



# **WACAD016 – Fundamentos de Teste de Software - Aula 01**

---

Júlia Luiza

[jlslc@icomp.ufam.edu.br](mailto:jlslc@icomp.ufam.edu.br)

# Cronograma: Aula 01

---

## Visão geral

- Por que é importante testar?
- Diferentes tipos de teste;
- Ferramentas disponíveis.

## Jest

- O que é e vantagens;
- Instalação utilizando npm;
- Configurações adicionais.

## Como escrever testes

- Sintaxe do Jest;
- Cobertura de código;
- Prática com testes para funções;
- Boas práticas.

# Avaliação

## Questionários no colabweb

- Aula 1 -> Questionário 1
- Aula 2 -> Questionário 2
- Média questionários = N1

## Trabalhos práticos

- Aula 1 -> Prática 1
- Aula 2 -> Prática 2
- Média práticas = N2

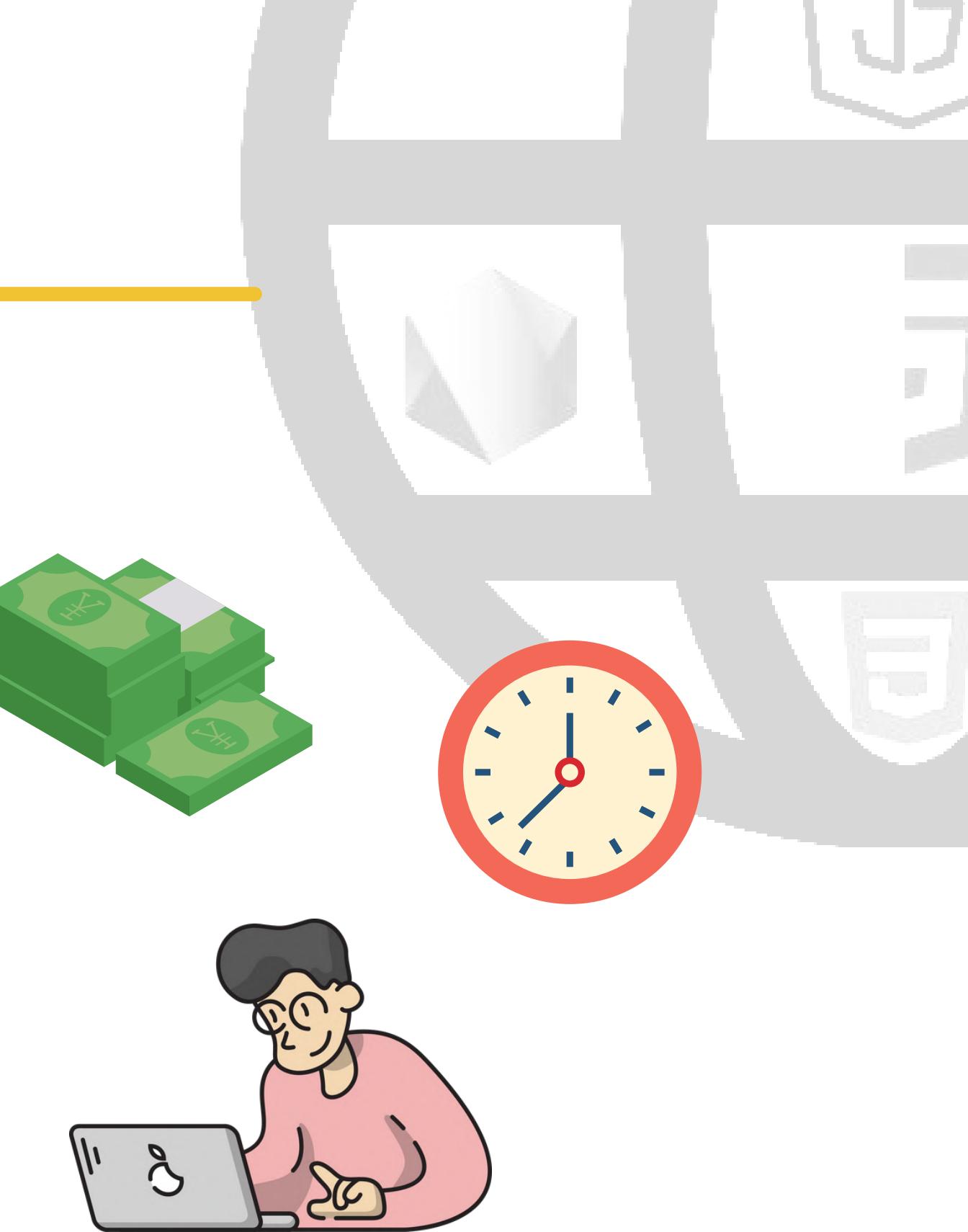
$$\text{Média disciplina} = (N1 + N2) / 2$$

