

Matchers mais usados com Jest

Matchers em Jest

O Jest usa matchers para testar os diferentes valores que o código pode ter. Vamos começar com um exemplo de uma parte de código desenvolvida 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, floats, 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

.toBeLessThan

O valor é menor que:

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

JS

.toBeLessThanOrEqual

O valor é menor ou igual que:

```
test('Resultado menor ou igual que...', () => {  
    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 que:

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

JS

Mais exemplos

Ao nosso código de exemplo, vamos adicionar outras operações lógicas, que nos permitirão 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

Arrays e Strings

Para continuar, veremos também alguns matchers que são usados para trabalhar com arrays e strings. Para fazer isso, adicionamos as seguintes linhas ao nosso código:

```
const estados = ['Minas Gerais', 'Rio de Janeiro', 'Pará', 'Pernambuco', 'Espírito Santo', 'São Paulo', 'Paraná'];
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 arrEstados = () => estados;
export const arrDias = () => dias;
export const objExpReg = () => expReg;
```

.toContain

Contém o elemento dentro do array:

```
test('São Paulo existe no array', () => {  
  expect(arrEstados()).toContain('São Paulo');  
});
```

JS

.toHaveLength (array)

O array tem o comprimento:

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

JS

.toHaveLength (string)

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

```
const exp = objExpReg();  
test('Verificamos o comprimento da string', () => {  
  expect(exp.responseFAIL).toHaveLength(13);  
});
```

JS

.toMatch

Verifica se um texto corresponde a uma expressão regular:

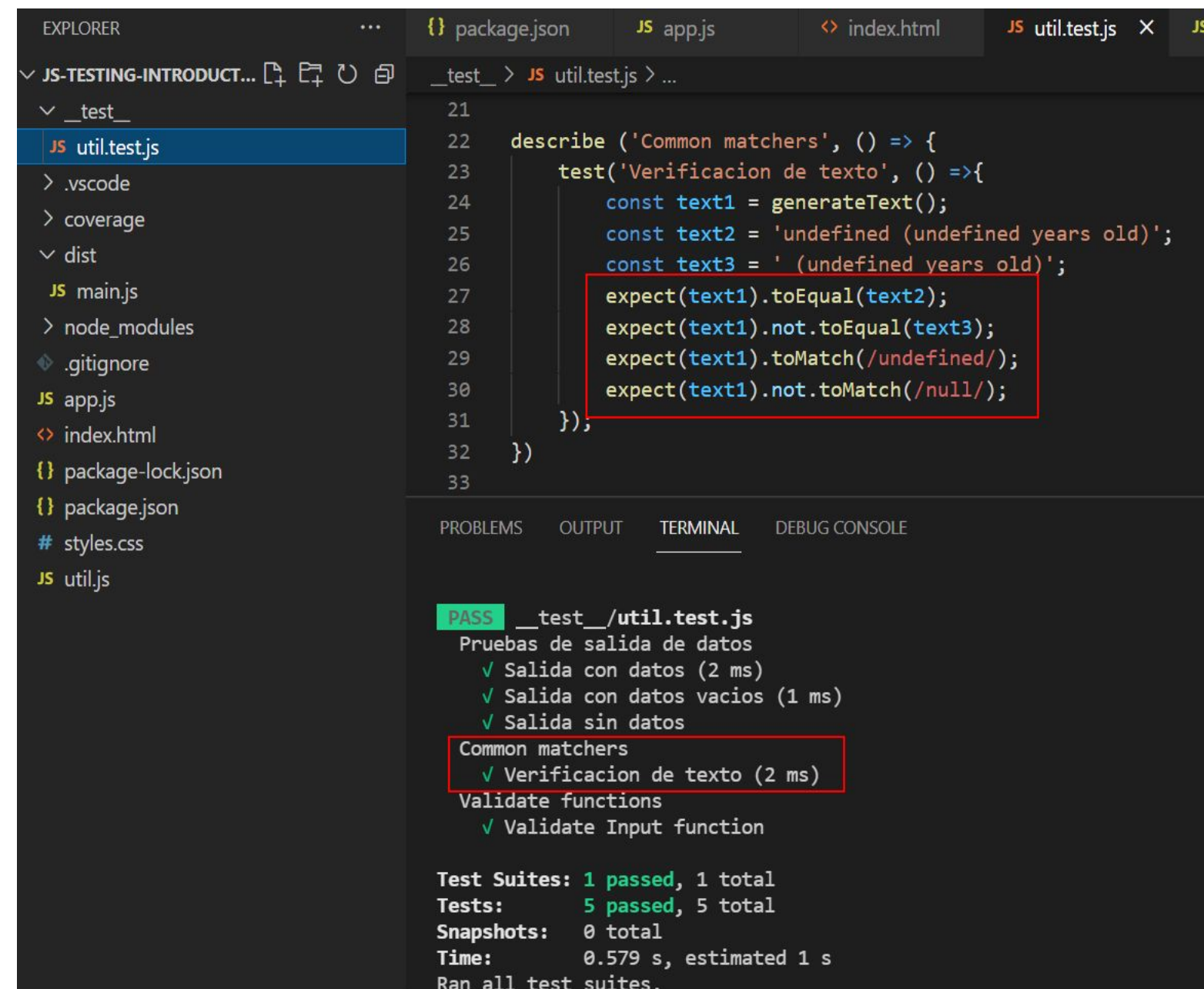
```
const exp = objExpReg();  
test('Verificamos o formato de e-mail', () => {  
  expect(exp.email).toMatch(/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/);  
});
```

JS

Vamos continuar aprendendo!

