Chapitre 3
Configuration de Cypress

1. Configurez Cypress

La configuration de l'outil vous permet de vous assurer que vos tests sont non seulement efficaces mais aussi adaptés à l'environnement dans lequel ils s'exécutent. Cela inclut l'établissement des paramètres d'exécution des tests, la gestion des différentes versions, et l'adaptation du framework à vos besoins spécifiques.

1.1 Exécution de Cypress

Démarrer avec Cypress est relativement simple, mais nécessite une attention particulière aux détails lors de la configuration initiale. Il est essentiel de comprendre comment il interagit avec votre navigateur et votre application pour tirer le meilleur parti de vos tests.

Rien de plus simple, car Cypress a un assistant intégré à l'outil qui vous aide à le configurer correctement.

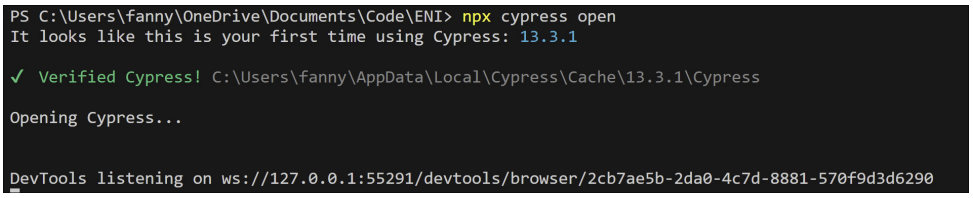
Comme nous l'avons vu précédemment, pour lancer l'assistant, vous devez exécuter la ligne de commande dans votre outil de développement :
`npx cypress open`.

Au chapitre précèdent, nous avons aussi vu comment installer Cypress avec la commande `npm install cypress -save-dev`.

Vous pouvez ensuite le lancer avec `npm run cypress :open` ou `npm run cypress :run`.

Ici, en utilisant npx, qui est un outil inclus à npm, vous pouvez exécuter des paquets node.js directement, sans avoir besoin de les installer.

Dans la console, vous devriez avoir quelque chose de similaire :



Remarque

Cypress est continuellement mis à jour avec de nouvelles fonctionnalités, des corrections de bugs et des améliorations de performance. Lorsque nous discutons des versions antérieures à 10 et postérieures (ou égales) à 10, nous devons prendre en compte que des changements significatifs peuvent affecter la façon dont les développeurs configurent et utilisent Cypress.

Voici un examen approfondi des différences de configuration entre les versions antérieures à 10 et supérieures à 10. Au moment où ce livre est publié, nous sommes à la version 13.

1.2 Avant Cypress 10

Cypress a toujours été axé sur une structure de fichiers qui facilite la configuration, la personnalisation et l'écriture de tests, mais il y a eu des améliorations et des changements au fil du temps.

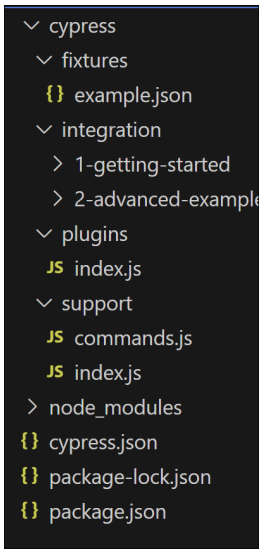
Avant la version 10, il utilisait une configuration standard qui pouvait être étendue ou personnalisée selon les besoins du projet. Comprendre l'approche historique de la configuration peut être utile pour maintenir des projets plus anciens ou migrer vers de nouvelles versions.

Dans les versions antérieures de Cypress (ici version 9), la configuration était largement gérée via le fichier cypress.json. Ce fichier était la source centrale pour tous les paramètres globaux, y compris les délais d'expiration, les URL de base, les répertoires de fichiers, etc. À cette époque, les utilisateurs se fiaient entièrement à ce fichier pour toutes les configurations de base et devaient souvent jongler avec des configurations supplémentaires pour les environnements de test spécifiques en utilisant des approches personnalisées.

En exécutant Cypress avec la commande :

```
npx cypress open
```

l'outil génère sa propre architecture :



Vous pouvez retrouver des nouveaux fichiers et répertoires :

– **cypress/fixtures/** : comme dans les versions plus récentes, le dossier fixtures est utilisé pour stocker des données statiques ou des exemples de données qui peuvent être utilisés lors de vos tests. Vous pourriez avoir des fichiers JSON, des images, ou d'autres contenus statiques qui peuvent être référencés dans vos tests. Ces fichiers sont chargés en utilisant `cy.fixture()` dans vos tests et sont utiles pour simuler des scénarios avec des données cohérentes.

