

# **Le test fonctionnel**

# Table des matières

<b>I. Les tests fonctionnels</b>	<b>3</b>
<b>II. Exercice : Quiz</b>	<b>6</b>
<b>III. Mise en place d'un test fonctionnel avec cypress.js</b>	<b>7</b>
<b>IV. Exercices</b>	<b>12</b>
A. Exercice : QCU/QCM .....	12
B. Exercice : Ordonnancement.....	13
<b>V. Essentiel</b>	<b>13</b>
<b>VI. Auto-évaluation</b>	<b>14</b>
A. Exercice .....	14
B. Test.....	14
<b>Solutions des exercices</b>	<b>15</b>

# I. Les tests fonctionnels

**Durée :** 1 h 30

**Prérequis :**

- Savoir utiliser le terminal
- Installer Node.js
- Installer npm
- Installer Cypress.js

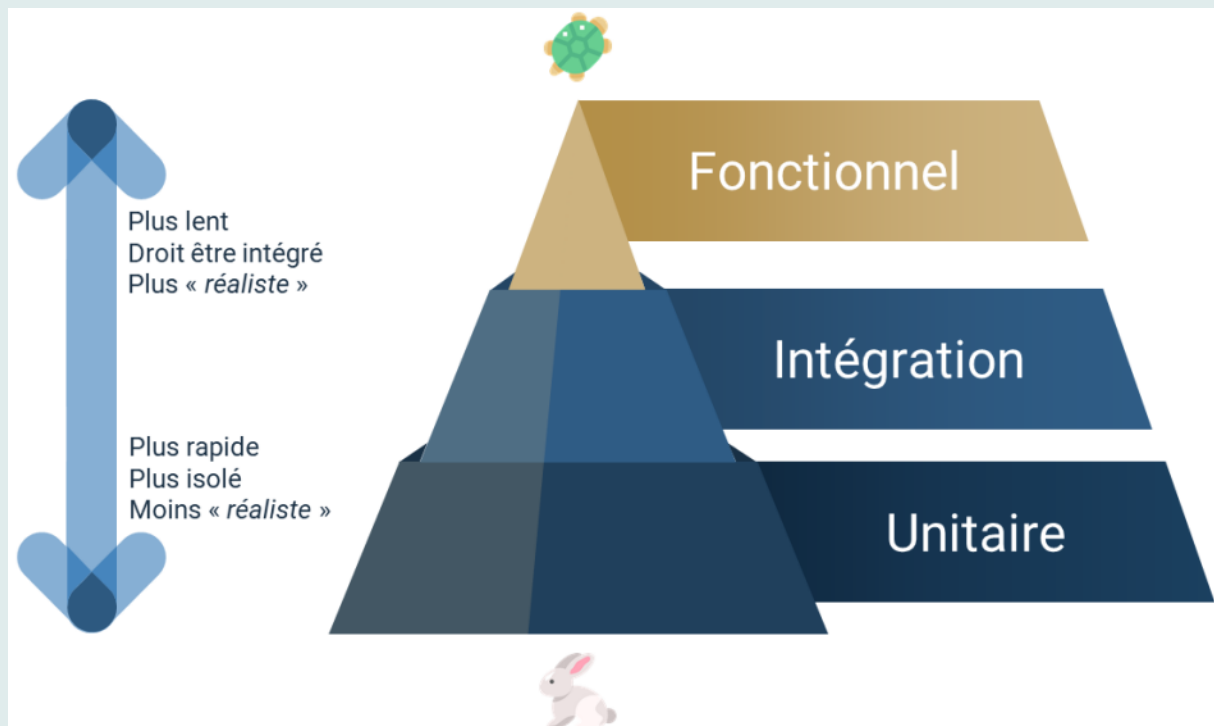
**Environnement de travail :** Visual Studio Code

## Contexte

Dans un projet informatique, des sites web ou des applications sont créés pour répondre aux besoins des clients. Cette exigence client prend la forme d'une liste de fonctionnalités qui ont chacune une liste d'exigences fonctionnelles. En tant que professionnel, vous aurez à effectuer un grand nombre de tests fonctionnels. Voyons ensemble en quoi cela consiste.

## Définition Test fonctionnel

Dans le prolongement des tests unitaires, les tests fonctionnels sont destinés à s'assurer que toutes les fonctionnalités de votre application sont conformes aux exigences attendues.