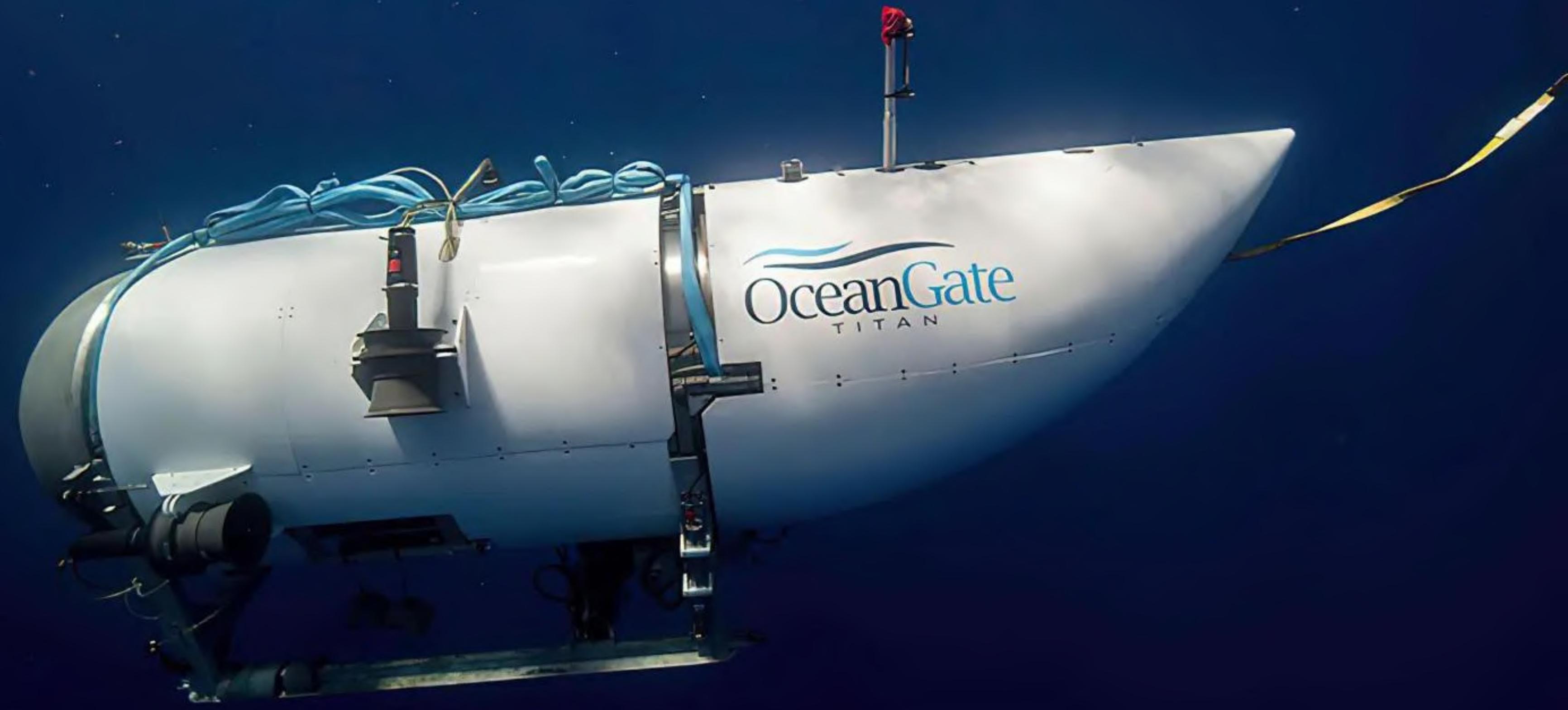


# Por que testar software?

---

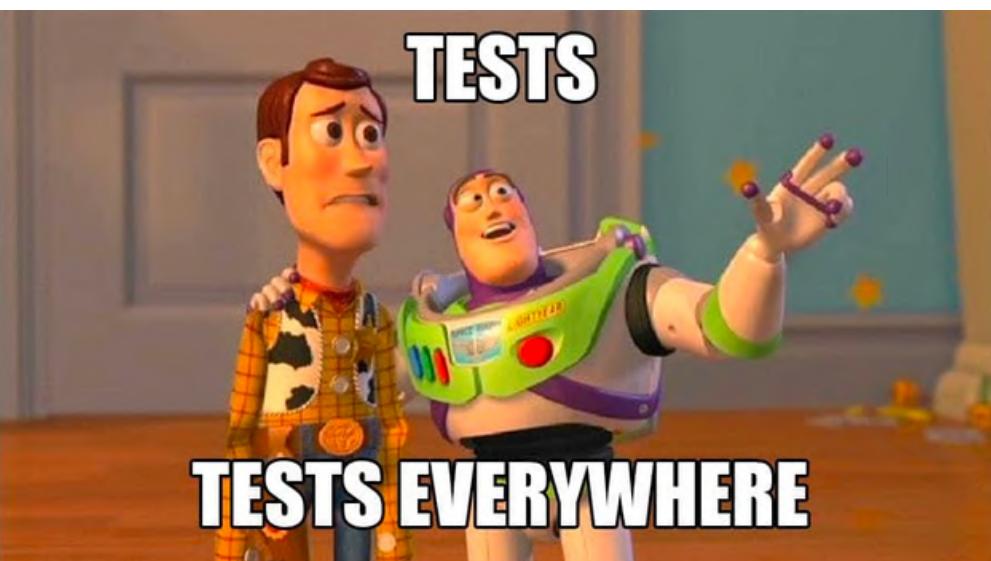
- Verificação dos requisitos;
- Qualidade;
- Consistência;
- Manutenibilidade;
- Confiabilidade;
- Evolução do software;
- Evitar que erros cheguem até o usuário final.

