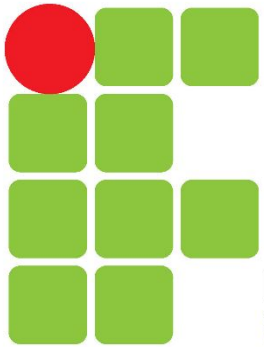


INSTITUTO FEDERAL
SÃO PAULO

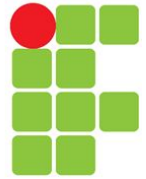
Testes Unitários no Node.js e Express

Desenvolvimento de Sistemas Web (DSWI6)

Prof. Luiz Gustavo Diniz de Oliveira Véras

E-mail: gustavo_veras@ifsp.edu.br

Objetivos

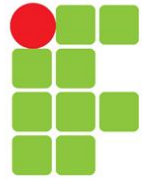


- ✔ Testes Unitários
  - ✔ Jest
  - ✔ expect
  - ✔ matchers
  - ✔ Testando APIs com supertest

Jest

Jest é uma ferramenta de test unitário desenvolvida pelo facebook, inicialmente tendo como foco testes de componentes no framework React.

Jest



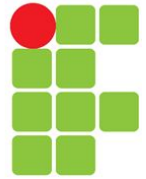
Fluxo básico para começar com Jest.

Instalando o jest no seu projeto

- Instale o jest em seu projeto
  - `npm install --save-dev jest`
- Configure o package.json com um script que invoca o jest.
  - ```
    "scripts": {
      "test": "jest"
    },
    ```

Pronto! Assim quando executamos o comando, todos os testes serão verificados.
    ```

Jest



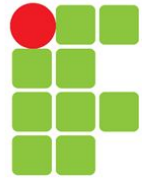
Fluxo básico para começar com Jest.

Criando testes

- Crie um módulo e exporte-o:
  - Exemplo: *`funcoes.js`*
- Crie um arquivo de testes. Usamos o prefixo **test.js** para indicar ao jest que ali existem testes a serem executados.
  - Exemplo: *`funcoes.test.js`*
- Execute o script de teste inserido no *package.json*.
  - `npm run test`

Pronto! Você terá executado seu primeiro teste!

Jest



Estrutura básica de um teste

Um teste consiste na sua descrição e em um callback que executará o teste. Geralmente é necessário importar módulos javascript que serão testados pelo jest. Testamos valores com a função **expect**.

```
const modulo = require("./modulo");

test("Descrição do test", () =>{
    expect(modulo()).toBe([valor esperado]);
})
```

Jest

Estrutura básica de

Um teste consiste e em um callback que executará o teste. sário importar módulos javascript que serão testados pelo jest. Testamos valores com a função **expect**.

Você pode ver alguns casos o uso da função **it()**, ao invés de **test().** Não há problema, eles são equivalente.

```
const modulo = require("./modulo");


test("Descrição do test", () =>{
    expect(modulo()).toBe([valor esperado]);
})
```

Exemplo

Testes para uma função de soma

soma.js

```
const soma = (a, b) => {
    return a + b;
};

module.exports = soma;
```

soma.test.js

```
const soma = require("./soma");

test("Soma 1 + 2 igual a 3", () =>{
    expect(soma(1, 2)).toBe(3);
})

test("Soma 1 + 2 não é igual a 2", () =>{
    expect(soma(1, 2)).not.toBe(2);
})
```

npm run test
(Lê qualquer arquivo com sufixo test.js)

```
Determining test suites to run.. PASS ./soma.test.js

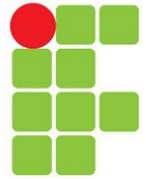
  √ Soma 1 + 2 igual a 3 (2 ms)
  √ Soma 1 + 2 não é igual a 2 (1 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.628 s, estimated 1 s
Ran all test suites.
```

Resultado dos testes

Jest

Matchers



Quando você está escrevendo testes, muitas das vezes você precisa checar se os valores satisfazem certas condições. **Expect()** lhe dá acesso a inúmeros "*matchers*" que permitem validar diferentes coisas.

Existem *matchers* para testar:

- Números
- Objetos
- Valores Truthy e Falsy
- Strings
- Arrays e iteraveis
- Exceções
- dentre outras possibilidades

Exemplos