## Les objectifs du test fonctionnel

« Est-ce que l'application marche dans son ensemble ? » Pour répondre à cette question, le test fonctionnel s'appuie sur le cahier des charges. Par exemple, sur un site e-commerce, vous pouvez avoir à tester qu'un utilisateur puisse créer un compte, se connecter, mettre au panier un article, payer l'article, etc.

Il est également possible de dérouler un scénario composé d'une liste d'actions et pour chaque action d'effectuer une liste de vérifications pour arriver à un résultat attendu. Cela va permettre d'imiter le comportement d'un utilisateur lorsque celui-ci teste une fonctionnalité.

Les objectifs du test fonctionnel sont multiples :

- Valider que le logiciel est conforme aux expressions de besoins,
- Garantir la qualité du livrable,
- Valider le bon fonctionnement des parcours utilisateurs.

NB : le test fonctionnel n'a pas pour objectif de diagnostiquer la cause des erreurs.

## Les niveaux des tests

Il existe d'après l'ISTQB (International Software Testing Qualifications Board), 4 différents niveaux de tests.



- Les tests de composants :

Le but des tests de composants est de tester différents composants d'un site (ou d'un logiciel) pour s'assurer que chacun peut fonctionner normalement.

Par exemple, pour une déconnexion, le bouton « *se déconnecter* » est un composant.

- Les tests d'intégration :

Le test d'intégration est un test effectué entre les composants pour s'assurer que l'interaction entre les différents éléments s'exécute normalement. Ces tests sont généralement gérés par les développeurs.

Exemple : lors d'une authentification, on vérifie que le message envoyé après l'appui sur le bouton « *Se connecter* » est bien reçu par le serveur d'authentification.

Ces tests peuvent être automatisés ou manuels.

- Les tests système :

Ce sont les tests au sens le plus instinctif et c'est généralement les seuls qui sont effectués par les ingénieurs de tests.

Leur objectif est de vérifier que l'application répond aux exigences définies dans les spécifications. Exemple : pendant un test système, on vérifie que l'authentification fonctionne bien, que les bonnes erreurs sont remontées, etc.

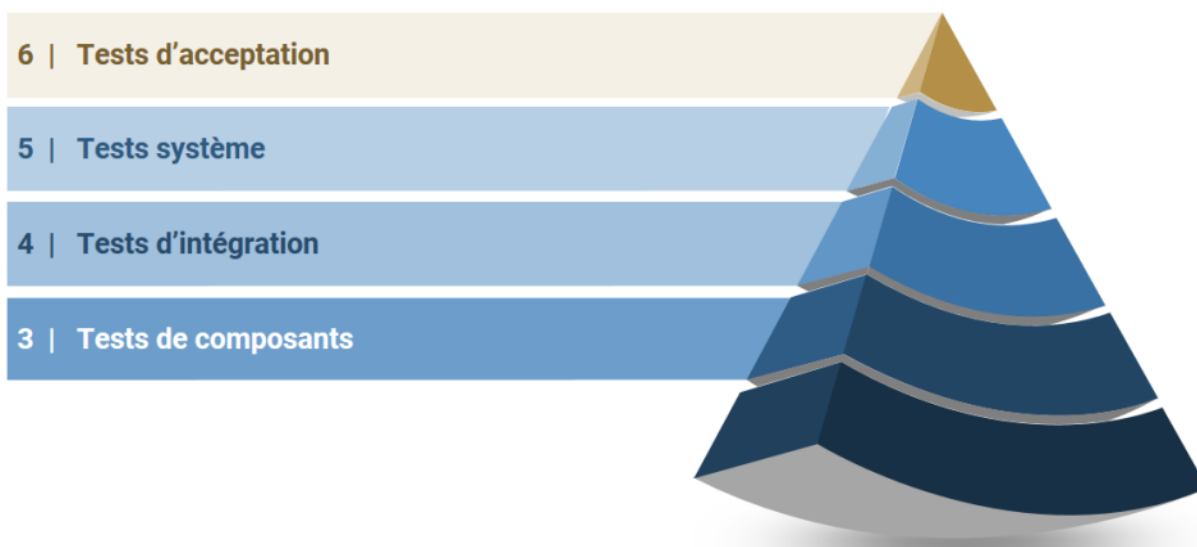
Ces tests peuvent être manuels ou automatisés. Pour un meilleur bénéfice, il est préférable de faire un mixte entre les tests manuel et automatisés.