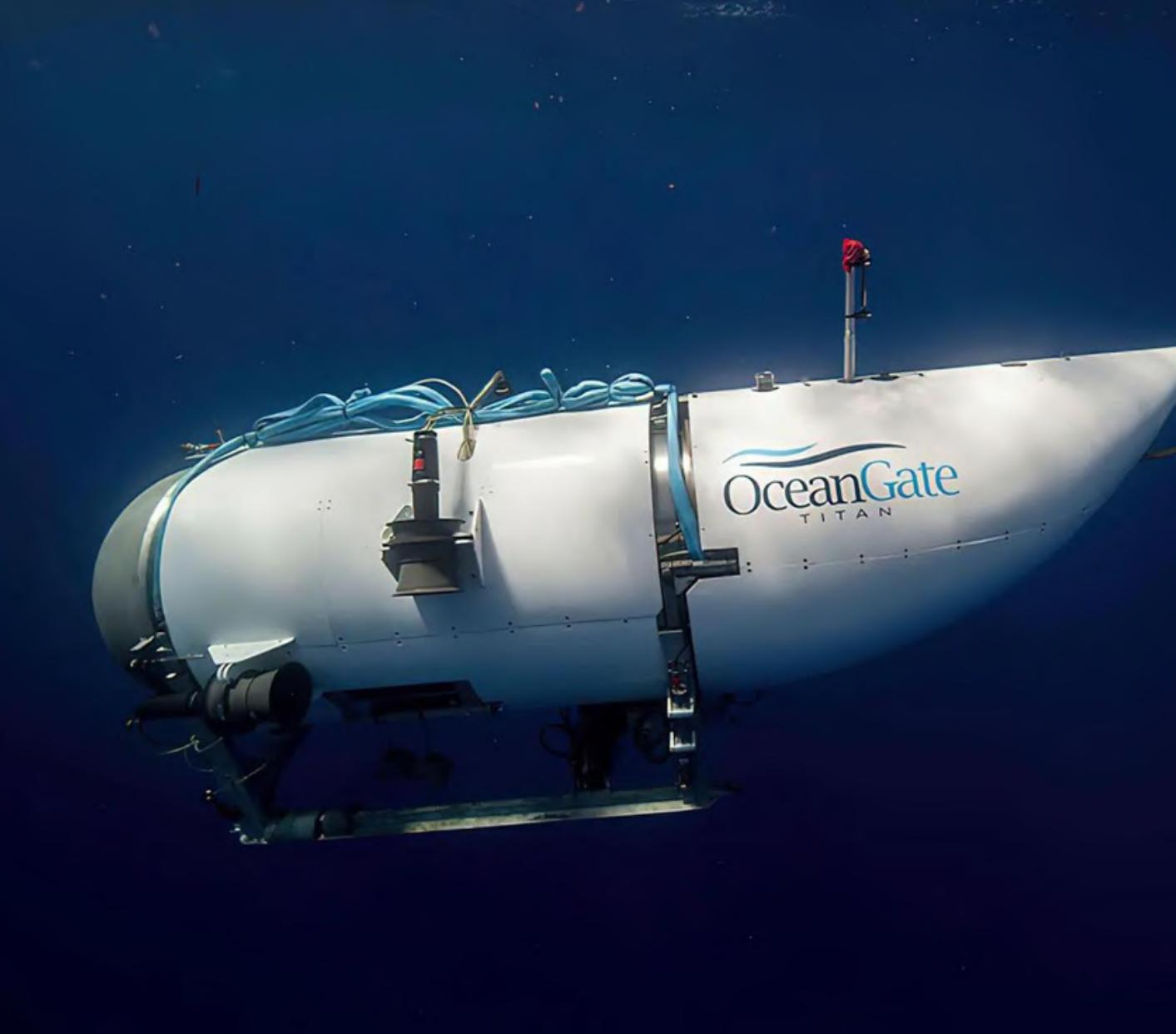
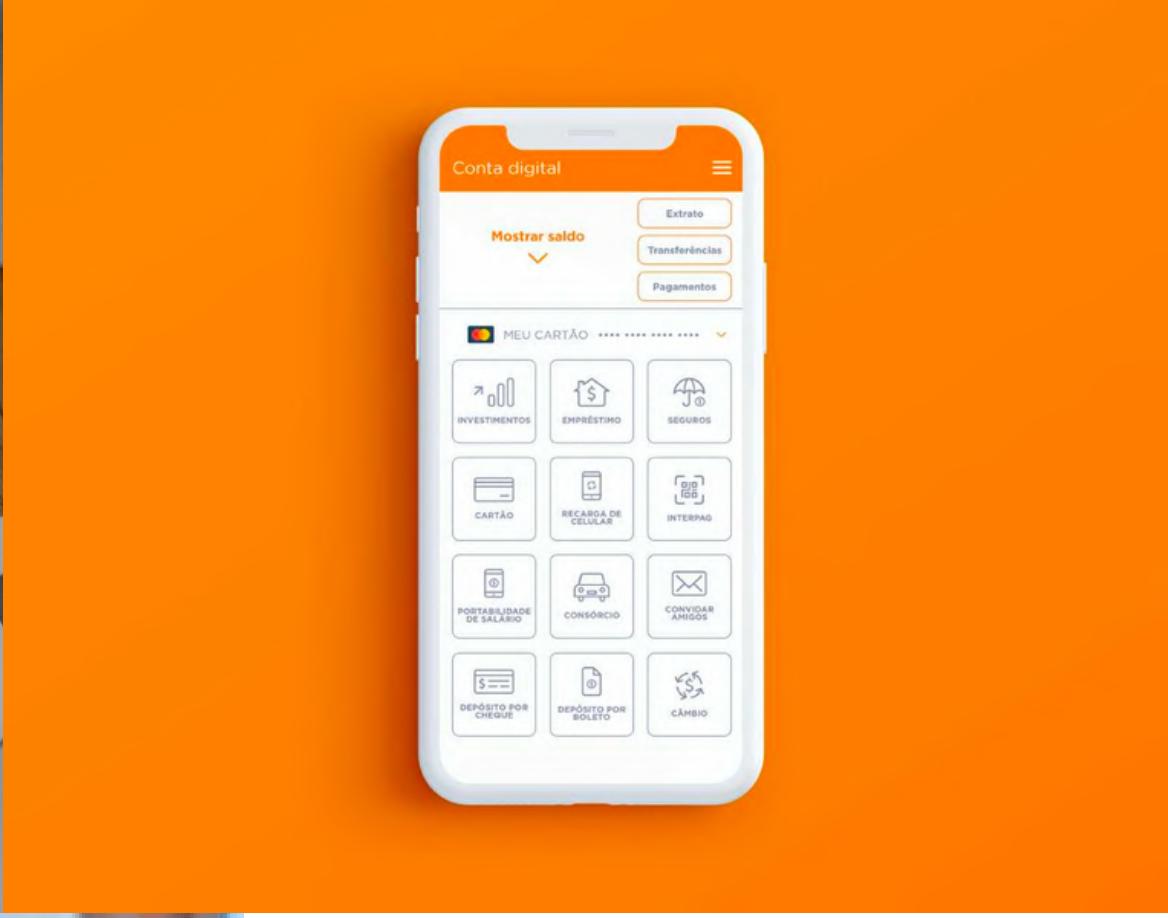




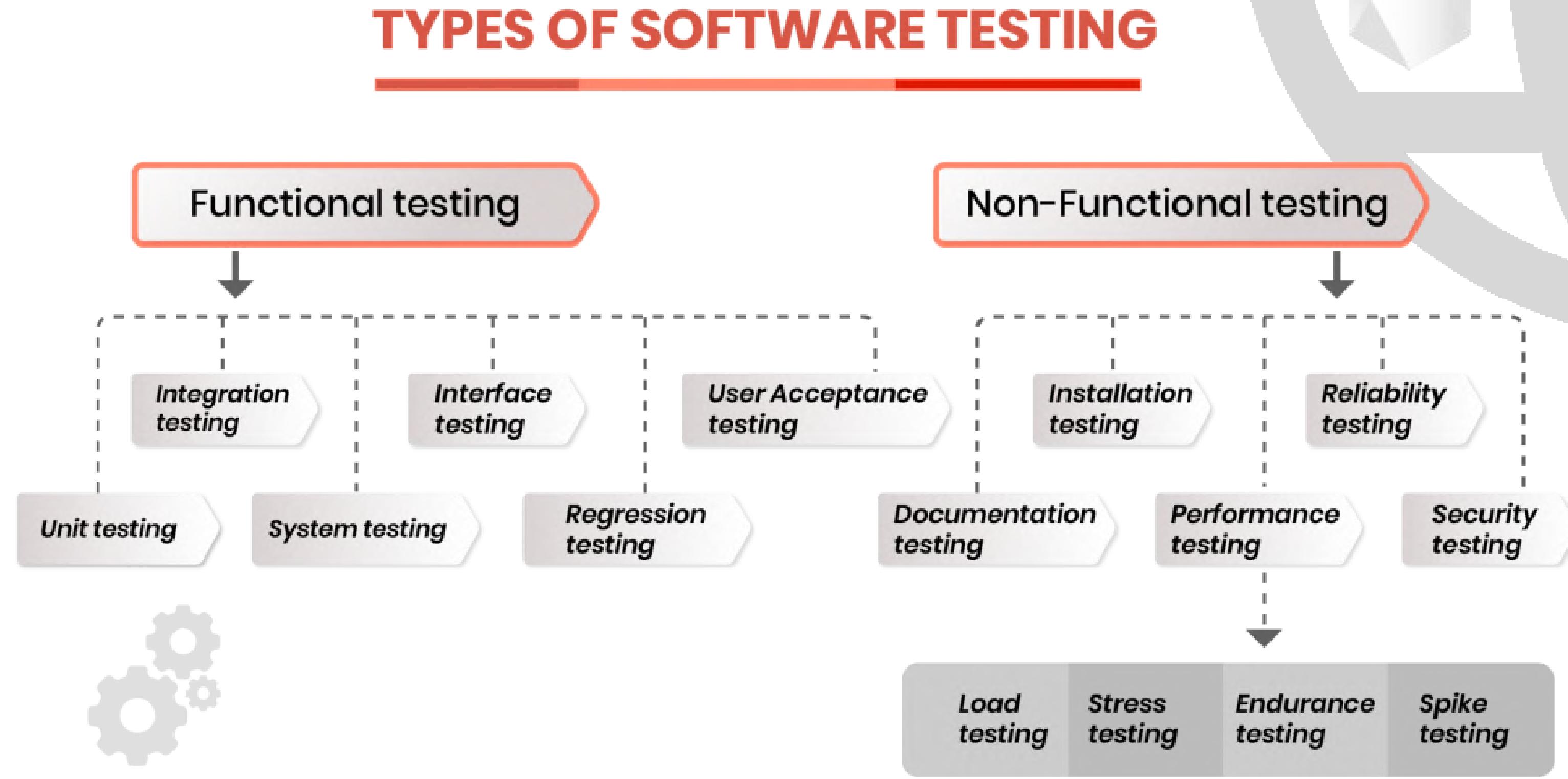
# Quando testar?

