1. Jest

## 1.1 Jest là gì?

Là công cụ được xây dựng bởi Facebook, một công cụ mạnh mẽ hỗ trợ cho việc viết unit test trên các ứng dụng Javascript.

## 1.2 Cài đặt

Có thể dùng command npm hoặc yarn để cài đặt

npm install --save-dev jest

Hoặc

yarn add --dev jest

Thêm vào đoạn script tại package.json

"scripts": {

"test": "jest"

}

Tạo file test với dạng \*.test.js

### 1.3 Demo

// file DemoSimple.js

const add = (a, b) => a + b;

module.exports = add;

// file .test.js

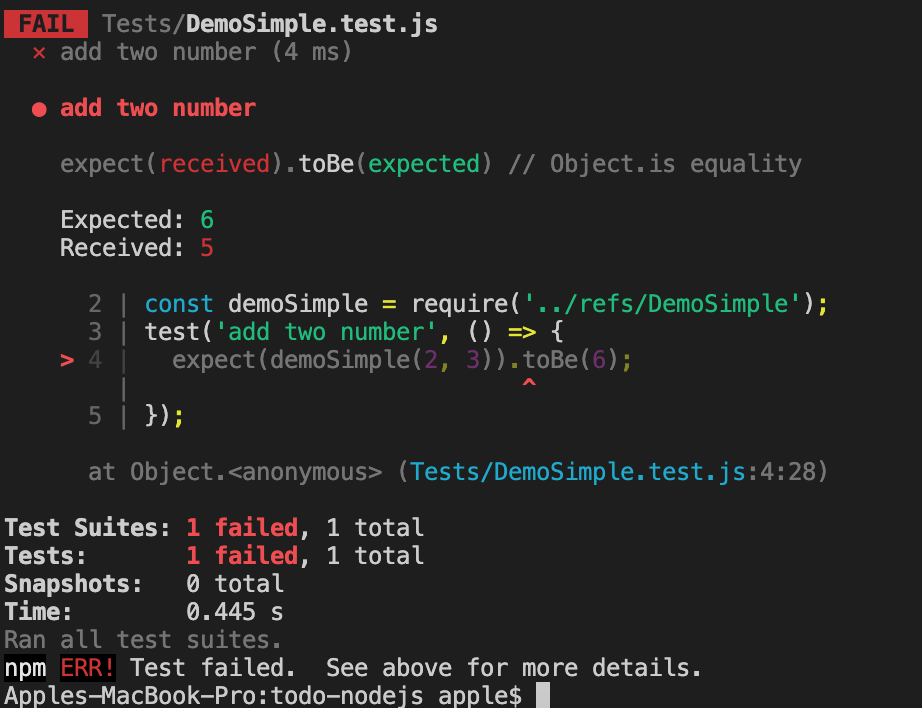
const demoSimple = require('../refs/DemoSimple');

test('add two number', () => {

expect(demoSimple(2, 3)).toBe(6);

});

Chạy lệnh npm test được kết quả



## 2. Các toán từ(matchers) trong Jest

### 2.1 toBe

Sử dụng toBe để kiểm tra các toán tử trả về

expect(add(2, 2)).toBe(4);

### 2.2 toEqual

So sánh kết quả trả về dạng object

const add = (a, b) => ({ a, b });

test('check obj', () => {

expect(add(2, 2)).toEqual({ a: 2, b: 2 });

});

### 2.3 Truthiness

Phân biệt giữa undefined, null và false...

* toBeNull chỉ đối sánh với null
* toBeUndefined chỉ đối sánh không xác định
* toBeDefined đối lập với toBeUndefined
* toBeTruthy khớp với bất kỳ thứ gì mà câu lệnh if coi là true
* toBeFalsy khớp với bất kỳ thứ gì mà câu lệnh if coi là sai

test('null', () => {

const n = null;

expect(n).toBeNull();

expect(n).toBeDefined();

expect(n).not.toBeUndefined();

expect(n).not.toBeTruthy();

expect(n).toBeFalsy();

});

test('zero', () => {

const z = 0;

expect(z).not.toBeNull();

expect(z).toBeDefined();

expect(z).not.toBeUndefined();

expect(z).not.toBeTruthy();

expect(z).toBeFalsy();

});

### 2.4 Number

Có các biểu thức so sánh như sau:

test('test matching number', () => {

const value = 2 + 2;

expect(value).toBeGreaterThan(3);

expect(value).toBeGreaterThanOrEqual(3.5);

expect(value).toBeLessThan(5);

expect(value).toBeLessThanOrEqual(4.5);

// Với dữ liệu dạng số thì các bạn có thể dùng toBe hoặc toEqual

// để so sánh bằng đều được

expect(value).toBe(4);

expect(value).toEqual(4);

});

### 2.5 String

Các biểu thức so sánh string như sau

test('contain I', () => {

expect('team').not.toMatch(/I/);

});

test('Not contain abc', () => {

expect('Sabcdasdas').toMatch(/abc/);

});

### 2.6 Arrays

Sử dụng biểu thức contain

test('contain abc', () => {

expect(['abc', 'xyz']).toContain('abc');