- Les tests d'acceptation :

Ce sont les tests « *finaux* » qui ont pour but de confirmer que le produit final correspond bien aux exigences des utilisateurs.

NB : ce n'est pas parce qu'une application répond aux spécifications qu'elle répond aux besoins des utilisateurs. Cela peut arriver pour plusieurs raisons telles que des problèmes dans les spécifications, des problèmes d'ergonomie, etc.

Les tests d'acceptation sont des tests manuels.



#### Définition Les processus itératifs

Ils désignent la pratique qui consiste à créer, affiner et améliorer un projet, un produit ou une initiative. Les équipes qui utilisent ce type de **processus** créent, testent et révisent jusqu'à ce qu'elles soient satisfaites du résultat final.

#### Trois méthodes pour trois types de test fonctionnel

- Méthode « *Capture + Replay* » : elle inclut l'enregistrement du parcours utilisateur en jouant étape par étape le contexte d'exécution. Les opérations manuelles seront enregistrées pour une utilisation ultérieure. Cette méthode est simple pour créer rapidement des tests fonctionnels, mais elle est très compliquée à maintenir pendant le processus itératif.
- Méthode de scénario de test : le but de cette méthode est de tester de bout en bout pour un problème complexe spécifique du logiciel.
- Méthode structurée basée sur des scripts : cela inclut l'écriture de scripts de chemin et l'utilisation d'une sémantique dédiée qui décrit les opérations. C'est très puissant et flexible.

La maintenance demande à l'informaticien de l'expérience et des compétences dédiées.

#### Les tests de progression et les tests de non régression

Test de progression	Test de non-régression
Test réalisé sur une exigence qui vient d'être développée.	Réalisé sur une exigence existante déjà avant les développements en cours.  Il permet de vérifier si les corrections ou évolutions dans le code n'ont pas créées de nouvelles anomalies

Si une action échoue, lors d'un test fonctionnel, ou qu'une vérification n'est pas satisfaite, on est alors en présence d'un bug informatique.

## Bugs

Il existe deux types de bugs.

Bug d'implémentation	Bug de régression
Un problème introduit lors du développement initial d'une fonctionnalité.	Un bug apparaît lors du développement d'une autre fonctionnalité.  Lors d'une maintenance corrective sur la fonctionnalité en question une autre.

### Définition Automatisation des tests

Un test automatisé est un test où l'exécution ne nécessite pas l'intervention d'un humain.

L'automatisation des tests sert à conforter les chemins critiques des applicatifs, ainsi que les régressions, afin de trouver au plus tôt les anomalies et réduire le coût des correctifs.

L'automatisation du test sert à réduire les efforts nécessaires au test, et considérablement augmenter la quantité de tests effectués dans un temps limité. Ce qui permet de trouver au plus tôt les anomalies, réduire le coût des correctifs et éviter les régressions.

L'automatisation des tests est nécessaire selon les critères suivants :

- La taille du projet
- Le nombre de tests à mener
- La fréquence de répétition des tâches
- La durée du projet et sa complexité

### Les avantages liés à l'automatisation des tests fonctionnels

Automatiser les tests fonctionnels présente plusieurs avantages :

- Accroître la rapidité des tests
- Paralléliser les tests
- Augmenter la fréquence des tests
- Permettre la planification des tests
- Réduire la maintenance corrective

### Complément La norme ISO 9126

## Exercice : Quiz

[solution n°1 p.17]

### Question 1

Un test fonctionnel sert à tester le bon fonctionnement d'une fonction ?

- ☐ Vrai
- ☐ Faux

### Question 2

Parmi ces niveaux lesquels n'existent pas selon l'ISTQB ?

- ☐ Test de composant
- ☐ Test amélioration
- ☐ Test d'intégration
- ☐ Test système
- ☐ Test de non régression
- ☐ Test manuel
- ☐ Test d'acceptation

Question 3

La méthode structurée basée sur des scripts consiste à écrire les étapes et le chemin d'utilisation d'un composant.

- ☐ Vrai
- ☐ Faux

Question 4

Il existe deux types de bug :

- ☐ Bug d'implémentation, Bug de régression
- ☐ Bug d'implémentation, Bug de non-régression
- ☐ Bug de progression, Bug de régression
- ☐ Bug de progression, Bug de non-régression

Question 5

L'automatisation des tests est-elle obligatoire ?