- **Durante todas as fases do projeto:**
  - Antes de iniciar o desenvolvimento em si;
  - Durante o desenvolvimento:
    - Implementação de novas funcionalidades;
    - Manutenção de funcionalidades existentes.
  - Depois do lançamento do software para o usuário final.
- Independentemente da criticidade e complexidade do software, **os testes são fundamentais.**

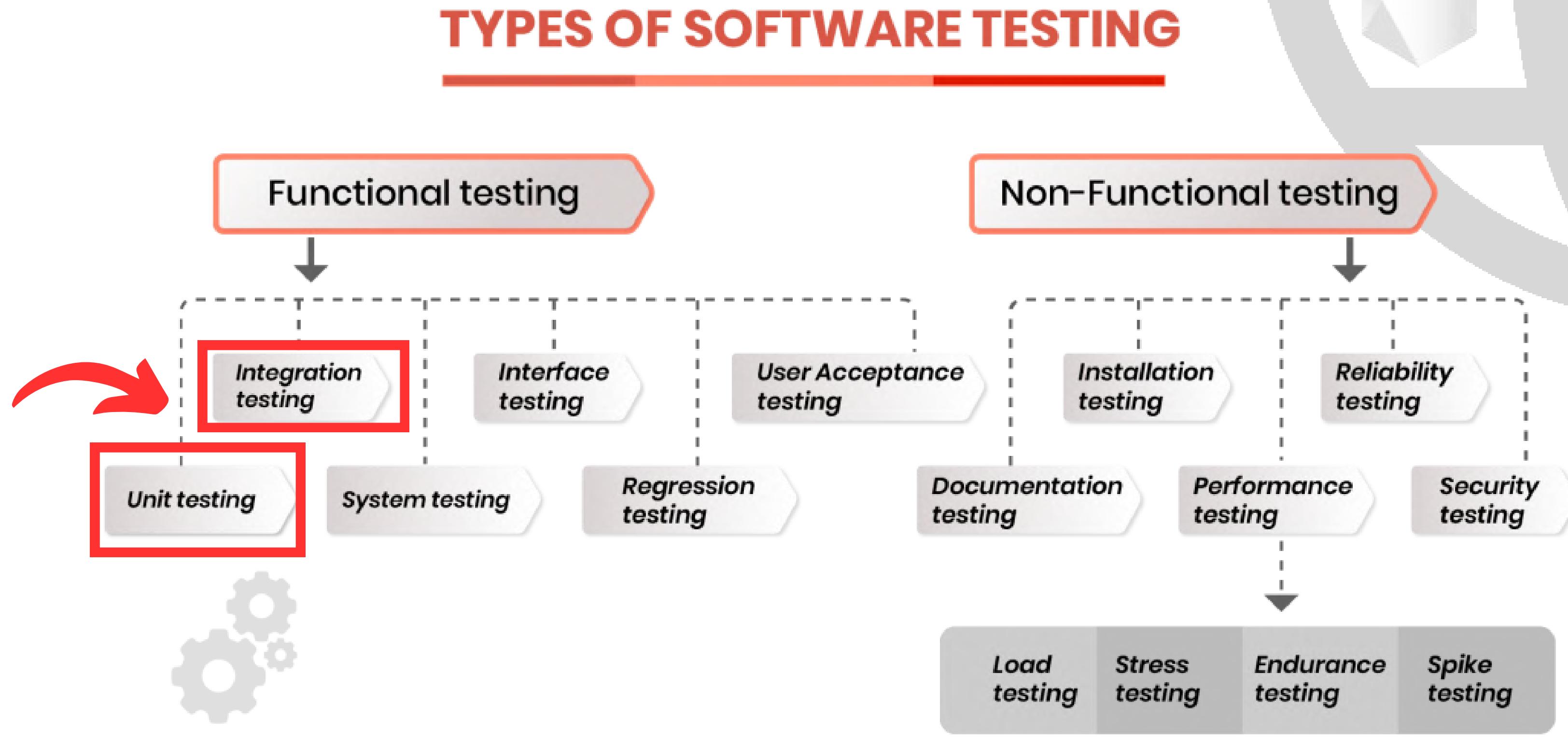




# Diferentes tipos de teste



# Diferentes tipos de teste





## Comprar novamente

[Ver todos e gerenciar](#)

Umidificador de Ar Elgin  
Digital Inteligente, 2,5L  
R\$ 189,99

✓prime

[Adicionar ao carrinho](#)

Cola Puzzle Brilhante  
R\$ 28,99

✓prime

[Adicionar ao carrinho](#)

Antipulgas Zoetis  
Simparic 40 Mg Para Cães  
R\$ 219,90  
(R\$ 73,30/unidade)

✓prime

[Adicionar ao carrinho](#)

## Suas listas

[Lista de compras](#)[Cha de bebe da Erica](#)[Márcia](#)[Criar uma Lista de desejos](#)

## Sua conta

[Sua conta](#)[Seus pedidos](#)[Sua Lista de desejos](#)[Recomendados para você](#)[Programe e Poupe](#)[Sua assinatura Prime](#)[Inscrições e assinaturas](#)[Gerencie seu conteúdo e dispositivos](#)[Seu Amazon Music](#)[Seu Prime Video](#)[Seu Kindle Unlimited](#)[Seu Amazon Drive em Amazon.com](#)[Seus aplicativos e dispositivos](#)[Trocá de conta](#)[Sair da conta](#)

## Continue de onde parou

[Apple iPhone SE \(3ª geração\) - Apple iPhone 14 \(128GB\)](#)[Apple iPhone 14 Pro \(128GB\) - Kit Capa Anti Impacto](#)

## Continuar comprando

[Ebooks kindle  
Visto 2x](#)[Malas de bordo  
Visto 1x](#)[Kits de brinquedos diferentes  
Visto 5x](#)[Colchonetes para exteriores  
Visto 2x](#)[Visualize seu histórico de navegação](#)

