Тестування засобами Jest

https://jestjs.io/uk/docs/getting-started

Засобами Vitest (краще цей !!!)

https://vitest.dev/guide/

	jest	Vitest
Встановлення	`npm install --save-dev jest`	`npm install -D vitest`

Додайте наступну секцію до вашого `package.json`	`{`
	`  "scripts": {`
	`  "test": "jest"`
	`  }`
	`}`
		`{`
		`  "scripts": {`
		`  "test": "vitest"`
		`  }`
		`}`

Запуск	`npm run test`	`npm run test`
		`або`
		`npm run test:unit  (якщо додано при створенні(подивитись у package.json)`

Створення модульних тестів

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>Маємо файл з функціями</td><td>назва файлу, що тестується . js</td><td rowspan="4">---------- MyFile.js----------
function sum(a,b) {
    return a+b;
}
function mult(a,b) {
    return a*b;
}
.....
export {sum,mult,MyFunc}

---------- MyFile.test.js----------
import {sum, mult, MyFunc} from './MyFile'

describe("Tests of my functions ", () => {
//- - - - - - - - - - - - - - - - - - -
it("test sum 1", () => {
expect(sum(2, 3)).toEqual(5)
})
//- - - - - - - - - - - - - - - - - - -
it("test sum 2", () => {
expect(sum()).toEqual(NaN)
})
//- - - - - - - - - - - - - - - - - - -
it("test mult", () => {
expect(mult(5, 3)).toBeGreaterThan(10)
})
//- - - - - - - - - - - - - - - - - - -
it("Test Mu func", () => {
expect(MyFunc(5)).toMatchSnapshot()
})
}
);</td></tr>
<tr><td>Створюємо файл з тестами (до назви файлу з кодом додаємо “.test”)</td><td>назва файлу, що тестується .test .js</td></tr>
<tr><td>Описуємо групу, що буде містити тести</td><td>describe(назва групи текстів , ()=> {
...
опис тестів
...
})</td></tr>
<tr><td>Описуємо тести</td><td>it(текстовий опис тесту, () => {

expect(вираз для тестування).toEqual(очікуване значення)

})

Або

test(текстовий опис тесту, () => {

expect(вираз для тестування).методПорівняння (очікуване значення)

})

У якості методу перевірки можна використовувати багато спеціалізованих методів
https://jestjs.io/docs/en/expect.html</td></tr>
</table>

Запуск тестів	**npm run test**	

Методи для порівняння результату з очікуваним значення при тестуванні

https://jestjs.io/docs/en/expect.html

- `expect(value)`
- `expect.extend(matchers)`
- `expect.anything()`
- `expect.any(constructor)`
- `expect.arrayContaining(array)`
- `expect.assertions(number)`
- `expect.hasAssertions()`
- `expect.not.arrayContaining(array)`
- `expect.not.objectContaining(object)`
- `expect.not.stringContaining(string)`
- `expect.not.stringMatching(string | regexp)`
- `expect.objectContaining(object)`
- `expect.stringContaining(string)`
- `expect.stringMatching(string | regexp)`
- `expect.addSnapshotSerializer(serializer)`
- `.not`
- `.resolves`
- `.rejects`
- `.toBe(value)`
- `.toHaveBeenCalled()`
- `.toHaveBeenCalledTimes(number)`
- `.toHaveBeenCalledWith(arg1, arg2, ...)`
- `.toHaveBeenLastCalledWith(arg1, arg2, ...)`
- `.toHaveBeenNthCalledWith(nthCall, arg1, arg2, ....)`
- `.toHaveReturned()`
- `.toHaveReturnedTimes(number)`
- `.toHaveReturnedWith(value)`
- `.toHaveLastReturnedWith(value)`
- `.toHaveNthReturnedWith(nthCall, value)`
- `.toBeCloseTo(number, numDigits)`
- `.toBeDefined()`
- `.toBeFalsy()`
- `.toBeGreaterThan(number)`
- `.toBeGreaterThanOrEqual(number)`
- `.toBeLessThan(number)`
- `.toBeLessThanOrEqual(number)`
- `.toBeInstanceOf(Class)`
- `.toBeNull()`
- `.toBeTruthy()`
- `.toBeUndefined()`
- `.toBeNaN()`
- `.toContain(item)`
- `.toContainEqual(item)`
- `.toEqual(value)`
- `.toHaveLength(number)`
- `.toMatch(regexpOrString)`
- `.toMatchObject(object)`
- `.toHaveProperty(keyPath, value)`
- `.toMatchSnapshot(propertyMatchers, snapshotName)`
- `.toMatchInlineSnapshot(propertyMatchers, inlineSnapshot)`
- `.toStrictEqual(value)`
- `.toThrow(error)`
- `.toThrowErrorMatchingSnapshot()`
- `.toThrowErrorMatchingInlineSnapshot()`