- ☐ Vrai
- ☐ Faux

### III. Mise en place d'un test fonctionnel avec cypress.js

#### **Définition**    **Présentation Cypress**

Cypress est un framework JS destiné au testing d'application.

#### **Méthode**    **Installation de Cypress**

Prérequis :

- Installer VS code<sup>1</sup>, ou autre éditeur de texte
- Installer Node.js<sup>2</sup>
- Installer npm<sup>3</sup>

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

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

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

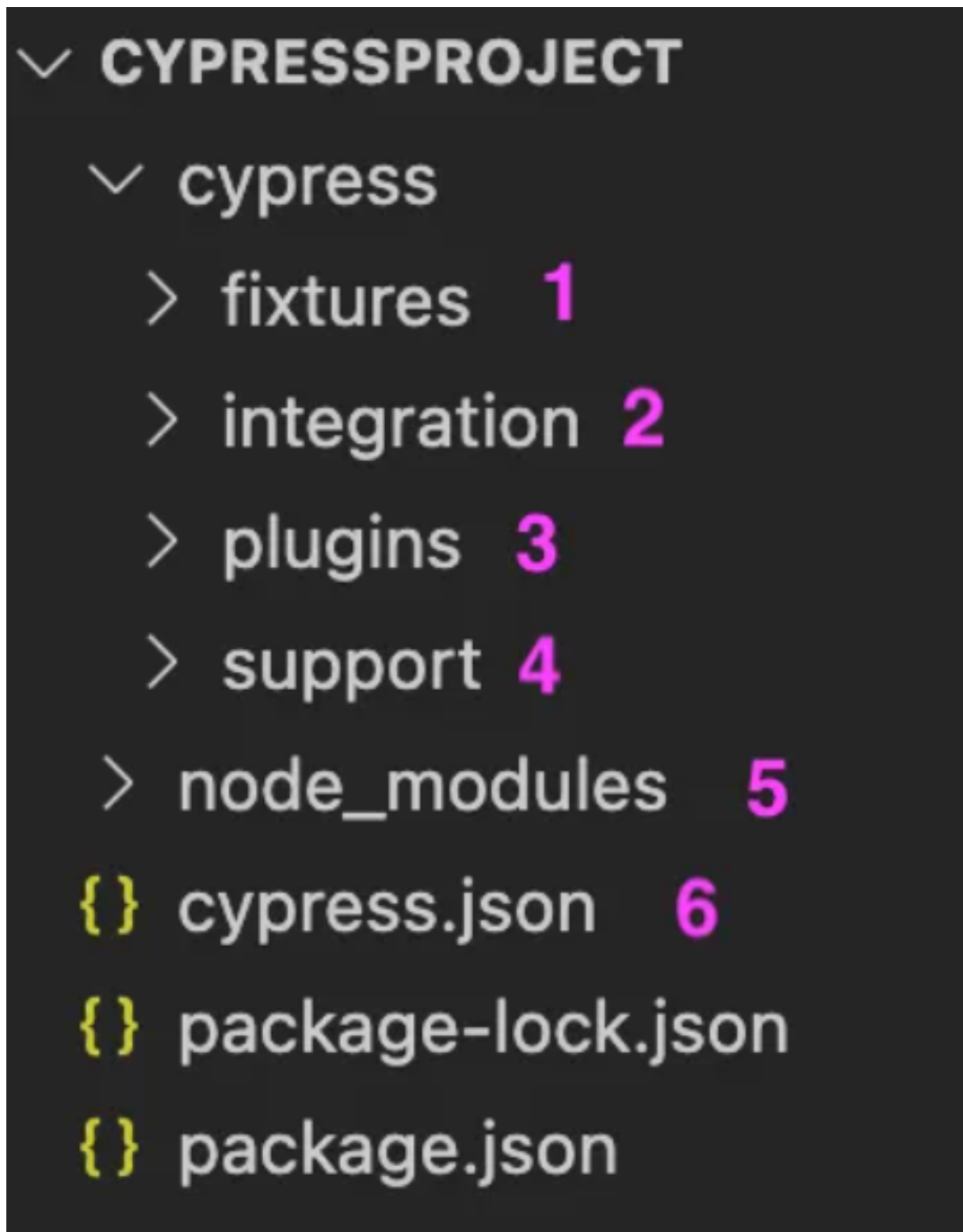
Pour commencer il faut créer un nouveau dossier et dans ce dossier initialiser npm.