## Compre novamente

[Ver todos e gerenciar](#)

Umidificador de Ar Elgin  
Digital Inteligente, 2,5L  
R\$ 189,99

✓prime

[Adicionar ao carrinho](#)

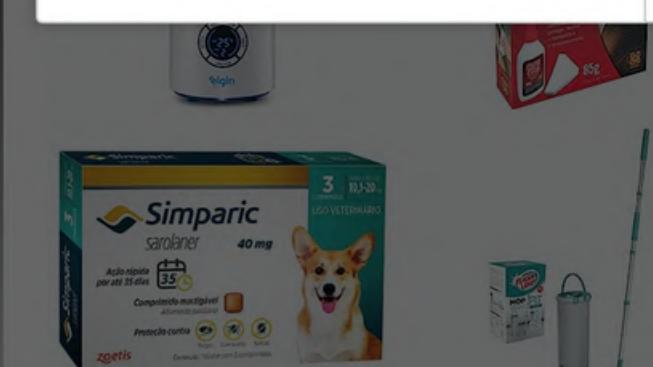
Cola Puzzle Brilhante  
R\$ 28,99

✓prime

[Adicionar ao carrinho](#)

Antipulgas Zoetis  
Simparic 40 Mg Para Cães  
R\$ 219,90  
(R\$ 73,30/unidade)

✓prime

[Adicionar ao carrinho](#)[Compre seus itens essenciais do dia a dia](#)[Ver mais itens para comprar novamente](#)

## Prime Video: Recomendado para você

[O Pacto](#)

## Ganhe créditos completando missões



## Ofertas do Dia



**amazon.com.br** Enviar para Júlia Todos ▾ Pesquisa Amazon.com.br

Devoluções e Pedidos Carrinho

**Ver endereços cadastrados**

**Continue de onde parou**

Apple iPhone SE (3<sup>a</sup> ger...) Apple iPhone 14 (128...  
Apple iPhone 14 Pro (1... Kit Capa Anti Impacto...

**Continuar comprando**

Ebooks kindle Malas de bordo  
Visto 1 Visto 2

Kits de brinquedos d... Colchonetes para ex...  
Visto 5x Visto 2x

Visualize seu histórico de navegação

**Comprar novamente**

Ver todos e gerenciar

Umidificador de Ar Elgin Digital Inteligente, 2,5... R\$ 189,99 ✓prime Adicionar ao carrinho

Cola Puzzle Brilhante R\$ 28,99 ✓prime Adicionar ao carrinho

Antipulgas Zoetis Simparic 40 Mg Para C... R\$ 219,90 (R\$ 73,30/unidade) ✓prime Adicionar ao carrinho

**Suas listas**

Lista de compras Cha de bebe da Erica Márcia  
Criar uma Lista de desejos

**Sua conta**

Sua conta Seus pedid... Sua Lista d... Recomendados para você Programe e Poupe Sua assinatura Prime Inscrições e assinaturas Gerencie seu conteúdo e dispositivos Seu Amazon Music Seu Prime Video Seu Kindle Unlimited Seu Amazon Drive em Amazon.com Seus aplicativos e dispositivos Trocar de conta Sair da conta

**Adicionar ao carrinho**

**Features isoladas**

Compre seus itens essenciais do dia a dia

Ver mais itens para comprar novamente

prime video | CHANNELS ASSINE JÁ VERIFIQUE A CLASSIFICAÇÃO INDICATIVA Patrocinado

**Prime Video: Recomendado para você**

O Pacto

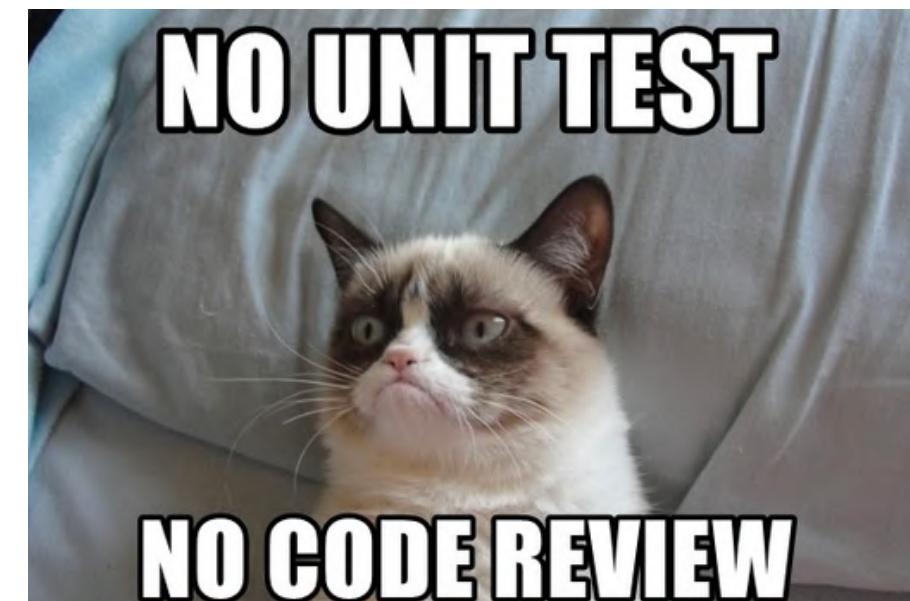
**Ganhe créditos completando missões**

**Ofertas do Dia**

w.amazon.com.br/gp/css/homepage.html?ref\_=nav\_youraccount\_btn

# Testes unitários

- Componentes individuais, com lógicas isoladas e *preferencialmente* sem efeitos colaterais são testados;
- **Valida o funcionamento de cada unidade do software, individualmente;**
- Implementados durante o desenvolvimento do software, pelos desenvolvedores:
  - Após a implementação ou manutenção de uma função, método, classe, um componente front-end com comportamento, etc.





Venda na Amazon Ofertas do Dia eBooks Kindle Ferramentas e Construção Atendimento ao Cliente Comprar nov...

Buscar produto

Continue de onde parou



Apple iPhone SE (3ª ger...) Apple iPhone 14 (128...



Apple iPhone 14 Pro (1... Kit Capa Anti Impacto...

Continuar comprando



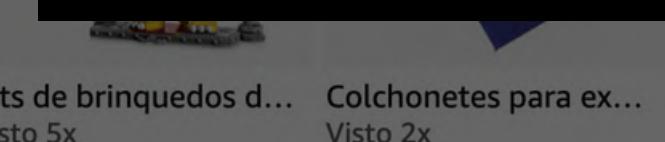
Ebooks kindle Malas de bordo

Visto 1x Visto 1x

Kits de brinquedos d... Colchonetes para ex...

Visto 5x Visto 2x

Visualize seu histórico de navegação



Comprar novamente

[Ver todos e gerenciar](#)



Umidificador de Ar Elgin Digital Inteligente, 2,5...