Метод	***Призначення***	***Приклад***
toBe	перевіряє на строгу відповідність	`expect(sum(2, 2)).toBe(4);`
toEqual	рекурсивно перевіряє відповідність, корисно для порівняння масивів і об'єктів. Порівняння відбувається за значеннями полів, а не посиланнями на самі значення.	`expect(getUserData()).toEqual({ name: 'John', age: 30 });`
toBeNull	перевірка на відповідність значення null	`expect(findElement()).toBeNull();`
toBeUndefined	перевірка на відповідність undefined	`expect(getValue()).toBeUndefined();`
toBeDefined	перевірка на наявність методу чи поля об'єкта. Перевіряється на те, що значення не є undefined	`expect(result).toBeDefined();`
toBeTruthly	перевірка на відповідність true	`expect(isValid()).toBeTruthy();`
toBeFalsy	перевірка на відповідність false	`expect(isEmpty()).toBeFalsy();`
toBeGreaterThan	перевірка на те, що число більше	`expect(price).toBeGreaterThan(discountedPrice);`

toBeGreaterThanOrEqual	перевірка на більше або рівно	`expect(quantity).toBeGreaterThanOrEqual(minimumQuantity);`
toBeLessThan	перевірка на те, що число менше	`expect(speed).toBeLessThan(maximumSpeed);`
toBeLessThanOrEqual	перевірка на менше або рівно	`expect(count).toBeLessThanOrEqual(maximumCount);`

toBeCloseTo	порівняння з вказівкою округлення, другим аргументом. Все через те, що 0.2 + 0.1 = 0.30000000000000004	`expect(result).toBeCloseTo(expected, decimalPlaces);`
		
		`expect(calculatedValue).toBeCloseTo(expectedValue, 2);`

toMatch	порівняння рядка за допомогою регулярного виразу	`expect(email).toMatch(/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+.[a-zA-Z]{2,4}$/);`
toContain	перевірка наявності елемента в масиві	`expect(fruits).toContain('banana');`

Тестування асинхронного коду

```
------ з використанням then -----------
test('опис', () => {
  return асинхронна_дія.then(data => {

expect(вираз_тестування).toBe(значення_для_порівняння);
  });
});
```

```
//попредньо треба встановити axios: npm i axios
import axios from 'axios'

test('the data loaded', () => {
  return axios
    .get('https://dog.ceo/api/breeds/image/random')
    .then((res) => res.data)
    .then((data) => {
      expect(data.status).toBe('success')
    })
})
```

```
------- з використанням await --------
-- для перевірки на позитивний результат ---
test('опис', () => {
    const результат_асинхронної_дії= await fetchData();

expect(вираз_тестування).toBe(значення_для_порівняння);
 });

-- для перевірки на негативний результат ---
test('опис', () => {
  expect.assertions(1)
  try {
    await fetchData();
  } catch (e) {

expect(вираз_тестування).toBe(значення_для_порівняння);
  }
 });
```

```
test('the fetch - ok', async () => {
  const res = await  axios.get('https://dog.ceo/api/breeds/image/random')
  expect(res.data.status).toBe('success')
})

test('the fetch fails with an error', async () => {
  expect.assertions(1)
  try {
    await axios.get('https://dog.ceo/api/breeds/image/wrong_random')
  } catch (e) {
    expect(e.code).toMatch('ERR_BAD_REQUEST')
  }
})
```

Ви можете комбінувати `async` і `await` з `.resolves, .rejects.` або `.catch`

```
test('the data is peanut butter', async () => {
  await expect(fetchData()).resolves.toBe('peanut butter');
});
```

Ви також можете використати матчер `.resolves` у ваших конструкціях expect. Тоді Jest чекатиме поки проміс буде виконано.

```
test('the data is peanut butter', async () => {
  await expect(
    axios
      .get('https://dog.ceo/api/breeds/image/random')
      .then((res) => res.data.status)
  ).resolves.toBe('success')
})
```

```
test('the fetch fails with an error', async () => {
  await expect(fetchData()).rejects.toMatch('error');
});
```

```
test('the fetch fails with an error', () => {
  expect.assertions(1);
  return fetchData().catch(e => expect(e).toMatch('error'));
});
```

Snapshot

Якщо вираз для тестування дуже складний і важко визначити результат, який повинен бути при заданих параметрах, то можна використати метод toMatchSnapshot при використанні якого у перший раз при запуску тесту запам’ятовується значення функції. Це значення функції буде використовуватися як правильне значення при наступних запусках.

https://jestjs.io/docs/en/snapshot-testing

	---------- функція ----------
	`function MyFunc(x) {`
	`  return Math.floor(10*Math.sin(x)+311*Math.cos(x));`
	`}`
	
	---------- тест ----------
	`it("Test Mu func",()=>{`
	`  expect(MyFunc(5)).toMatchSnapshot()`
	`})`

it('текстовий опис тесту', () => {
expect(вираз).toMatchSnapshot();
});
	`import React from 'react';`
	`import Link from '../Link.react';`
	`import renderer from 'react-test-renderer';`
	
	`it('renders correctly', () => {`
	`  const tree = renderer`
	`  .create(<Link page="http://www.facebook.com">Facebook</Link>)`
	`  .toJSON();`
	`  expect(tree).toMatchSnapshot();`
	`});`

Для оновлення
1)додаємо у package.json ще один скрипт для оновлення з параметром «-u»
 	
2) Виконуємо команду
npm run testUpdate
 	
3) І далі виконуємо тестування оновлених значень як зазвичай «npm run test»
	`"scripts": {`
	`  "testUpdate": "jest -u"`
	`},`