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

```
1 <code>
2 mkdir cypressProject //crée un dossier
3 cd cypressProject //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>
```



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



1. fixtures : contient les données statiques et réutilisables tout au long du projet.
2. integration : dossier où l'on crée les tests.
3. plugins : contient les fichiers qui permettent de modifier le comportement interne de cypress.
4. support : 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.

#### Méthode Mise en ligne d'un site de test

On vous propose la mise en forme d'une page présente sur un site web pour voir s'il correspond au cahier des charges. Pour cela il nous faut un site web qui tourne sur un serveur ou localhost. Créez à la racine de cypressTest le fichier index.html qui contiendra notre magnifique site.

```

1 <code>
2
3 <!DOCTYPE html>
4 <html lang="en">
5   <head>
6     <meta charset="UTF-8" />
7     <title>Cypress tutorial for beginners</title>
8     <link rel="stylesheet" href="style.css">
9   </head>
10  <body>
11    <h1>Hello </h1>
12    <p>Ce texte est bleu</p>
13    <button class="button" type="button">Je suis un bouton </button>
14  </body>
15 </html>
16
17 </code>

```

Créez le fichier style.css pour donner du style à notre page

```

1 <code>
2
3 p {color:blue}
4 button {
5   border: 0;
6   line-height: 2.5;
7   padding: 0 20px;
8   font-size: 1rem;
9   text-align: center;
10  color: #fff;
11  text-shadow: 1px 1px 1px #000;
12  border-radius: 10px;
13  background-color: rgba(220, 0, 0, 1);
14  background-image: linear-gradient(to top left,
15                                   rgba(0, 0, 0, .2),
16                                   rgba(0, 0, 0, .2) 30%,
17                                   rgba(0, 0, 0, 0));
18  box-shadow: inset 2px 2px 3px rgba(255, 255, 255, .6),
19              inset -2px -2px 3px rgba(0, 0, 0, .6);
20 }
21 </code>

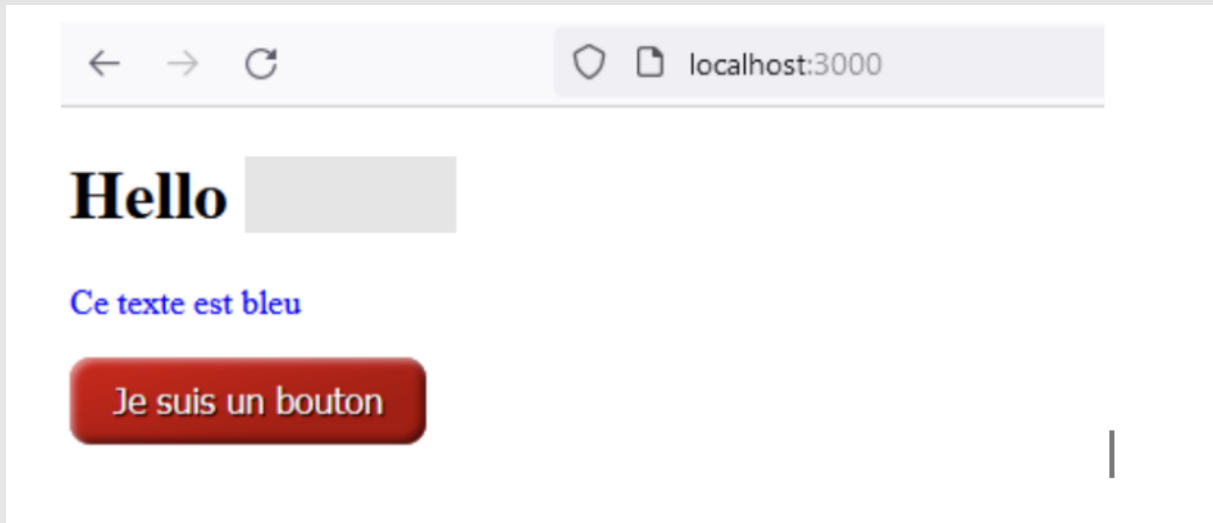
```

Enfin retournez dans votre terminal à la racine de votre dossier et tapez

```

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

```



Enfin ajoutez 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 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<sup>1</sup>.