```
// Este teste irá falhar
test("2 String com toBe", () => {
    // Comparação restrita de String com Number dará erro
    expect("2").toBe(2);
});

// Este teste irá passar
test("2 number com toBe", () => {
    expect(2).toBe(2);
});

// Este teste irá falhar
test("Objeto igual com toEqual", () => {
    let obj = {nome: "Tancredo"};
    obj["idade"] = 55;
    // Faltou o atributo "idade" no toEqual
    expect(obj).toEqual({nome: "Tancredo"});
});

// Este teste irá passar
test("Objeto igual com toEqual", () => {
    let obj = {nome: "Tancredo"};
    obj["idade"] = 55;
    expect(obj).toEqual({nome: "Tancredo", idade: 55});
});
```

Matchers Comuns

- **toBe()**
  - toBe utiliza Object.is para testar a igualdade exata.
  - Equivalente ao operador de comparação ===
- **toEqual()**
  - Mais usado para comparação com objetos.
  - Faz uma comparação em profundidade, visitando cada atributo do objeto e comparado com o valor passado em expect().

npm run test

```
FAIL ./be_equal.test.js
  × 2 String com toBe (5 ms)
  √ 2 number com toBe
  × Objeto igual com toEqual (6 ms)
  √ Objeto igual com toEqual (1 ms)
```

Exemplos

```
// Este teste irá falhar
test("2 String com toBe", () => {
    // Comparação restrita de String com Number dará erro
    expect("2").toBe(2);
});

// Este teste irá passar
test("2 number com toBe", () => {
    expect(2).toBe(2);
});

// Este teste irá falhar
test("Objeto igual com toEqual", () => {
    let obj = {nome: "Tancredo"};
    obj["idade"] = 55;
    // Faltou o atributo "idade" no toEqual
    expect(obj).toEqual({nome: "Tancredo"});
});

// Este teste irá passar
test("Objeto igual com toEqual", () => {
    let obj = {nome: "Tancredo"};
    obj["idade"] = 55;
    expect(obj).toEqual({nome: "Tancredo", idade: 55});
});
```

Se um dos testes do arquivo falhar, então todo o teste é considerado falho.

Matchers Comuns

- **toBe()**
  - toBe utiliza Object.is para testar a igualdade exata.
  - Equivalente ao operador de comparação ===
- **toEqual()**
  - Mais usado para comparação com objetos.
  - Faz uma comparação em profundidade, visitando cada atributo do objeto e comparado com o valor passado em expect().

npm run test

```
FAIL ./be_equal.test.js
  × 2 String com toBe (5 ms)
  √ 2 number com toBe
  × Objeto igual com toEqual (6 ms)
  √ Objeto igual com toEqual (1 ms)
```

Exemplos

```
// Este teste irá passar
test("2 String diferente de Number com toBe", () => {
    // Como o teste foi negado, irá passar
    expect("2").not.toBe(2);
});

// Este teste irá passar
test("2 number com toBe", () => {
    expect(2).toBe(2);
});

// Este teste irá passar
test("Objeto diferente com toEqual", () => {
    let obj = {nome: "Tancredo"};
    obj["idade"] = 55;
    // Como o teste foi negado, irá passar
    expect(obj).not.toEqual({nome: "Tancredo"});
});

// Este teste irá passar
test("Objeto igual com toEqual", () => {
    let obj = {nome: "Tancredo"};
    obj["idade"] = 55;
    expect(obj).toEqual({nome: "Tancredo", idade: 55});
});
```

Matchers Comuns

- **not**
  - Você pode adicionar o oposto do matcher com **not**.

npm run test

```
PASS ./not_be_equal.test.js
  √ 2 String com toBe (2 ms)
  √ 2 number com toBe (1 ms)
  √ Objeto igual com toEqual (1 ms)
  √ Objeto igual com toEqual
```

Outros tipos de matchers

```
// ------------------- Strings
test('não existe I em team', () => {
    expect('team').not.toMatch(/I/);
});

test('mas existe "stop" em Christoph', () => {
    expect('Christoph').toMatch(/stop/);
});
```

