Questions générales sur Cypress

1. **Qu'est-ce que Cypress ?**
   Cypress est un framework de tests end-to-end moderne conçu pour tester des applications web. Il permet de vérifier le comportement des interfaces utilisateur, de déboguer et de valider des fonctionnalités interactives directement dans le navigateur.
2. **Comment Cypress se différencie-t-il de Selenium ?**
   - **Architecture :** Cypress fonctionne directement dans le même processus que l'application, contrairement à Selenium qui utilise un serveur intermédiaire.
   - **Configuration :** Cypress est plus simple à configurer et ne nécessite pas d'installation séparée de WebDriver.
   - **Débogage :** Cypress fournit des outils intégrés, comme les snapshots et un excellent support des DevTools.
3. **Quels types de tests peut-on effectuer avec Cypress ?**
   - Tests end-to-end.
   - Tests d'intégration.
   - Tests unitaires (moins courants mais possibles).

Configuration et installation

4. **Comment installer Cypress dans un projet ?**

   ```
   css
   Copier le code
   npm install cypress --save-dev
   ```

5. **Comment démarrer Cypress une fois installé ?**

   ```
   arduino
   Copier le code
   npx cypress open
   ```

   Cela ouvre l'interface graphique de Cypress où vous pouvez exécuter vos tests.

6. **Où Cypress stocke-t-il les fichiers de configuration ?**
   Dans le fichier `cypress.config.js` (ou `.ts` si TypeScript est utilisé).

Fonctionnalités et commandes principales

7. **Quelles sont les commandes les plus courantes dans Cypress ?**
   - `cy.visit()` : pour naviguer vers une URL.
   - `cy.get()` : pour sélectionner un élément DOM.
   - `cy.contains()` : pour trouver un élément contenant un texte spécifique.
   - `cy.click()` : pour cliquer sur un élément.
   - `cy.type()` : pour entrer du texte dans un champ.
8. **Comment vérifier qu'un élément est visible sur la page ?**

```
javascript
Copier le code
cy.get('selector').should('be.visible');
```

9. **Comment gérer les attentes (wait) dans Cypress ?**
   Cypress utilise des attentes automatiques. Cependant, pour des cas spécifiques, on peut utiliser :

   ```
   javascript
   Copier le code
   cy.wait(2000); // attendre 2 secondes (non recommandé)
   ```

   Préférez les attentes dynamiques avec des assertions.

Tests avancés

10. **Comment Cypress gère-t-il les API ?**
    Utilisez `cy.request()` pour interagir avec les API :

```
javascript
Copier le code
cy.request('GET', '/api/resource').then((response) => {
    expect(response.status).to.eq(200);
});
```

11. **Comment Cypress gère-t-il les tests en cas de multiples onglets ou iframes ?**
    Cypress ne supporte pas directement les tests sur plusieurs onglets. Cependant, vous pouvez interagir avec les iframes en utilisant une combinaison de `cy.get()` et de commandes spécifiques pour cibler les contenus.
12. **Comment effectuer des tests de mock ou stub dans Cypress ?**
    Utilisez `cy.intercept()` pour intercepter et simuler des requêtes réseau :

```
javascript
Copier le code
cy.intercept('GET', '/api/resource', { fixture: 'mockData.json' });
```

Gestion des erreurs et débogage

13. **Que faire si un test échoue à cause d'une erreur imprévue ?**

- Utilisez les snapshots fournis dans l'interface graphique pour voir l'état du DOM.
- Ajoutez des logs avec `cy.log()` pour mieux comprendre le contexte.
- Activez les DevTools pour inspecter les erreurs dans le navigateur.

14. **Comment ignorer temporairement un test ?**
    Utilisez `.skip` :

```
javascript
Copier le code
it.skip('Test ignoré temporairement', () => {
```

```
    // test non exécuté
});
```

15. **Comment relancer automatiquement un test échoué ?**
    Cypress le fait automatiquement grâce à ses attentes intégrées.

Questions ouvertes pour évaluer les connaissances

16. **Quels sont les avantages et limites de Cypress selon vous ?**

- **Avantages :** Facilité d'utilisation, intégration rapide, excellent débogage, riche en fonctionnalités.
- **Limites :** Pas de support direct pour plusieurs onglets, difficile à utiliser pour des tests mobiles natifs.

17. **Comment organisez-vous vos tests avec Cypress ?**

- Séparer les tests en catégories (unitaires, d'intégration, E2E).
- Utiliser des hooks (`before`, `after`, `beforeEach`, `afterEach`) pour configurer l'environnement de test.
- Centraliser les sélecteurs et fonctions communes dans des fichiers utilitaires.

18. **Que sont les fixtures dans Cypress ?**
    Les **fixtures** sont des fichiers JSON utilisés pour fournir des données statiques ou simulées à vos tests.