Introduction to Cypress<sup>2</sup>

Créons notre premier test first.spec.js ; ce test viendra seulement vérifier le contenu de la balise <h1> cela n'a pas un grand intérêt en soi mais ciblé un élément, évaluer son contenu et la base qui vous permettra de réaliser des scénarios de plus en plus complexes.

```
1 <code>
2 describe("First test", () => { // initie le test + description
3   it("Vérifier le contenu de la page", () => { // déclare l'action
4     cy.visit("/"); // url à rechercher
5
6     //On prend la balise h1
7     cy.get('h1');
8     // On récupère son contenu dans une variable
9     cy.should(Titre => {
10      expect(Titre).to.contain("Hello studi"); // on check si la variable contient Hello studi
11    });
12  });
13 });
14
15 </code>
```

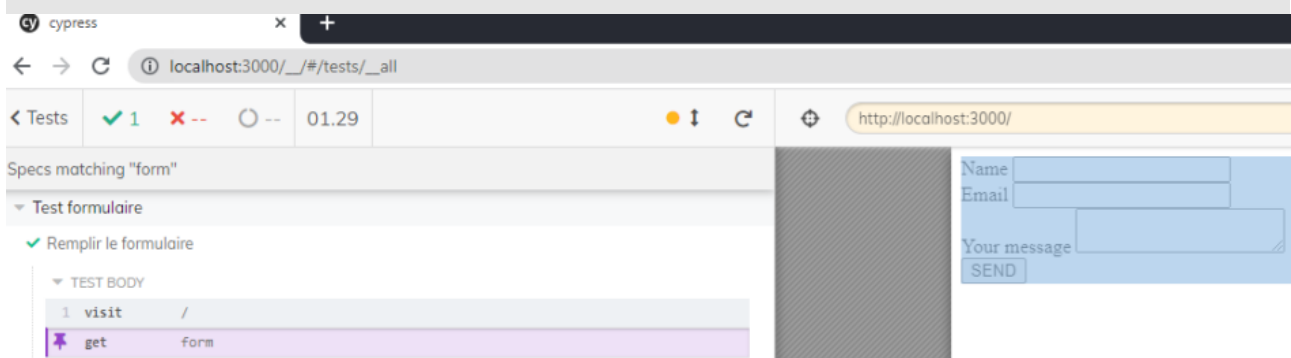
Retournez dans la fenêtre cypress et recherchez votre test, puis lancez le à l'aide de RUN.

1 <https://docs.cypress.io/guides/core-concepts/introduction-to-cypress#Cypress-Can-Be-Simple-Sometimes>

2 <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 .visit() et .get() .should() décrit dans le code de votre first.spec.js



Lorsque l'on clique sur le dernier bloc de notre test, celui-ci cible bien notre élément HTML `<h1>` et nous retourne l'information « *expected <h1> to contain Hello studi* » notre test fonctionnel est donc valide et validé.

## IV. Exercices

### A. Exercice : QCU/QCM

[solution n°2 p.18]

#### Question 1

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

- ☐ button.json.js
- ☐ form.cyp.js
- ☐ login.spec.js
- ☐ button\_spec.js

#### Question 2

Un bug de régression est un bug qui est introduit lors du développement initial d'une fonctionnalité.

- ☐ Vrai
- ☐ Faux

#### Question 3

Cypress est un langage de programmation.

- ☐ Vrai
- ☐ Faux

Question 4

cy.should() crée une assertion (assert) ?

- ☐ Vrai
- ☐ Faux

### B. Exercice : Ordonnancement

[solution n°3 p.19]

Mettez ces étapes dans l'ordre.

1. Effectuer l'analyse de l'application
2. Valider l'application
3. Pour chaque exigence déterminez le scénario de test
4. Lire le cahier des charges et déterminer les exigences
5. Pour chaque scénario, il faut déterminer les actions possibles (case test)

Réponse : \_\_\_\_

## V. Essentiel

Les tests fonctionnels sont destinés à s'assurer que toutes les fonctionnalités sont conformes aux exigences inscrites dans le cahier des charges. Cela va permettre de valider le bon fonctionnement des parcours utilisateurs, et par suite garantir la qualité du livrable.

