Jest — это популярная тестовая среда для JavaScript, разработанная Facebook. Она используется для тестирования приложений и библиотек, особенно тех, которые написаны с использованием React. Jest предлагает множество функций, таких как изоляция тестов, мокирование, асинхронное тестирование и многое другое. Давайте рассмотрим основные аспекты написания тестов с использованием Jest.

**Установка Jest**

npm install --save-dev jest

**Установка babel**

npm install --save-dev babel-jest @babel/core @babel/preset-env

**Создание файла babel.config.js**

module.exports = {

presets: [['@babel/preset-env', {targets: {node: 'current'}}]],

};

**Создание файла jest.config.js**

module.exports = {

transform: {

'\\.js$': 'babel-jest'

}

};

**В package.json добавить скрипт для запуска тестов:**

{

  "scripts": {

    "test": "jest"

  }

}

**Установка babel**

npm install --save-dev eslint-plugin-jest (Чтобы eslint не выдавал ошибки jest)

В файле eslint.js

"env": {

"jest/globals": true

}

{

"plugins": ["jest"]

}

"extends": ["eslint:recommended"],

**Добавляем авто дополнение методов и функций для jest**

npm install --save-dev @types/jest

**Основные концепции Jest (Создаем файл name.test.js)**

1. Тестовые файлы: По умолчанию Jest ищет файлы с расширением .test.js или .spec.js, а также файлы внутри папки \_\_tests\_\_.

**2. Тестовые функции: Основные функции, используемые в Jest:**

   • test(name, fn): Определяет тест с названием name и функцией fn.

   • describe(name, fn): Группирует несколько тестов под одним названием.

   • expect(value): Используется для проверки значений.

3. Матчеры (методы утверждений): Jest предоставляет богатый набор матчеров для проверки значений:

1. toBe(value): Проверяет, что значение точно соответствует ожидаемому. Используется для примитивных типов данных (числа, строки, булевы значения).

expect(2 + 2).toBe(4);

2. toEqual(value): Проверяет, что значение эквивалентно ожидаемому. Используется для сравнения объектов и массивов.

expect({ name: 'John' }).toEqual({ name: 'John' });

3. toStrictEqual(value): Похоже на toEqual, но также проверяет наличие и тип всех свойств объекта, включая undefined.

expect({ name: 'John', age: undefined }).toStrictEqual({ name: 'John', age: undefined });

4. toBeNull(): Проверяет, что значение является null.

expect(null).toBeNull();

5. toBeUndefined(): Проверяет, что значение является undefined.

expect(undefined).toBeUndefined();

6. toBeDefined(): Проверяет, что значение не является undefined.

expect(4).toBeDefined();

7. toBeTruthy(): Проверяет, что значение является истинным в логическом контексте.

expect(true).toBeTruthy();

8. toBeFalsy(): Проверяет, что значение является ложным в логическом контексте.

expect(false).toBeFalsy();

9. toBeGreaterThan(number): Проверяет, что значение больше указанного числа.

expect(10).toBeGreaterThan(5);

10. toBeGreaterThanOrEqual(number): Проверяет, что значение больше или равно указанному числу.

expect(10).toBeGreaterThanOrEqual(10);

11. toBeLessThan(number): Проверяет, что значение меньше указанного числа.

expect(5).toBeLessThan(10);

12. toBeLessThanOrEqual(number): Проверяет, что значение меньше или равно указанному числу.

expect(10).toBeLessThanOrEqual(10)

13. toContain(item): Проверяет, что массив или строка содержит указанный элемент.

expect(['apple', 'banana']).toContain('banana');

14. toHaveLength(number): Проверяет длину массива или строки.

expect([1, 2, 3]).toHaveLength(3);

15. toMatch(regex): Проверяет, что строка соответствует регулярному выражению.

expect('Hello World').toMatch(/World/);

16. toThrow(error?): Проверяет, что функция выбрасывает ошибку при выполнении.

function throwError() {

throw new Error('Error!');

}

expect(() => throwError()).toThrow('Error!');

**Пример простого теста**

Рассмотрим простой пример функции и теста для нее:

// utils.js

function sum(a, b) {

  return a + b;

}

module.exports = sum;

// utils.test.js

const sum = require('./utils');

test('adds 1 + 2 to equal 3', () => {

  expect(sum(1, 2)).toBe(3);

});

**Группировка тестов**

Вы можете использовать describe для группировки связанных тестов:

describe('Math operations', () => {

  test('adds 1 + 2 to equal 3', () => {

    expect(sum(1, 2)).toBe(3);

  });

  test('adds 2 + 2 to equal 4', () => {

    expect(sum(2, 2)).toBe(4);

  });

});

**Асинхронное тестирование**

Jest поддерживает асинхронные тесты с использованием колбэков, промисов и async/await.

Пример с async/await:

// asyncFunction.js

async function fetchData() {

  return 'peanut butter';

}

module.exports = fetchData;

// asyncFunction.test.js

const fetchData = require('./asyncFunction');

test('the data is peanut butter', async () => {

  const data = await fetchData();

  expect(data).toBe('peanut butter');

});

Мокирование

Jest позволяет мокировать модули и функции для изоляции тестов.

Пример мокирования функции:

const myModule = {

  method: jest.fn(() => 'default value'),

};

test('method returns default value', () => {

  expect(myModule.method()).toBe('default value');

  expect(myModule.method).toHaveBeenCalled();

});

**Жизненный цикл тестов**

Jest предоставляет хуки для выполнения кода до и после тестов:

• beforeEach(fn): Выполняется перед каждым тестом.

• afterEach(fn): Выполняется после каждого теста.

• beforeAll(fn): Выполняется один раз перед всеми тестами.

• afterAll(fn): Выполняется один раз после всех тестов.

**Заключение**

Jest — это мощный инструмент для тестирования JavaScript-кода, который предоставляет множество функций для упрощения процесса написания и выполнения тестов. Он легко интегрируется с другими инструментами разработки и поддерживает как синхронное, так и асинхронное тестирование.