Unit testing / Matchers mais usados com Jest

Certified Developer
The Ultimate Tech Degree

DigitalHouse >
Coding School

Matchers no Jest

Jest usa **matchers** para testar os diferentes valores que o código pode ter.

Vamos utilizar como exemplo uma parte do código desenvolvido que permite realizar operações matemáticas básicas e aplicar diferentes matchers a ela:

```
export const somar = (a, b) => a + b;
export const subtrair = (a, b) => a - b;
export const multiplicar = (a, b) => a * b;
export const dividir = (a, b) => a / b;
```

.toBe

Usado para comparar valores primitivos (inteiros, flutuantes, etc)

```
describe('Operações matemáticas', () => {
    test('Realizamos a soma', () => {
            expect(somar(1,1)).toBe(2);
    });
    test('Realizamos a subtração', () => {
          expect(subtrair(1,1)).toBe(0);
     });
});
```

JS

.toEqual

Usado para comparar objetos e todas as suas propriedades.

```
describe('Common matchers', () => {
   const dados = {
       name: 'Person 1',
        age: 10
   }
   const dados2 = {
       name: 'Person 1',
         age: 10
   }
  test('Verificamos se os objetos são iguais', () => {
       expect(dados).toEqual(dados2);
  });
});
```

.toBeLessThan

O valor é menor que:

```
test('Resultado menor que...', () => {
       expect(subtrair(5,3)).toBeLessThan(3);
});
```

JS

.toBeLessThanOrEqual

O valor é menor ou igual a:

```
  test('Resultado menor ou igual a...', () => {
      expect(subtrair(5,3)).toBeLessThanOrEqual(2);
   });
```

JS

.toBeGreaterThan

O valor é maior que:

```
test('Resultado maior que...', () => {
    expect(somar(5,5)).toBeGreaterThan(9);
 });
```

JS

.toBeGreaterThanOrEqual

O valor é maior ou igual a:

```
test('Resultado maior ou igual a...', () => {
     expect(multiplicar(2,5)).toBeGreaterThanOrEqual(10);
});
```

JS

Continuamos com mais exemplos

Vamos adicionar outras operações lógicas ao nosso código de exemplo, o que nos permitirá comparar valores booleanos, indefinidos e nulos.

```
export const somar = (a, b) => a + b;
export const subtrair = (a, b) => a - b;
export const multiplicar = (a, b) => a * b;
export const dividir = (a, b) => a / b;
export const isNull = null;
export const isFalse = false;
export const isTrue = true;
export const isUndefined = undefined;
```

.toBeTruthy

O valor é verdadeiro:

```
test('Resultado True', () => {
      expect(isTrue).toBeTruthy();
 });
```

JS

.toBeFalsy

O valor é falso:

```
 test('Resultado False', () => {
     expect(isFalse).toBeFalsy();
 });
```

JS

.toBeUndefined

O valor é Undefined:

```
test('Resultado Undefined...', () => {
       expect(isUndefined).toBeUndefined();
});
```

JS

.toBeNull

O valor é Null:

```
  test('Resultado Null...', () => {
      expect(isNull).toBeNull();
   });
```

JS

Continuamos com array e strings

Para continuar, veremos também alguns matchers usados para trabalhar com Arrays e Strings. Para isso, adicionamos as seguintes linhas ao nosso código:

```
const provincias = ['Álava','Girona','Huelva','Jaén','La Rioja','Madrid','Navarra'];
const dias =['Segunda-feira','Terça-feira','Quarta-feira',
             'Quinta-feira','Sexta-feira','Sábado','Domingo'];
const expReg = { responseOK: 'Response OK', responseFAIL: 'Response FAIL',
                 email: 'test@test.com', telefone: '919784852'}
export const arrProvincias = () => provincias;
export const arrDias = () => dias;
export const objExpReg = () => expReg;
```

.toBeContain

Contém o elemento dentro do array:

```
test('Madrid existe no array', () => {
    expect(arrProvincias()).toContain('Madrid');
 });
```

JS

.toHaveLength (array)

O array tem o tamanho:

```
test('O array dias tem 7 elementos', () => {
    expect(arrDias()).toHaveLength(7);
});
```

JS

.toHaveLength (String)

Também podemos usar este matcher para ver o tamanho de uma string:

```
const exp = objExpReg();

     test('Verificamos o tamanho da string', () => {

          expect(exp.responseFAIL).toHaveLength(13);

     });
```

JS

.toMuch

Verifica se um texto corresponde a uma expressão regular:

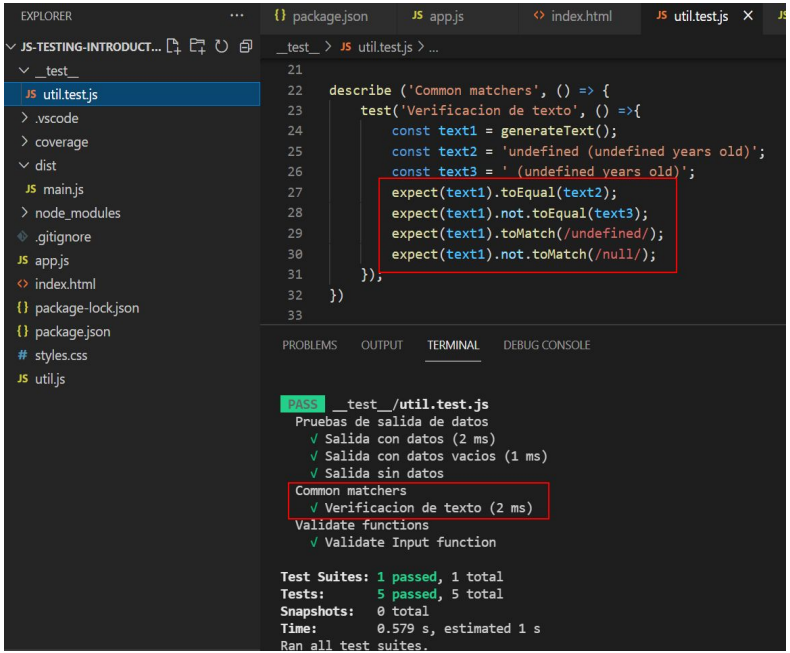
```
const exp = objExpReg();

  test('Verificamos o formato do e-mail', () => {

   expect(exp.email).toMatch(/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.([a-zA-Z]{2,4})+$/);

  })
```

JS

Seguimos aprendendo!

Se quiserem conhecer com mais detalhes ou conhecer toda a lista de matchers que podem ser usados com o Jest, é possível ver na documentação oficial:

https://jestjs.io/docs/expect



DigitalHouse>
Coding School