**como hacer los test a las promise?**

Para realizar pruebas a una función que devuelve una Promise en Jest, se puede utilizar la sintaxis de pruebas asíncronas proporcionada por Jest. A continuación, te explicaré los pasos para hacerlo:

Paso 1: Crear la función a probar

En este ejemplo, vamos a crear una función que devuelve una Promise que se resuelve después de un cierto tiempo:

javascript

function fetchData() {

return new Promise((resolve, reject) => {

setTimeout(() => {

resolve('Datos cargados correctamente');

}, 1000);

});

}

Paso 2: Crear la prueba

Para probar esta función, vamos a utilizar la sintaxis de pruebas asíncronas proporcionada por Jest. Podemos hacer esto utilizando la función test de Jest y pasando una función asíncrona como argumento:

javascript

test('fetchData resuelve con datos cargados correctamente', async () => {

const data = await fetchData();

expect(data).toBe('Datos cargados correctamente');

});

La función async indica a Jest que esta prueba contiene código asíncrono y debe esperar a que se resuelva la Promise antes de continuar.

En esta prueba, llamamos a la función fetchData y esperamos a que la Promise se resuelva utilizando la palabra clave await. A continuación, comprobamos que los datos devueltos por la Promise sean los esperados utilizando la función expect.

Paso 3: Ejecutar la prueba

Finalmente, podemos ejecutar esta prueba utilizando el comando jest. Jest detectará automáticamente esta prueba y la ejecutará. Si la prueba pasa, Jest mostrará un mensaje indicando que la prueba se ha pasado correctamente. Si la prueba falla, Jest mostrará un mensaje indicando dónde ha fallado la prueba.

Conclusión

Jest proporciona una sintaxis clara y fácil de entender para probar funciones que devuelven Promise. Con Jest, podemos asegurarnos de que nuestras funciones asíncronas funcionan correctamente y mantener la calidad de nuestro proyecto.

**como hacer los test a una funcion callback?**

Para realizar pruebas a una función que recibe un callback en Jest, podemos seguir los siguientes pasos:

Paso 1: Crear la función a probar

En este ejemplo, crearemos una función que recibe un número y un callback. Si el número es mayor que 10, llamará al callback con el mensaje "Mayor de 10", de lo contrario, llamará al callback con el mensaje "Menor de 10":

javascript

function callbackExample(num, callback) {

if (num > 10) {

callback("Mayor de 10");

} else {

callback("Menor de 10");

}

}

Paso 2: Crear la prueba

Para probar esta función, podemos crear una función mock que simule el comportamiento del callback. Utilizaremos la función jest.fn() para crear la función mock y luego la pasaremos como argumento a nuestra función a probar. Luego, podemos verificar que el callback se llamó correctamente utilizando la función expect:

javascript

test('callbackExample llama al callback con el mensaje correcto', () => {

const mockCallback = jest.fn();

callbackExample(15, mockCallback);

expect(mockCallback).toHaveBeenCalledWith("Mayor de 10");

});

En esta prueba, creamos la función mock utilizando jest.fn(), luego llamamos a la función callbackExample con el número 15 y la función mock. Finalmente, verificamos que el callback se llamó con el mensaje "Mayor de 10" utilizando expect y la función toHaveBeenCalledWith.

Paso 3: Ejecutar la prueba

Podemos ejecutar esta prueba utilizando el comando jest. Jest detectará automáticamente esta prueba y la ejecutará. Si la prueba pasa, Jest mostrará un mensaje indicando que la prueba se ha pasado correctamente. Si la prueba falla, Jest mostrará un mensaje indicando dónde ha fallado la prueba.

Conclusión

Jest proporciona una sintaxis clara y fácil de entender para probar funciones que reciben callbacks. Con Jest, podemos asegurarnos de que nuestras funciones callback funcionen correctamente y mantener la calidad de nuestro proyecto.

**como hacer un test a una promise que ha resuelto correctamente el parametro dado**

Para realizar un test a una Promise que ha resuelto correctamente con un parámetro dado en Jest, podemos seguir los siguientes pasos:

Paso 1: Crear la función a probar

En este ejemplo, crearemos una función que devuelve una Promise que se resuelve con el parámetro que le pasemos:

javascript

function myPromise(param) {

return new Promise((resolve, reject) => {

resolve(param);

});

}

Paso 2: Crear la prueba

Para probar esta función, podemos utilizar la función expect para verificar que el parámetro que recibimos es el que esperábamos. Para hacer esto, debemos esperar a que la Promise se resuelva utilizando la palabra clave await dentro de una función asíncrona:

javascript

test('myPromise resuelve correctamente con el parametro dado', async () => {

const param = 'Hola';

const result = await myPromise(param);

expect(result).toBe(param);

});

En esta prueba, creamos un parámetro llamado param y lo pasamos a la función myPromise. Luego, esperamos a que la Promise se resuelva y almacenamos el resultado en la variable result. Finalmente, verificamos que el resultado sea igual al parámetro utilizando la función expect y la propiedad toBe.

Paso 3: Ejecutar la prueba

Podemos ejecutar esta prueba utilizando el comando jest. Jest detectará automáticamente esta prueba y la ejecutará. Si la prueba pasa, Jest mostrará un mensaje indicando que la prueba se ha pasado correctamente. Si la prueba falla, Jest mostrará un mensaje indicando dónde ha fallado la prueba.

Conclusión

Jest proporciona una sintaxis clara y fácil de entender para probar funciones que devuelven Promise. Con Jest, podemos asegurarnos de que nuestras funciones asíncronas funcionen correctamente y mantener la calidad de nuestro proyecto. En este caso, pudimos verificar que la Promise se resolvió correctamente con el parámetro que le pasamos.

**Que es jest.spyOn?**

jest.spyOn es una función proporcionada por Jest que permite crear un "spy" o "espía" de una función existente. Al crear un espía de una función, podemos verificar cómo se llama la función, con qué parámetros se llama y qué valor devuelve. También podemos modificar el comportamiento de la función espía para que devuelva valores específicos o lanzar errores en determinadas situaciones.

La sintaxis básica de jest.spyOn es la siguiente:

javascript

const object = {

method() {}

};

const spy = jest.spyOn(object, 'method');

En este ejemplo, creamos un objeto llamado object que tiene un método llamado method. Luego, creamos un espía de este método utilizando jest.spyOn y lo almacenamos en una variable llamada spy. Podemos utilizar esta variable spy para verificar cómo se llama la función, con qué parámetros se llama y qué valor devuelve.

Por ejemplo, podemos verificar cómo se llama la función espía utilizando la función toHaveBeenCalled:

javascript

object.method();

expect(spy).toHaveBeenCalled();

En este ejemplo, llamamos a la función method del objeto object y luego verificamos que la función espía spy haya sido llamada utilizando la función toHaveBeenCalled.

También podemos modificar el comportamiento de la función espía utilizando la función mockImplementation:

javascript

spy.mockImplementation(() => 5);

expect(object.method()).toEqual(5);

En este ejemplo, modificamos el comportamiento de la función espía spy para que siempre devuelva el número 5 utilizando la función mockImplementation. Luego, llamamos a la función method del objeto object y verificamos que devuelva el número 5 utilizando la función toEqual.

En resumen, jest.spyOn es una herramienta poderosa para probar funciones existentes y verificar su comportamiento. Podemos crear espías de funciones, verificar cómo se llaman, con qué parámetros se llaman y qué valor devuelven, y modificar su comportamiento para adaptarse a diferentes situaciones de prueba.