R\$ 189,99

✓prime

[Adicionar ao carrinho](#)



Cola Puzzle Brilhante

R\$ 28,99

✓prime

[Adicionar ao carrinho](#)



Antipulgas Zoetis Simparic 40 Mg Para C...

R\$ 219,90

(R\$ 73,30/unidade)

✓prime

[Adicionar ao carrinho](#)

Suas li...

[Lista de](#)

Cha de bebe da Erica

Márcia

[Criar uma Lista de desejos](#)

Realizar login

Seus pedidos

Realizar compra de  
produto

Inscrições e assinaturas

Gerencie seu conteúdo e  
dispositivos

Seu Amazon Music

Seu Prime Video

Seu Kindle Unlimited

Seu Amazon Drive em  
Amazon.com

Seus aplicativos e dispositivos

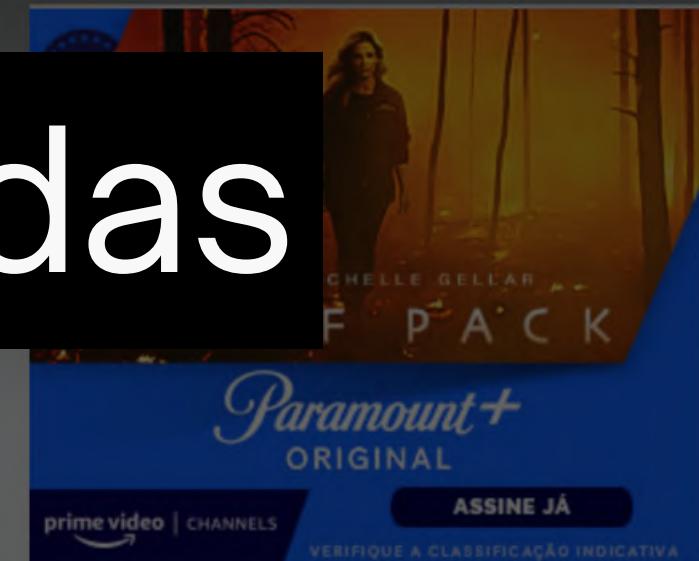
Trocar de conta

Sair da conta

# Features integradas

Compre seus itens essenciais do dia a dia

[Ver mais itens para comprar novamente](#)



