Tests pour React

Jest

- le plus populaire
- maintenu par Facebook
- utilisé par Facebook, Airbnb, etc
- recommandé par React
- fonctionne avec Typescript
- framework de test par défaut pour NestJs
- très rapide
- tests snapshot, parallel testing et de méthode asynchrone
- plus simple car moins de bibliothèques ou autres outils à installer
- nécessite très souvent Enzyme

Enzyme

Test de composants React.

- un des frameworks les plus utilisés
- combiné avec d'autres frameworks tels que Jest, Chai ou Mocha
- Enzyme permet d'afficher des composants, accéder à des éléments, rechercher des éléments, interagir avec des éléments et simuler des événements. Chai ou Jest peuvent être utilisés pour faire les assertions
- Indique précisément la source des erreurs

Mocha

- Rapports de couverture
- Beaucoup moins rapide que Jest
- Front-end et Back-end
- Tests asynchrones
- Nécessite Chai (bibliothèque d'assertions bdd / tdd pour fonctions) ou Enzyme (pour composants) pour utiliser des assertions

Cypress

Framework de test de bout en bout.

- très rapide
- seulement front-end
- aucun framework de test supplémentaire nécessaire
- facile à utiliser avec des outils de CI
- exécution des tests dans le vrai navigateur ou en ligne de commande utilisation des outils de développement de navigateur côte à côte
- panneau de contrôle pour contrôler l'état des tests
- contrôle du trafic réseau sans toucher au serveur pour tester les cas extrêmes

React testing library

- recommandé par React
- tester facilement les composants et simuler le comportement de l'utilisateur
- semblable à enzyme

- on ne peut pas accéder aux activités internes des composants (ex: leurs états)

Tests pour NestJs

Jest

- intégré directement à NestJs
- tests unitaires
- tests end to end

Introduction: https://docs.nestjs.com/fundamentals/testing#testing
Documentation : https://jestjs.io/fr/docs/getting-started

Supertest

- intégré directement à NestJs
- à utiliser avec Jest
- librairie qui permet de simuler des requêtes HTTP

Documentation : https://github.com/visionmedia/supertest

Vocabulaire

TDD (Test Driven Development)

Écrire les tests unitaires d'une fonction avant son contenu.

- code bien segmenté donc plus facile à tester avec des tests unitaires
- connaître les résultats attendus permet de partir dans la bonne direction dès le démarrage des développements

Principe :

1. Ecrire le test unitaire
2. Lancer le test et vérifier qu'il échoue (car classe ou fonction par encore implémentée)
3. Ecrire la classe ou la fonction à tester avec le minimum pour faire marcher le test
4. Lancer le test et vérifier qu'il fonctionne
5. Terminer l'implémentation de la classe ou fonction
6. Vérifier que le test fonctionne toujours (non-régression)

BDD (Behavior Driven Development)

Créer des tests fonctionnels avec un langage naturel compris de tous.

- tester chaque fonction de code ne permet pas de valider des comportements complets
- guider les développements par rapport aux comportements attendus
- rapprocher les équipes techniques et fonctionnelles (méthode agile)
- tests écrits avec un langage naturel appelé Gherkin (given, when, the)

Exemple (traduit) :
Etant donné que je suis le client sur la fiche produit
Quand je clique sur ajouter au panier sur le produit d'identifiant "255"
Et que le stock est à "3"
Alors le produit s'ajoute au panier.

E2E (End to End)

Tester l'entièreté du logiciel du début à la fin pour vérifier que le parcours utilisateur fonctionne correctement.

Parallel testing

Exécuter simultanément plusieurs scripts d'automatisation des tests, chaque script consommant des ressources différentes. Permet de vérifier plusieurs composants applicatifs sur des ordinateurs séparés, ou exécuter des scripts sur différents appareils et navigateurs en même temps.

Tests snapshot

Afficher un composant UI, prendre un screenshot, et le comparer à un screenshot de référence stocké avec le test.