- **cypress/support/commands.js** : il est utilisé pour ajouter des commandes personnalisées à Cypress et pour étendre ses fonctionnalités. Les commandes définies ici sont ajoutées au contexte global `cy` utilisé dans Cypress, permettant leur utilisation dans n'importe quel fichier de test.
- **cypress/support/index.js** : le fichier index.js dans le répertoire support est un point d'entrée pour personnaliser le comportement de Cypress avant l'exécution des tests. Il est utilisé pour importer des commandes personnalisées à partir de commands.js, importer des assertions ou des bibliothèques supplémentaires, et configurer des comportements qui doivent être initialisés avant l'exécution des suites de tests. Ce fichier peut également être utilisé pour gérer des événements globaux ou des comportements communs à tous vos tests.
- **cypress.json** : ce fichier permet aux développeurs de spécifier les paramètres globaux et la configuration du comportement des tests.
- **cypress/integration/** : ce dossier contient vos tests réels. Dans les versions antérieures, ce modèle était le même : vous écrivez vos tests dans ce dossier, et ils sont organisés en suites de tests, reflétant généralement la structure ou les fonctionnalités de votre application. Chaque fichier dans ce dossier est considéré comme une suite de tests distincte, et vous pouvez avoir autant de fichiers que nécessaire pour couvrir les différents aspects de votre application.

Si vous ne pouvez pas mettre à jour l'outil pour des raisons techniques de compatibilité, voici une explication de sa configuration.

Cypress nécessite un fichier de configuration pour spécifier comment exécuter les tests. Lors de la première exécution de Cypress est créé un fichier cypress.json à la racine de votre projet. Vous pouvez y spécifier des paramètres tels que l'URL de votre application, les navigateurs à utiliser, etc.

Exemple de contenu du fichier cypress.json :

```
{
  "baseUrl": "https://www.votre-site.com",
  "chromeWebSecurity": false,
  "defaultCommandTimeout": 10000,
  "viewportWidth": 1280,
  "viewportHeight": 720
}
```

– `"baseUrl": "https://www.votre-site.com"` : cette ligne définit l'URL de base de l'application que vous testez. Lorsque vous utilisez des commandes de navigation comme `cy.visit('/page')`, Cypress ajoutera automatiquement l'URL de base spécifiée ici pour accéder à la page souhaitée.
– `"chromeWebSecurity": false` : cette ligne indique à Cypress de désactiver la sécurité web de Chrome lors de l'exécution des tests. Cela permet de contourner les restrictions de sécurité liées aux requêtes cross-origin, ce qui peut être nécessaire pour certaines applications web qui effectuent des requêtes à partir de domaines différents.
– `"defaultCommandTimeout": 10000` : cette ligne définit le délai d'attente par défaut pour toutes les commandes Cypress en millisecondes. Dans ce cas, le délai d'attente est de 10 secondes (10000 millisecondes). Si une commande prend plus de temps que le délai spécifié, Cypress la considérera comme échouée.
– `"viewportWidth": 1280` : cette ligne définit la largeur par défaut du viewport du navigateur lors de l'exécution des tests. Cela peut être utile pour simuler différentes tailles d'écran et tester la réactivité de votre application. Le viewport contrôle la taille et l'orientation de l'écran de l'application web.
– `"viewportHeight": 720` : cette ligne définit la hauteur par défaut du viewport du navigateur lors de l'exécution des tests. Comme pour la largeur du viewport, cela peut être utile pour tester la réactivité de votre application à différentes tailles d'écran.