Ofertas do Dia



Prime Video: Recomendado para você

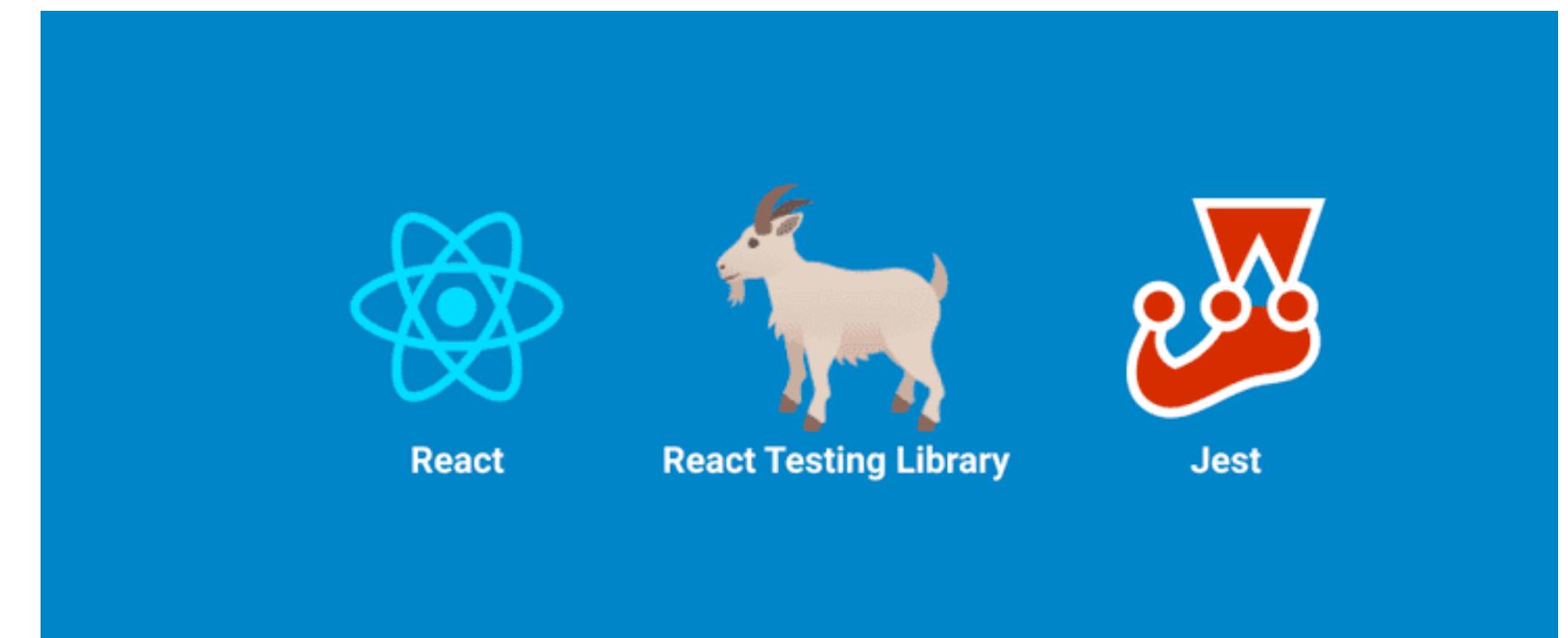
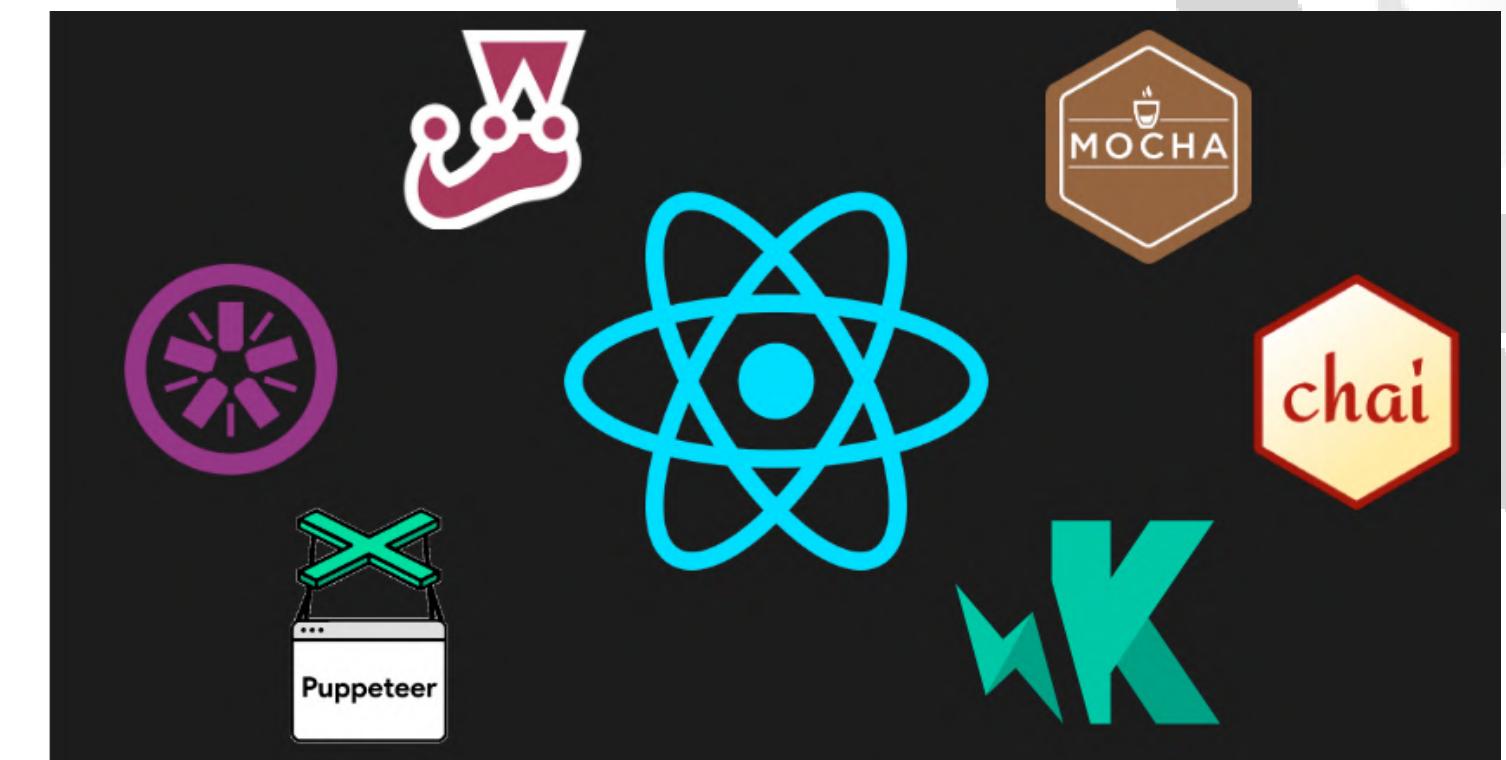
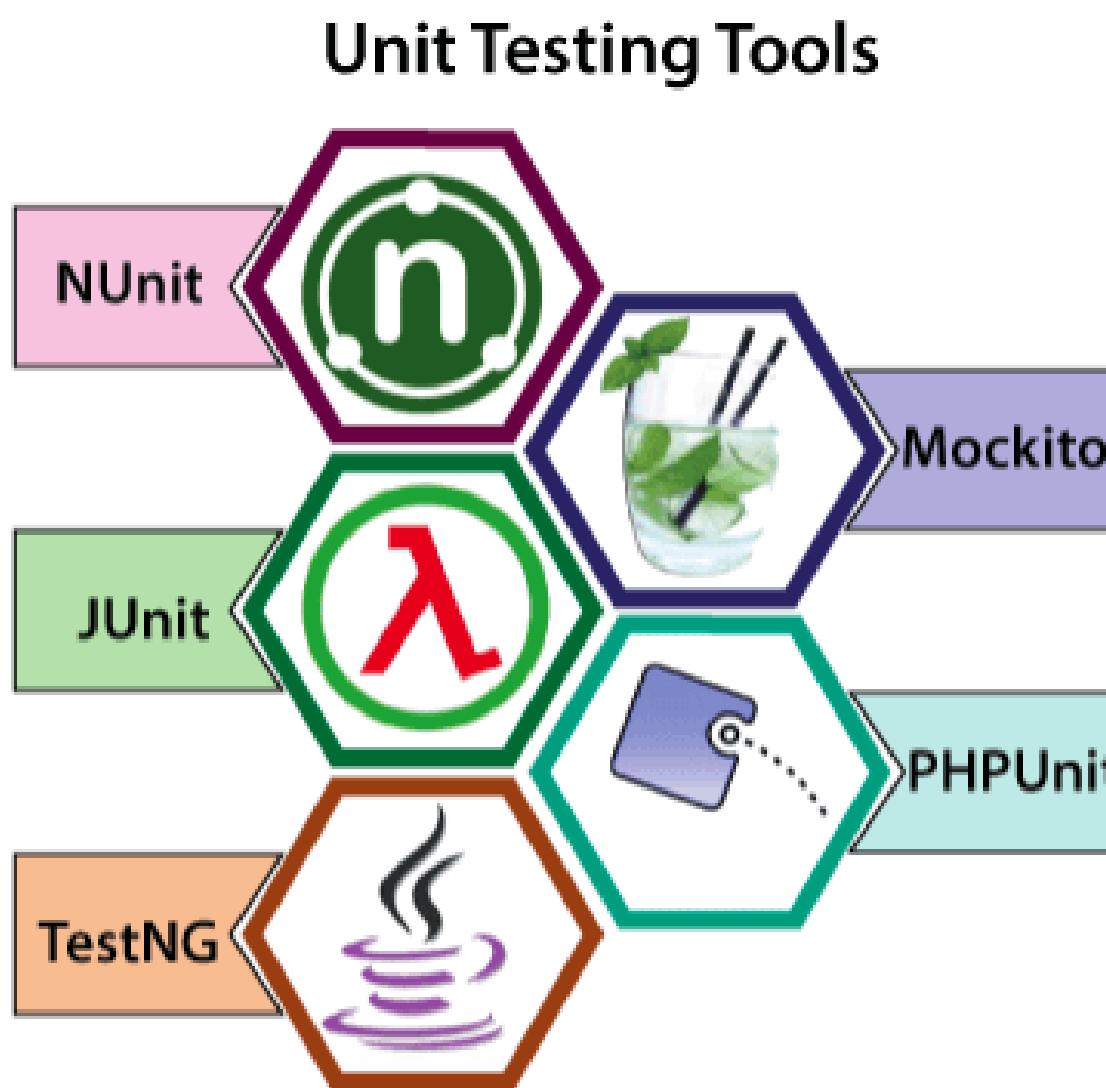
O Pacto

# Testes de integração

- Múltiplos componentes, com lógicas integradas e efeitos colaterais são testados;
- **Valida o funcionamento das unidades de software de forma integrada;**
- Complementares aos testes unitários;
- Exemplos: chamada de um módulo para outro, chamada externa, acesso ao banco de dados, acesso à API.



# Ferramentas de testes



# Cronograma: Aula 01

## Visão geral

- Por que é importante testar?
- Diferentes tipos de teste;
- Ferramentas disponíveis.

## Jest

- O que é e vantagens;
- Instalação utilizando npm;
- Configurações adicionais.

## Como escrever testes

- Sintaxe do Jest;
- Cobertura de código;
- Prática com testes para funções;
- Boas práticas.



ATENÇÃO

Dúvidas?

