Tuto Test React avec Jest et Cypress

Les test serve à tester le code avant la mise en production

Test Unitaire

Il n’y a pas besoin d’installer de dépendance supplémentaire.

Les tests unitaires en React serve a testé un composant. Ce sont les tests les plus basique.

Etape 0 :

Faite un dossier \_\_test\_\_ dans le même dossier du composant et faite un fichier TodoItem.test.js

Etape 1 :

Importer les dépendances :



Ici on import render, fireEvent, screen, act ainsi que le composant TodoItem.

Render permet de rendre le composant, fireEvent d’exécuter des évènements sur le composant, screen permet d’accéder au composant rendu et act de faire des events et d’attendre que le composant et fini de ce refresh.

Etape 2 :

BeforeEach et after each  et Mock :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ici on fait 3 mock, une constance contenant une liste d’objets représentant une todo.

MockUpdateTodo est une function mock, elle permet de verifié si c’est function sont appelé plus tard. Il y en a aussi une pour MockDeleteTodo.

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Ici le beforeEach nous permet de faire quelque chose avant tous les tests.

On va donc render le composant en lui donnant toute nos mock comme props.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Ici le afterEach s’exécute après chaque test. Il va clear toutes les mocks pour nettoyer la mémoire.

Etape 3 :

Faire un test

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ce test vérifie que tous les éléments sélectionné existe bien dans le composants avec la ligne expect(elements).toBeDefined(). Pour récupéré c’est éléments on utilise screen.getByTestId(), pour cela il faut donnée un data-testid au éléments souhaiter dans le composant.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Maintenant dans le terminal faite npm test. Si vous avez bien tous suivit il devrait vous dire que le test est passé.

Etape 4 :

Test plus complexe avec les mock

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Ce test peut faire peur mais il est très simple.

Nous créont une fausse réponse avec comme content : « Test 1 » ainsi que done a false et edit a true

Puis nous créons une fonction espionne fetch elle va remplacer nos appelle au back dans le composant en renvoyant la mockResponse.

On prend le boutonEdit. On crée un block act qui permet d’attendre que le bouton soit cliqué et que le composant ce re-rende.

Ensuite on regarde si la function fetch a été appelé avec comme arguments la mock response ainsi que les arguments de base d’appelle au back.

Finalement on vérifie que la function updateTodo est été appelé avec comme arguments la mock response.

Pour cela lancer votre back-End et faite npm test normalement tous doit passer

Test Integration

Les tests d’intégration sont très importants il permette de vérifier que plusieurs composant marche bien ensemble.

Etape 0 :

Nous allons utiliser cypress pour cela faite npm i cypress eslint-plugin-cypress -D

Puis npx cypress open

Enfin clické sur E2E test puis crée dans le dossier cypress un fichier AppAddtodoInt.cy.js

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Etape 1 :

Dans cypress nous utilisant describe et it, describe décrit une suite de test et it est la pour un seule test.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Ce test est simple. Function describe puis beforeEach ou au lieu de rendre le composant comme sur jest nous allons visiter le front end (vous devez donc le lancer).

Ensuite dans un it nous faisons un test. Nous créons un texte. Puis ont récupéré l’input de addtodo. On le focus puis le clear avec. focus() et .clear(). Nous somme donc sur qu’il est vide. Ensuite avec type(), on écrit dedans la valeur. Puis on le blur avec blur().

Après on cherche le button add avec cy.contains(« Add »). On click dessus avec click().

Ensuite on récupéré tous les li enfant de ul avec cy.get(« ul > li » ) (> veut dire parent de (ex : ul parent de li)) . Et on vérifie que ça longueur et bien de 1 donc test a été ajouté.

Le deuxième test it fait la même chose mais sans ajouter de valeur. La longueur doit être de 4 pour les 3 ensuite todo et le test ajouter précédemment.

L’avantage de cypress est son côté graphique vous pouvez voir en direct le test

TEST E2E

Les tests E2E (end to end) test toutes l’application ils sont là pour simuler une possible interaction complète utilisateurs

Etape 0 :

Crée un fichier E2E.cy.js

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Etape 1 :

Crée des tests, avec toute ce que l’on a vu avant avec cypress vous pouvez tous tester facilement comme ci suit.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Ce sont les mêmes test que ce d’integration sauf le qu’ici au lieu de regarder la longueur on regarde si le textes et présent quelque part dans les éléments sélectionné

Cy.get(« ul> li > span) récupére tous les span enfant de ul et li

.contains(MockTest) contients le mockText

.should(« not.exist ») ne doit pas exister

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ici on récupére le quatrième button Edit et on click dessus (cela crée un input)

On selectionne l’input crée et on écrit un text modifié dedans.

Puis on click sur le button save et enfin vérifie que la valeurs a bien changé

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ici on sélectionne encore le quatrième button mais cette fois si delete.

On verifie que la liste de li ne contient plus que 3 elements avec should(« have.length », 3)

Conclusion

Les tests sont importants pour voir qu’une app a un comportement conforme et vérifié des bugs potentiel c’est une étable indispensable avant de la mettre en production ou même avant de le push sur un git communs pour être sur de ne pas envoyer des bugs