1.3 À partir de Cypress 10

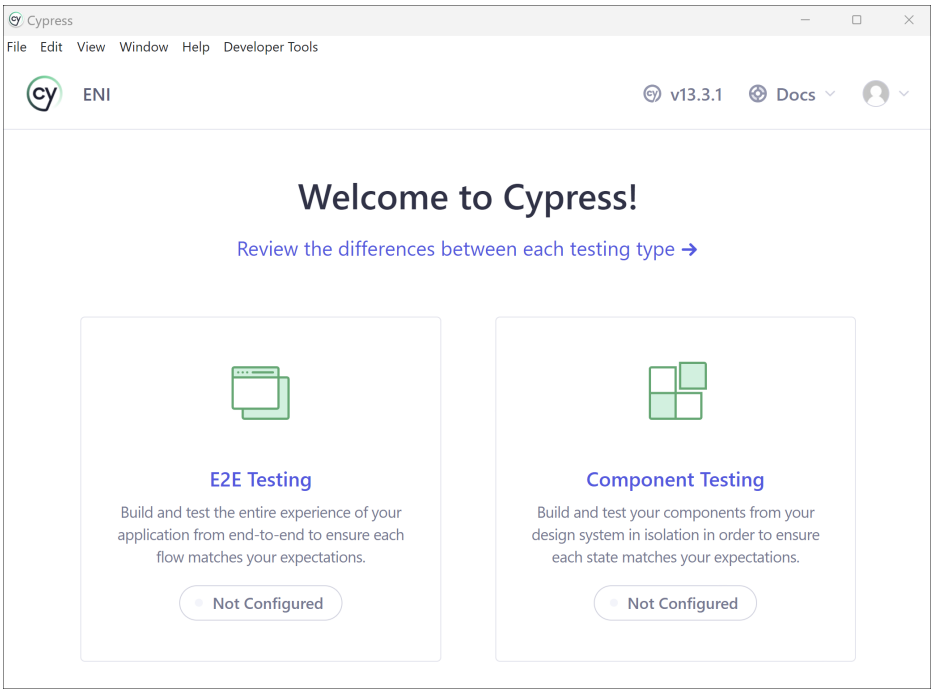
Avec la version 10, Cypress a introduit des améliorations significatives dans la gestion de la configuration. L'une des caractéristiques clés est la capacité de configurer des profils de test spécifiques. Plutôt que de se fier uniquement au fichier cypress.json, vous pouvez créer des fichiers de configuration dédiés pour différents environnements (par exemple, cypress.prod.json, cypress.dev.json), permettant une séparation claire et une gestion simplifiée. Cette fonctionnalité est particulièrement utile dans les systèmes CI/CD où différents paramètres sont nécessaires pour les différentes étapes du pipeline.

Dans ces nouvelles versions, Cypress a introduit une interface utilisateur mise à jour qui inclut des options séparées pour la configuration des tests de bout en bout (E2E) et les tests de composants. Cette distinction a été faite pour offrir une meilleure expérience de configuration et de gestion des différents types de tests.

En exécutant la commande suivante :

```
npx cypress open
```

une nouvelle fenêtre s'est ouverte et ressemble à la capture d'écran ci-dessous :



Vous avez donc deux choix possibles à ce niveau :

– **E2E Testing**, pour les tests end-to-end. Comme leur nom l'indique, ces tests sont destinés à tester le comportement d'une application du début à la fin, simulant le parcours complet d'un utilisateur à travers l'application, du navigateur web au backend en passant par les intégrations avec les API et les services tiers. Dans le contexte de Cypress, les tests E2E sont utilisés pour vérifier le bon fonctionnement global de l'application, en simulant les actions d'un utilisateur réel, telles que la navigation entre les pages, le remplissage de formulaires, le clic sur des boutons, etc. Ces tests vérifient que toutes les parties de l'application fonctionnent correctement ensemble.
– **Component Testing**, pour les tests composants. Les tests composants se concentrent sur des parties spécifiques d'une application, telles que des composants ou des modules individuels comme des boutons, des champs de saisie, des listes, etc. Ces tests permettent de s'assurer que chaque composant fonctionne correctement et produit les résultats attendus indépendamment du reste de l'application.

Comme vous pouvez le voir, le choix est important ; chacune de ces options comporte des avantages et des inconvénients :

	Test E2E	**Tests composants**
Avantages	Les tests correspondent à l'expérience utilisateur. Peut être écrit par des développeurs ou des équipes d'assurance qualité. Peut également être utilisé pour les tests d'intégration.	Plus facile de tester les composants isolément. Rapide et fiable. Facile de mettre en place des scénarios spécifiques dans les tests.
Inconvénients	Plus difficile à configurer, à exécuter et à entretenir. A souvent besoin d'une configuration CI complexe. Tester certains scénarios nécessite plus de configuration.	Ne garantit pas la qualité globale de l'application. Ne fait pas appel à des API/ services externes. Généralement écrit par les développeurs travaillant sur le composant.