Se quisermos saber mais detalhes ou conhecer toda a lista de matchers que podem ser usados com o Jest, podemos ver na documentação oficial:

<https://jestjs.io/docs/expect>



The screenshot shows a VS Code editor with a project named 'JS-TESTING-INTRODUCT...'. The Explorer sidebar on the left shows a file tree with folders like '.vscode', 'coverage', 'dist', and files like 'main.js', 'app.js', 'index.html', 'package-lock.json', 'package.json', 'styles.css', and 'util.js'. The 'util.test.js' file is selected. The main editor displays the source code of 'util.test.js', which includes a 'describe' block for 'Common matchers' and a 'test' block for 'Verificacion de texto'. The test code defines three variables: 'text1' (from 'generateText()'), 'text2' ('undefined (undefined years old)'), and 'text3' ('(undefined years old)'). It then uses 'expect' to verify 'text1' against 'text2' (toEqual), 'text3' (not toEqual), and two regex patterns ('/undefined/' and '/null/'). A red box highlights these four expect statements. The bottom panel shows the Jest test results, indicating a 'PASS' for '___test___/util.test.js'. The results list several tests as passed, including 'Verificacion de texto (2 ms)', which is also highlighted with a red box. Summary statistics at the bottom show 1 passed test suite and 5 passed tests.

```
21
22 describe ('Common matchers', () => {
23   test('Verificacion de texto', () =>{
24     const text1 = generateText();
25     const text2 = 'undefined (undefined years old)';
26     const text3 = ' (undefined years old)';
27     expect(text1).toEqual(text2);
28     expect(text1).not.toEqual(text3);
29     expect(text1).toMatch(/undefined/);
30     expect(text1).not.toMatch(/null/);
31   });
32 })
33
```

PASS ___test___/util.test.js
Pruebas de salida de datos
✓ Salida con datos (2 ms)
✓ Salida con datos vacios (1 ms)
✓ Salida sin datos
Common matchers
✓ Verificacion de texto (2 ms)
Validate functions
✓ Validate Input function

Test Suites: 1 passed, 1 total
Tests: 5 passed, 5 total
Snapshots: 0 total
Time: 0.579 s, estimated 1 s
Ran all test suites.

Muito obrigado!