**Jest Documentation**

1. **Definition**

Jest est un framework de test JavaScript développé par Facebook. Il est principalement utilisé pour tester des applications et des bibliothèques écrites avec React, mais il peut également être utilisé pour d'autres frameworks JavaScript tels que Angular et Vue.js.

1. **Premiers pas**

**Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement**

"scripts": {

    "test":"jest \_\_tests\_\_ --coverage"

  },

**npm run test == >Will run all test in \_\_tests\_\_ folder et nous donne un coverage**

1. **Describe & Test**

En Jest les fonctions **`describe`** et **`test`** sont utilizer pour structurer et écrire des suites de tests.

La fonction "test" de Jest est utilisée pour définir des cas de test unitaires et vérifier le comportement attendu d'un code. "Describe" est utilisé pour regrouper des cas de test liés et les organiser de manière cohérente.

1. **Utilisation des comparateurs**

Jest propose plusieurs comparateurs couramment utilisés pour effectuer des assertions dans les tests. Voici quelques-uns des comparateurs les plus utilisés en Jest :

1. **expect().toBe(value)** : Vérifie si la valeur attendue est strictement égale (**===**) à la valeur reçue.
2. **expect().toEqual(value)** : Vérifie si la valeur attendue est équivalente à la valeur reçue en utilisant une comparaison récursive des propriétés pour les objets et les tableaux.
3. **expect().toBeNull()** : Vérifie si la valeur reçue est **null**.
4. **expect().toBeDefined()** : Vérifie si la valeur reçue est définie (**!== undefined**).
5. **expect().toBeUndefined()** : Vérifie si la valeur reçue est indéfinie (**=== undefined**).
6. **expect().toBeTruthy()** : Vérifie si la valeur reçue est évaluée à **true**.
7. **expect().toBeFalsy()** : Vérifie si la valeur reçue est évaluée à **false**.
8. **expect().toContain(item)** : Vérifie si un tableau ou une chaîne de caractères contient un élément spécifique.
9. **expect().toHaveLength(length)** : Vérifie si un tableau ou une chaîne de caractères a une longueur spécifique.
10. **expect().toHaveProperty(propertyPath, value?)** : Vérifie si un objet a une propriété spécifique avec une valeur facultative.
11. **expect().toMatch(regexpOrString)** : Vérifie si une chaîne de caractères correspond à une expression régulière ou à une sous-chaîne.
12. **expect().toThrow(error?)** : Vérifie si une fonction lance une exception, avec une option pour vérifier le type ou le message de l'exception.
13. **expect().not.toBe()**Vous pouvez également tester pour le contraire d'un comparateur en utilisant not:

Ces comparateurs vous permettent de formuler des assertions précises dans vos tests pour vérifier le comportement attendu de votre code. Jest offre une large gamme de comparateurs pour couvrir différents cas de test et garantir la fiabilité de vos tests.

# **Tester du code asynchrone**

Il est fréquent en JavaScript que le code s'exécute de manière asynchrone. Lorsque vous avez du code qui s’exécute de façon asynchrone, Jest doit savoir quand le code qu'il teste est terminé, avant de passer à un autre test. Jest a plusieurs façons de gérer cela.

Pour tester du code asynchrone en utilisant Jest ,vous pouvez utilisez les fonctionaliter de gestion des promesses ou les fonctions asynchrone (`**async/await**`) , ou **done()** quand on utilise une fonction de rappel asynchrone .

* 1. Utiliser les promesses et les méthodes `**.then()`** et `**.catch()`**

Utilisation de la fonction then : Si votre code retourne une promesse, vous pouvez utiliser la fonction then pour définir les assertions à exécuter lorsque la promesse est résolue.

// Test avec then

test('Test asynchrone avec then', () => {

  return fetchData().then((result) => {

    expect(result).toBe('Données récupérées');

  });

});

* 1. Utilisation de la fonction **async/await** :

Utilisation de la fonction **async/await** : Vous pouvez marquer votre fonction de test comme **async** et utiliser le mot-clé **await** pour attendre la résolution de la promesse ou l'exécution d'une fonction asynchrone.

// Test avec async/await

test('Test asynchrone avec async/await', async () => {

  const result = await fetchData();

  expect(result).toBe('Données récupérées');

});

* 1. Utilisation de la fonction **expect** avec **resolves** :

Utilisation de la fonction expect avec resolves : Vous pouvez utiliser la fonction expect de Jest avec le modificateur resolves pour vérifier que la promesse est résolue avec une valeur spécifique.

// Test avec expect(resolves)

test('Test asynchrone avec expect(resolves)', () => {

  return expect(fetchData()).resolves.toBe('Données récupérées');

});

* 1. Utilisation de **done()**

Lorsqu'on utilise **done** dans Jest, on indique à Jest que le test asynchrone est terminé et qu'il peut continuer à exécuter les autres tests. **done** est généralement passé en tant que paramètre à une fonction de rappel asynchrone, et Jest attendra que **done** soit appelé pour marquer le test comme terminé.

// Fonction asynchrone à tester

function fetchData(callback) {

  // Supposons que fetchData effectue une requête asynchrone vers une API

  // et appelle le callback avec les données récupérées

  setTimeout(() => {

    const data = ['Donnée 1', 'Donnée 2', 'Donnée 3'];

    callback(data);

  }, 1000);

}

// Test avec done

test('Test asynchrone avec done', (done) => {

  fetchData((result) => {

    expect(result).toContain('Donnée 2');

    done(); // Appel de done() pour indiquer la fin du test asynchrone

  });

});