Il existe 4 niveaux de test :

Niveaux test :

- Test de composants
- Test d'intégration
- Test système
- Tests d'acceptation

On peut utiliser trois méthodes pour faire des tests fonctionnels, la première consiste à faire des captures d'écran du parcours pris par l'utilisateur. Cette méthode est simple à créer, mais difficile à maintenir. D'autre part, la deuxième méthode consiste à créer des scénarios des tests, ce qui permet de tester de bout en bout une fonctionnalité. Cette méthode est simple à faire et à maintenir. Pour finir, on trouve la méthode basée sur les scripts. Cette méthode consiste à écrire le chemin qui décrit l'utilisation, Cette méthode est flexible, mais elle demande de l'expérience et avoir des compétences dédiées.

Enfin, il est possible d'augmenter la qualité des livrables en automatisant les tests, cela veut dire que l'exécution ne nécessite pas l'intervention d'un humain. D'ailleurs, elle représente plusieurs avantages, parmi lesquels on peut trouver : l'augmentation de la fréquence des tests, où la réduction de la maintenance corrective.

## VI. Auto-évaluation

### A. Exercice

#### Question

[solution n°4 p.19]

À l'aide de la documentation technique de cypress.js et de vos recherches sur les méthodes .get() et .should() compléter le test précédent afin de vérifier que le texte de la balise <p> est bien *blue*. Vérifier que le background et la couleur du texte du bouton son conforme au cahier des charges (dans notre cas le fichier .css)

#### Indice :

- Plusieurs it() peuvent s'utiliser à la suite,
- Should peut avoir en argument hace.css (voir doc).

### B. Test

#### Exercice 1 : Quiz

[solution n°5 p.19]

##### Question 1

Les tests fonctionnels intègrent les tests :

- ☐ Unitaire
- ☐ D'acceptation
- ☐ Système
- ☐ De composant
- ☐ End-to-end

##### Question 2

Les tests fonctionnels sont-ils :

- ☐ Plus lents mais plus réalistes quant à l'utilisation de l'application finale
- ☐ Plus rapides mais moins réalistes quant à l'utilisation de l'application finale
- ☐ Plus lents et moins réalistes quant à l'utilisation de l'application finale
- ☐ Plus rapides et plus réalistes quant à l'utilisation de l'application finale

##### Question 3

Les tests « *finaux* » représentent les tests :

- ☐ Unitaires
- ☐ D'acceptation
- ☐ Système
- ☐ De composant
- ☐ End-to-end

##### Question 4

Quelle méthode inclut l'enregistrement du parcours utilisateurs ?

- ☐ « *capture + replay* »
- ☐ Scénario
- ☐ Structurée

Question 5

Quel répertoire contient les packages installés par npm ?

- ☐ Cypress
- ☐ node\_modules
- ☐ Plugins

## Solutions des exercices






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

Un test fonctionnel sert à tester le bon fonctionnement d'une fonction ?

☐ Vrai

☒ Faux

 Faux, le test fonctionnel sert à tester l'enchaînement de différentes fonctionnalités et vérifier qu'elles sont conformes aux besoins exprimés par les métiers.

**Question 2**

Parmi ces niveaux lesquels n'existent pas selon l'ISTQB ?

☐ Test de composant

☒ Test amélioration


☐ Test d'intégration

☐ Test système

☒ Test de non régression

☒ Test manuel

☐ Test d'acceptation


 Selon l'ISTQB il existe quatre niveaux de test : les tests de composants, les tests d'intégrations, les tests systèmes et les tests d'acceptation.

**Question 3**

La méthode structurée basée sur des scripts consiste à écrire les étapes et le chemin d'utilisation d'un composant.

☒ Vrai

☐ Faux

 Vrai, cette méthode inclut l'écriture de scripts de chemin et l'utilisation d'une sémantique dédiée qui décrit les opérations.

**Question 4**


Il existe deux types de bug :

☒ Bug d'implémentation, Bug de régression

☐ Bug d'implémentation, Bug de non-régression

☐ Bug de progression, Bug de régression

☐ Bug de progression, Bug de non-régression


 Il existe deux types de bug : bug d'implémentation et Bug de régression.  
Il y a deux types de test : test de progression et test de non-régression.

### Question 5

L'automatisation des tests est-elle obligatoire ?