```
// ------------------- Números
test('dois mais dois', () => {
    const value = 2 + 2;
    expect(value).toBeGreaterThan(3);
    expect(value).toBeGreaterThanOrEqual(3.5);
    expect(value).toBeLessThan(5);
    expect(value).toBeLessThanOrEqual(4.5);

    // toBe e toEqual são equivalentes para números
    expect(value).toBe(4);
    expect(value).toEqual(4);
});
```

```
// ------------------- Arrays e iteráveis
const shoppingList = [
    'fraldas',
    'kleenex',
    'sacos de lixo',
    'papel toalha',
    'leite',
];

test('a lista de compras tem leite nela', () => {
    expect(shoppingList).toContain('leite');
    expect(new Set(shoppingList)).toContain('leite');
});
```


```
// ------------------- Números
test('dois mais dois', () => {
    const value = 2 + 2;
    expect(value).toBeGreaterThan(3);
    expect(value).toBeGreaterThanOrEqual(3.5);
    expect(value).toBeLessThan(5);
    expect(value).toBeLessThanOrEqual(4.5);

    // toBe e toEqual são equivalentes para números
    expect(value).toBe(4);
    expect(value).toEqual(4);
});
```

Outros tipos de matchers

```
// ------------------- Strings
test('não existe I em team', () => {
    expect('team').not.toMatch(/I/);
});

test('mas existe "stop" em Christoph', ()
    expect('Christoph').toMatch(/stop/)
});
```

```
// ------------------- Números
test('dois mais dois', () => {
    const value = 2 + 2;
    expect(value).toBeGreaterThan(3);
    expect(value).toBeGreaterThanOrEqual(3.5);
                                        5);
                                    ual(4.5);

                                    ntes para números
```

Para a lista de matchers, veja o link

https://jestjs.io/pt-BR/docs/expect

```
// ------------------- Arrays e iteráve
const shoppingList = [
    'fraldas',
    'kleenex',
    'sacos de lixo',
    'papel toalha',
    'leite',
];

test('a lista de compras tem leite nela', () => {
    expect(shoppingList).toContain('leite');
    expect(new Set(shoppingList)).toContain('leite');
});
```

```
                                    (3);
                                    anOrEqual(3.5);
    expect(value).toBeLessThan(5);
    expect(value).toBeLessThanOrEqual(4.5);

    // toBe e toEqual são equivalentes para números
    expect(value).toBe(4);
    expect(value).toEqual(4);
});
```

Testando código assíncrono

É comum em JavaScript executar código de forma assíncrona. Quando você tiver o código que é executado de forma assíncrona, Jest precisa saber quando o código que está testando for concluído, antes que possa passar para outro teste. Jest tem várias maneiras de lidar com isso.

```
// Não faça isso!
test('o dado é manteiga de amendoim', () => {
    function callback() {
      const data = 'pasta de amendoim';
      // deveria haver a falha do teste aqui
      expect(data).toBe('manteiga de amendoim');
    }

    setTimeout(callback, 5000);
  });
```

npm run test

```
PASS ./termina_sem_teste.test.js
  √ o dado é manteiga de amendoim (1 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.694 s, estimated 3 s
Ran all test suites matching /termina_sem_teste.
test.js/i.
Jest did not exit one second after the test run
has completed.
```

Testando código assíncrono

Há uma forma alternativa de test que corrige isto. Em vez de colocar o teste em uma função com um **argumento vazio**, use um único argumento chamado **done**. Jest aguardará até que a "callback" done é chamada antes de terminar o teste.