});

function divideByZero() {

throw new Error('can not devide to 0');

}

test('test throw exeption divideByZero', () => {

expect(() => divideByZero()).toThrow();

expect(() => divideByZero()).toThrow(Error);

// compare message or use regex

expect(() => divideByZero()).toThrow('can not devide to 0');

expect(() => divideByZero()).toThrow(/can not/);

});

### 2.7 Async - promise

doublefunc = function(number) {

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

if(number%2 == 0) {

resolve(number\*2)

}

else {

reject("error 1")

}

})

}

test('test doublefunc', () => {

return demoSimple.doublefunc(12).then(data => {

expect(data).toBe(24);

});

});

### 2.8 async/await

doubleFunc = function(number) {

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

if(number%2 == 0) {

resolve(number\*2)

}

else {

reject("error 1")

}

})

}

syncFunc = async function(data) {

try {

const result = await compare1(data)

return result

} catch (err) {

throw err

}

}

Test

test('test async', async () => {

const data = await demoSimple.syncFunc(12);

expect(data).toBe(24);

});

### 2.9 Setup vs Teardown

Jest cung cấp một số function để setup tương tự như những công cụ test khác bao gồm

* beforeAll, afterAll: dùng để thiết lập một lần duy nhất cho các hàm test().
* beforeEach, afterEach: dùng để thiết lập cho mỗi hàm test().

beforeAll(() => console.log('1 - beforeAll'));

afterAll(() => console.log('1 - afterAll'));

beforeEach(() => console.log('1 - beforeEach'));

afterEach(() => console.log('1 - afterEach'));

test('', () => console.log('1 - test'));

Sẽ có kết quả sau:

1 - beforeAll

1 - beforeEach

1 - test

1 - afterEach

## 3. Mockup data

### 3.1 Mock function

Có hai cách để mock functions:

* Tạo một hàm giả để sử dụng trong test
* Viết một đoạn manual mock để ghi đè một module dependency.

Cách đơn giản nhất để tạo một Mock functions instance là dùng cú pháp jest.fn()

function forEach(items, callback) {

for (let index = 0; index < items.length; index++) {

callback(items[index]);

}

}

test('test foreach', () => {

// truyền vào tham số x, giá trị mockup trả về là 99 + x

const mockCallback = jest.fn(x => 99 + x);

forEach([0, 1], mockCallback);

// mock function được gọi 2 lần

expect(mockCallback.mock.calls.length).toBe(2);

// tham số thứ nhất của lần gọi đầu tiên là 0

expect(mockCallback.mock.calls[0][0]).toBe(0);

// giá trị trả về của lần gọi đầu tiên là = 0 + 99

expect(mockCallback.mock.results[0].value).toBe(99);

// tham số thứ nhất của lần gọi thứ 2 là 1

expect(mockCallback.mock.calls[1][0]).toBe(1);

// giá trị trả về của lần gọi đầu tiên là = 1 + 99

expect(mockCallback.mock.results[1].value).toBe(100);

});

### 3.2 Mock property

Là giá trị mỗi mock function lưu trữ được lấy thông qua .mock

const myMock = jest.fn();

const valueA = new myMock();

const valueB = {};

const bound = myMock.bind(valueB);

bound();

console.log(myMock.mock.instances);

// console.log

//[ mockConstructor {}, {} ]

### 3.3 Mock return value

Sử dụng mockReturnValueOnce để định nghĩa các giá trị trả về, nếu không sẽ lấy giá trị default qua ex sau

const myMockFn = jest

.fn()

.mockReturnValue('default')

.mockReturnValueOnce('first call')

.mockReturnValueOnce('second call');

// 'first call', 'second call', 'default', 'default'

console.log(myMockFn(), myMockFn(), myMockFn(), myMockFn());

### 3.4 Mock Module

Sử dụng cú pháp

jest.mock('module')

Ví dụ khi triển khai module user như sau

// users.js

import axios from 'axios';

class Users {

static all() {

return axios.get('/users.json').then(resp => resp.data);

}

}

export default Users;

Ta có thể tạo mock data trả về bằng cách

import axios from 'axios';

import Users from './users';

jest.mock('axios');

test('fetch users successful', () => {

const users = [{name: 'tphuong'}];

const resp = {data: users};

axios.get.mockResolvedValue(resp);

return Users.all().then(data => expect(data).toEqual(users));

});

Như vậy ở trên ta đã tạo mock data trả về resp = {data: users}; khi gọi Users.all() sẽ trả về data là resp.

### 3.5 Mock Implementations

Sử dụng mock implementations khi ta muốn định nghĩa lại implementation mặc định của module khác

Ex: Có 1 implement trả về giá trị false sau

// testImplement.js

module.exports = function () {

return false;

};

Nhưng ta muốn mock định nghĩa lại giá trị trả về là true

// mock.test.js

jest.mock('../testImplement');

const testImplement = require('../testImplement');

// testImplement is a mock function

testImplement.mockImplementation(() => true);

testImplement();

// ==> true