☐ Vrai

☒ Faux

 L'automatisation des tests est nécessaire selon les critères suivants :

- La taille du projet
- Le nombre de tests à mener
- La fréquence de répétition des tâches
- La durée du projet et sa complexité

### Exercice p. 12 Solution n°2

#### Question 1


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

☐ button.json.js

☐ form.cyp.js

☒ login.spec.js

☒ button\_spec.js


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

#### Question 2

Un bug de régression est un bug qui est introduit lors du développement initial d'une fonctionnalité.

☐ Vrai

☒ Faux


 Un bug d'implémentation est un bug qui est introduit lors du développement initial d'une fonctionnalité.

#### Question 3

Cypress est un langage de programmation.

☐ Vrai

☒ Faux

 Cypress est un framework et qui repose sur le langage javascript.

#### Question 4

cy.should() crée une assertion (assert) ?

☒ Vrai

☐ Faux

Q Les assertions sont automatiquement réessayées jusqu'à ce qu'elles passent ou expirent.

### Exercice p. 13 Solution n°3

Mettez ces étapes dans l'ordre.

Lire le cahier des charges et déterminer les exigences

Pour chaque exigence déterminez le scénario de test

Pour chaque scénario, il faut déterminer les actions possibles (case test)

Effectuer l'analyse de l'application

Valider l'application

- Q
- Lire le cahier des charges et déterminer les exigences,
  - Pour chaque exigence déterminez le scénario de test,
  - Pour chaque scénario, il faut déterminer les actions possibles (case test),
  - Effectuer l'analyse de l'application,
  - Valider l'application.

### p. 14 Solution n°4


```
1 <code>
2 describe("First test", () => { // initie le test + description
3   it("Vérifier le contenu de la page", () => { // déclare l'action
4     cy.visit("/"); // url a rechercher
5     //On prend la balise h1
6     cy.get('h1');
7     // On récupère son contenu dans une variable
8     cy.should('Titre' => {
9       expect(Titre).to.contain("Hello studi"); // on check si la variable contient Hello studi
10    });
11  });
12  it('Vérification de la couleur du texte', () => {
13    //On prend la balise "p"
14    cy.get('p');
15    //On vérifie la couleur format rgb
16    cy.should('have.css', 'color', 'rgb(0, 0, 255)');
17  });
18  it('Vérification du bouton', () => {
19    //On prend la balise button à l'aide de sa classe
20    cy.get('.button');
21    //On vérifie que le texte dans le bouton est blanc
22    cy.should('have.css', 'color', 'rgb(255, 255, 255)');
23    //On vérifie que le fond du bouton est rouge
24    cy.should('have.css', 'background-color', 'rgb(220, 0, 0)');
25  });
26  });
27 </code>
```

### Exercice p. 14 Solution n°5

### Question 1

Les tests fonctionnels intègrent les tests :


- ☐ Unitaire
- ☒ D'acceptation
- ☒ Système
- ☐ De composant
- ☐ End-to-end

 Il existe d'après l'ISTQB (International Software Testing Qualifications Board) 4 différents niveaux de tests dont 2 font partie des tests fonctionnels : les tests d'acceptation et les tests système.

### Question 2

Les tests fonctionnels sont-ils :


- ☒ Plus lents mais plus réalistes quant à l'utilisation de l'application finale
- ☐ Plus rapides mais moins réalistes quant à l'utilisation de l'application finale
- ☐ Plus lents et moins réalistes quant à l'utilisation de l'application finale
- ☐ Plus rapides et plus réalistes quant à l'utilisation de l'application finale

 Les tests fonctionnels sont effectivement plus lents mais ils ont l'avantage d'être plus réalistes quant à l'utilisation de l'application finale.

### Question 3

Les tests « *finaux* » représentent les tests :


- ☐ Unitaires
- ☒ D'acceptation
- ☐ Système
- ☐ De composant
- ☐ End-to-end

 Les tests d'acceptation sont les tests « *finaux* » qui ont pour but de confirmer que le produit final correspond bien aux exigences des utilisateurs.

### Question 4

Quelle méthode inclut l'enregistrement du parcours utilisateurs ?


- ☒ « *capture + replay* »
- ☐ Scénario
- ☐ Structurée

 La méthode « *capture + replay* » inclut l'enregistrement du parcours utilisateur en jouant étape par étape le contexte d'exécution.

### Question 5

---

Quel répertoire contient les packages installés par npm ?

- ☐ Cypress
- ☒ node\_modules
- ☐ Plugins
-  C'est le dossier « *node\_modules* » qui contient tous les packages installés par npm.