénario doit prendre en compte le code tester dans les étapes précédentes de la pyramide et l'incorporer dans un scénario plus global.

Question 3

Dans le cadre de test horizontal, le scénario peut être utilisé sur différentes applications.

☒ Vrai

☐ Faux


 Il est fortement possible qu'un scénario en rapport avec un site de e-commerce puisse être utilisable pour le test d'un autre site e-commerce.

Question 4

Faire des scénarios spécifiques à un *personas* est une pratique à éviter.

☒ Vrai

☐ Faux


 En théorie, notre scénario doit couvrir un maximum de possibilités applicables à un maximum de *personas*. En pratique, il n'est pas toujours possible de tester certaines parties d'une application sans écrire de scénario plus spécifique.

Question 5

Les tests E2E automatisés peuvent-ils s'intégrer au pipeline CI/CD ?

☒ Vrai

☐ Faux


 Dans le cadre du déploiement continue, les tests automatisés sont un facteur clé du déploiement en production.

Exercice p. 11 Solution n°2

Question 1

Lesquels de ses noms de fichiers désignent un fichier de test ?


- ☐ button.json.js
- ☐ form.cyp.js
- ☒ button.spec.js
- ☒ form_spec.js

 Il est courant de suivre des conventions de nommage dans les métiers du développement web. De manière générale, il faut faire apparaître spec ou _spec dans le nom des fichiers de test.

Question 2

Laquelle de ses lignes de commande relance la fenêtre cypress ?


- ☐ npm cypress
- ☐ npm run cypress
- ☐ npx serve
- ☒ node_modules/.bin/cypress open

 La commande exacte est open mais elle doit être effectuée dans le dossier cypress de node_modules. Le plus souvent notre terminal est ouvert à la racine du projet donc node_modules/.bin/cypress open.

Question 3

Quelle commande du terminal permet de créer un dossier ?


- ☒ mkdir
- ☐ npm
- ☐ init
- ☐ run

 mkdir est une commande Unix permettant de créer des répertoires.

Question 4

La commande .get().type() cypress permet d'écrire seulement dans un input de type « text » ?

- ☐ Vrai
- ☒ Faux

 Elle supporte les éléments text, password, email number, date, week, month, time, datetime, search, url et tel.

Question 5

Cypress permet de cibler seulement les sites du localhost ?

- ☐ Vrai
- ☒ Faux

- 🔍 La commande `.visit` permet de cibler n'importe quelle URL, mais aussi de spécifier la page et plein d'autre paramètre - Cypress¹ -

Question 6

La commande `cy.get('#username').type('bob@burgers.com2')` est une commande valide ?

- ☒ Vrai
- ☐ Faux

- 🔍 C'est vrai, elle permet d'injecter le texte `bob@burgers.com3` dans l'input avec un `id='username'`.

p. 12 Solution n°3

```
1 <code>
2 describe("Test formulaire", () => { // initie le test + description
3     it("Remplir le formulaire", () => { // déclare l'action
4         cy.visit("/"); // url a rechercher
5
6         cy.get('input[name="name"]')
7         .type("Molly")
8
9         cy.get('input[name="email"]')
10        .type("molly@dev.dev")
11
12        cy.get("textarea")
13        .type("Mind you if I ask some silly question?");
14
15        cy.get("form").submit();
16    });
17 });
18 </code>
```

Exercice p. 12 Solution n°4

Question 1

Le test de bout en bout intervient à quel moment ?

- ☐ Après le build
- ☐ Après les tests unitaires
- ☐ Après le monitoring
- ☒ Après les tests d'intégrations

- 🔍 Les tests E2E sont la dernière étape dans la pyramide des tests, ils interviennent donc après les tests d'intégrations.


1 <https://docs.cypress.io/api/commands/visit#Visit-is-automatically-prefixed-with-baseUrl>

2 <https://www.google.com/search?q=bob%40burgers.com&oq=bob%40burgers.com&aqs=chrome.69i59j69i58.1159j0j7&sourceid=chrome&ie=UTF-8>

3 <https://www.google.com/search?q=bob%40burgers.com&oq=bob%40burgers.com&aqs=chrome.0.69i59j69i58.1171j0j9&sourceid=chrome&ie=UTF-8>


Question 2

Laquelle de ses couches n'est pas testée par les tests E2E ?

- ☐ La couche UI
 - ☐ La couche logique
 - ☒ La couche éditique
 - ☐ La couche de persistance
-  Une couche éditique est un terme utilisé pour définir les ressources dédiées à l'impression et l'acquisition de document papier. Cette notion n'a aucun lien avec les tests E2E.


Question 3

Que nous permet l'automatisation des test E2E ?

- ☒ Diminuer les coûts
 - ☐ Diminuer les tests
 - ☐ Augmenter les coûts
 - ☒ Augmenter les tests
-  L'automatisation du processus permet une diminution des tâches répétitives, les équipes de tests ont donc plus de temps pour déployer plus de tests. De plus, les tests sont joués plus rapidement et donc moins coûteux.


Question 4

Le test End-to-End valide un certain morceau de code ?

- ☐ Vrai
 - ☒ Faux
-  Les tests E2E tendent à valider au maximum des modules / applications dans leurs globalités.

Question 5

Les tests E2E doivent-ils rester le plus simple possible pour éviter les écritures complexes ?

- ☐ Vrai
 - ☒ Faux
-  L'écriture de test complexe reste simple à mettre en place , c'est d'ailleurs une bonne pratique d'écrire des tests complexes couvrant un maximum de possibilité.