```
// Forma correta
test('o dado é manteiga de amendoim', (done) => {
    function callback() {
      const data = 'pasta de amendoim';
      // Agora o teste falha normalmente
      expect(data).not.toBe('manteiga de amendoim');
      done();
    }

    setTimeout(callback, 5000);
  });
```

Invocamos o done dentro da função assíncrona. Assim o teste não irá mais ser finalizado antes do tempo.

npm run test

```
FAIL ./done.test.js (7.163 s)
  × o dado é manteiga de amendoim (5025 ms)

  ● o dado é manteiga de amendoim
```

Testando código assíncrono

Se o seu código a ser testado é uma **Promise**, basta retornar para o JEST que ele vai esperar a promise ser resolvida. Se a promessa for rejeitada, o teste automaticamente irá falhar.

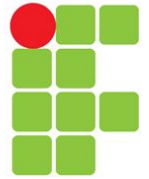
```
// Para promisses, o done não é necessário
test('o dado é manteiga de amendoim', () => {
  // Retorna uma Promise
  function asyncFunc(data) {
      return new Promise((resolve, reject) =>{
        resolve('manteiga de amendoim');
      })
    }

    const data = 'manteiga de amendoim';
    // Retornando a Promise o Jest aguardará a mesma
    // ser resolvida.
    return asyncFunc(data).then(
      (result) =>{
        expect(result).toBe(data);
      }
    );
  });
```

npm run test

```
PASS ./promise.test.js
  √ o dado é manteiga de amendoim
(36 ms)
```

Testando um aplicativo Express



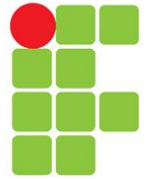
Podemos também utilizar o Jest para testes com nossa aplicação Express. Entretanto, precisamos de uma ferramenta para lidar especificamente com o protocolo HTTP.

SuperTest

A motivação com este módulo é fornecer uma abstração de alto nível para testar HTTP. Esse módulo ainda permite que você mantenha a API de nível inferior fornecida por um outro módulo de teste chamado **superagent (um client HTTP para o Node)**.

```
npm install supertest --save-dev
```

Testando um aplicativo Express



API do superagent para realizar requisições.

É assim que você utilizará o supertest:

```
const request = require('supertest');
const app = require('./app');

// callback
request(app)
  .post('/api/pet') // endpoint que será acessado
  .send({ name: 'Manny', species: 'cat' }) // envia um JSON post body
  .set('X-API-Key', 'foobar') // define HTTP Headers e seus valoers
  .set('accept', 'json')
  .end((err, res) => {
    // Calling the end function will send the request
    console.log(res.body)
 })
```

Similar ao superagent

```
const superagent = require('superagent');

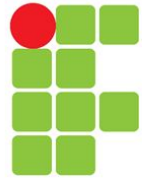
// callback
superagent
  .post('/api/pet') // endpoint que será acessado
  .send({ name: 'Manny', species: 'cat' }) // envia um JSON post body
  .set('X-API-Key', 'foobar') // define HTTP Headers e seus valoers
  .set('accept', 'json')
  .end((err, res) => {
    // Calling the end function will send the request
    console.log(res.body)
 })
```

Supertest: https://www.npmjs.com/package/supertest

Superagent: https://www.npmjs.com/package/supertest

Estrutura do response: https://devhints.io/superagent

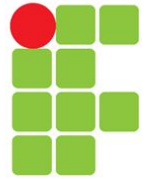
Testando um aplicativo Express



Etapas:

1. Instale os módulos em seu projeto;
2. Separe o **app** do **server;**
3. Crie os testes. Você pode componentizar os testes por rotas com **describe()**.
4. Use **done()** ou retorne a Promise para notificar que o teste encerra.

Testando um aplicativo Express

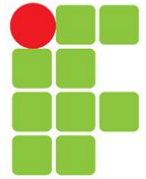


1. **Instale os módulos em seu projeto;**

```
npm install --save express
```

```
npm install --save-dev jest supertest
```

Testando um aplicativo Express



2. Separe o app do server;

A razão por trás disso é que ele ficará escutando a porta após o teste.

app.js

```
const express = require('express');
const bodyparser = require('body-parser');
const app = express();

app.use(bodyparser.text());

app.get('/', function(req, res) {
  res.status(200); //200 OK
  res.send("Retorno do callback para GET");
});

 //... Outros endpoints omitidos

module.exports = app;
```

server.js

```
const app = require("./app")
const port = 3000;

app.listen(port, function() {
  console.log(`Exemplo de app aguardando na porta ${port}!`)
});
```

Testando um aplicativo Express

3. **Crie os testes. Você pode componentizar os testes por rotas com describe().**

Estrutura dos testes com describe

```
const supertest = require("supertest");
describe("Todos os testes de um path da API",()
=> {
    test("Teste de Requisição GET", () =>{

    });
    test("Teste de Requisição POST", () =>{

    });
    test("Teste de Requisição PUT", () =>{

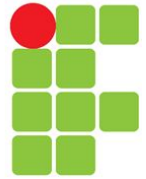
    });
    test("Teste de Requisição DELETE", () =>{

    });
});
```

Usamos a função request() do supertest para realizar uma requisição.

```
request(app)
        .get(path)
        .then(
          (response => {
                    // Teste de expectativa Jest
      }
    )
```

Testando um aplicativo Express



4. Use **done()** ou retorne a Promise para notificar que o teste encerra..

Teste de endpoint GET em app.test.js

```
// ... código omitido
test("Teste de Requisição GET", async () => {
        const response = await request(app)
                                    .get("/")
                                    .accept("Accept", "text/plain");
        expect(response.statusCode).toBe(200);
        expect(response.text)
            .toBe("Retorno do callback para GET");
});
// ... código omitido
```

Endpoint GET testado de app.js

```
// ... código omitido
app.get('/', function(req, res) {
  res.status(200); //200 OK
  res.set("Content-Type", "text/plain");
  res.send("Retorno do callback para GET");
});
// ... código omitido
```

Testando um aplicativo E

Definindo esta função como `async` terá o mesmo efeito de retornar uma Promise.

4. Use **done()** ou retorne a Promise para notificar que o teste encerra..

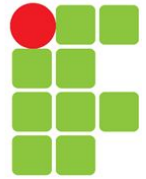
Teste de endpoint GET em app.test.js

```
// ... código omitido
test("Teste de Requisição GET", async () => {
        const response = await request(app)
                                .get("/")
                                .accept("Accept", "text/plain");
        expect(response.statusCode).toBe(200);
        expect(response.text)
            .toBe("Retorno do callback para GET");
});
// ... código omitido
```

Endpoint GET testado de app.js

```
// ... código omitido
app.get('/', function(req, res) {
  res.status(200); //200 OK
  res.set("Content-Type", "text/plain");
  res.send("Retorno do callback para GET");
});
// ... código omitido
```

Testando um aplicativo Express



4. Use **done()** ou retorne a Promise para notificar que o teste encerra..

Teste de endpoint POST em app.test.js

```
// ... código omitido
 test("Teste de Requisição POST", async () => {
        const dataToSend = "Dado enviado!";
        const response = await request(app)
            .post("/")
            .set('Accept', 'text/plain')
            .set('Content-Type', 'text/plain')
            .send(dataToSend); //envia dados para o servidor
        expect(response.statusCode).toBe(201);
        expect(response.text).toBe("Retorno do callback para POST. Info: "
                                    + dataToSend);
    });
// ... código omitido
```

Endpoint POST testado de app.js

```
// ... código omitido
app.post('/', function(req, res) {
  const data = req.body;
  res.status(201); //201 Created
  res.set("Content-Type", "text/plain");
  res.send("Retorno do callback para POST. Info: " +
             data);
});
// ... código omitido
```

Testando um ap

4. Use **done()** ou retorn r que o teste encerra..

Não se esqueça de alinhar o tipo esperado pelo client no teste e o tipo enviado pela API. Caso não seja o mesmo existe o risco do dado não ser lido.

Teste de endpoint GET (JSON) em app.test.js

```
// ... código omitido
test("Teste de Requisição GET JSON", async () => {
     const response = await request(app)
                          .get("/data")
                          .set('Accept', 'application/json')
     expect(response.statusCode).toBe(200);
     expect(response.body).toEqual({ nome: "Marcio", idade: 35 });
});
// ... código omitido
```

Endpoint GET (JSON) testado de app.js

```
// ... código omitido
app.get('/data', function(req, res) {
  res.status(200); //200 OK
  res.set("Content-Type", "application/json");
  res.send({ nome: "Marcio", idade: 35});
});
// ... código omitido
```

Links

https://jestjs.io/pt-BR/docs/using-matchers

https://jestjs.io/pt-BR/docs/expect

https://jestjs.io/pt-BR/docs/asynchronous