# Jest

---

"Jest é um poderoso Framework de Testes em JavaScript com um foco na simplicidade."

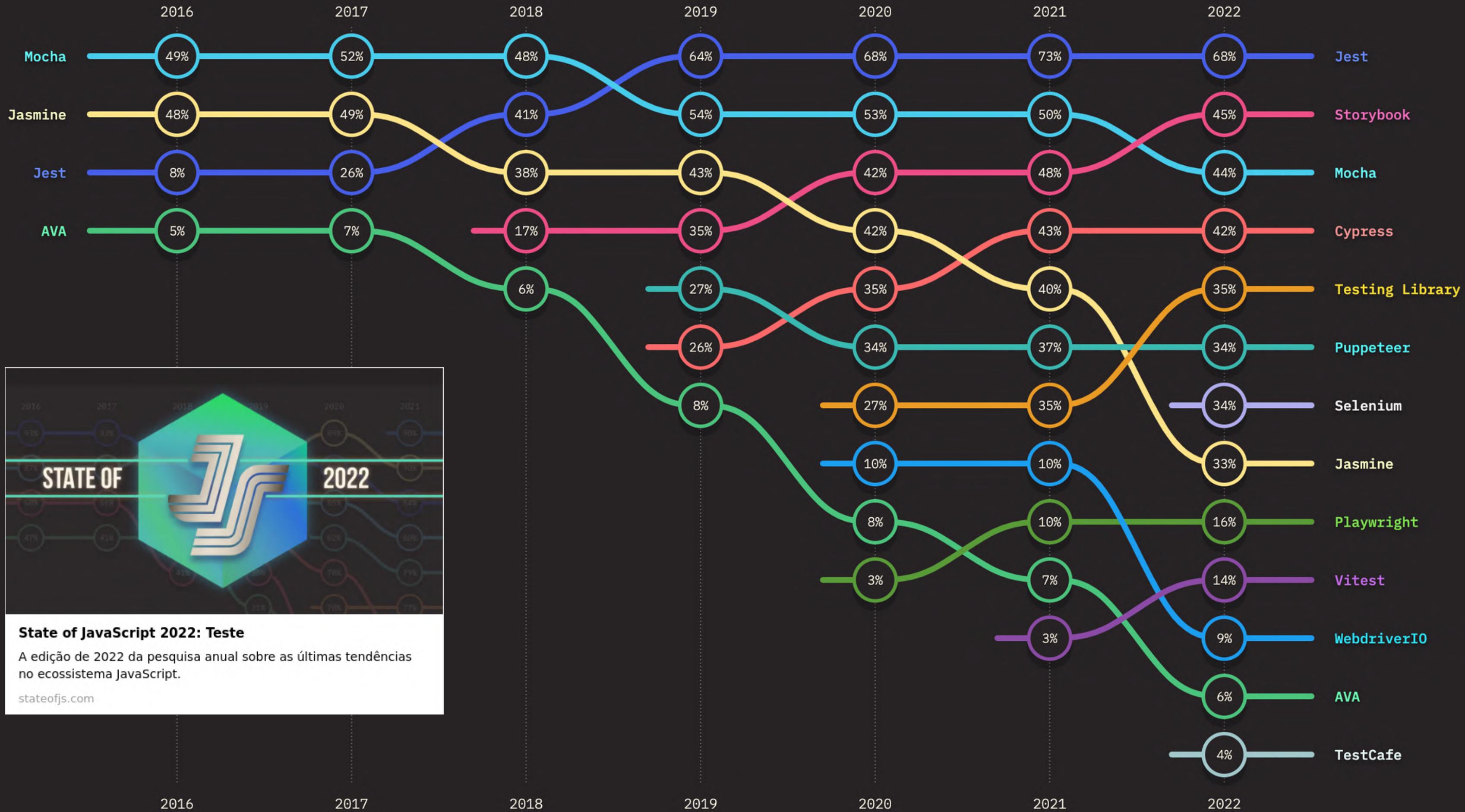
— <https://jestjs.io/pt-BR/>



## State of JavaScript 2022: Teste

A edição de 2022 da pesquisa anual sobre as últimas tendências no ecossistema JavaScript.

[stateofjs.com](http://stateofjs.com)



Conhecimento    Uso    Interesse    Satisfação

# Jest

## zero configuração

Jest visa trabalhar fora da caixa, sem configuração, na maioria dos projetos JavaScript.

## isolado

Os testes são paralelos e executados em seus próprios processos para maximizar o desempenho.

## excelente api

De `it` para `expect` - Jest tem todo o conjunto de ferramentas em um só lugar. Bem documentado e mantido.

### Install

```
> npm i jest
```

### Repository

❖ [github.com/facebook/jest](https://github.com/facebook/jest)

### Homepage

⌚ [jestjs.io/](https://jestjs.io/)

### ↓ Weekly Downloads

21.722.310



<https://www.npmjs.com/package/jest?activeTab=readme>

# RÁPIDO E SEGURO

```
PASS packages/diff-sequences/src/_tests__/index.test.js
PASS packages/jest-diff/src/_tests__/diff.test.js
PASS packages/jest-mock/src/_tests__/jest_mock.test.js
PASS packages/jest-util/src/_tests__/fakeTimers.test.js
PASS packages/pretty-format/src/_tests__/prettyFormat.test.js

RUNS packages/jest-haste-map/src/_tests__/index.test.js
RUNS packages/pretty-format/src/_tests__/DOMElement.test.js
RUNS packages/jest-config/src/_tests__/normalize.test.js
RUNS packages/expect/src/_tests__/matchers.test.js
RUNS packages/pretty-format/src/_tests__/Immutable.test.js
RUNS packages/expect/src/_tests__/spyMatchers.test.js
RUNS packages/jest-cli/src/_tests__/SearchSource.test.js
RUNS packages/jest-runtime/src/_tests__/script_transformer.test.js
RUNS packages/jest-cli/src/_tests__/watch.test.js
RUNS packages/jest-haste-map/src/crawlers/_tests__/watchman.test.js
RUNS packages/pretty-format/src/_tests__/react.test.js

Test Suites: 5 passed, 5 of 303 total
Tests:      332 passed, 332 total
```

Ao garantir que seus testes têm um estado global único, Jest pode executar testes em paralelo de forma confiável. Para tornar as coisas rápidas, Jest executa testes anteriores falharam primeiro e organiza novamente com base no quanto os arquivos de teste demoram.

## COBERTURA DE CÓDIGO

Crie relatórios de cobertura de código facilmente usando [--coverage](#). Sem necessidade adicional de configurações ou bibliotecas! Jest consegue coletar informações de cobertura de código de projetos inteiros, incluindo arquivos não testados.

```
~/d/p/j/e/timer $ yarn jest --coverage
yarn run v1.12.3
$ /Users/ortatherox/dev/projects/jest/jest/examples/timer/node_modules/.bin/jest --coverage
PASS __tests__/timer_game.test.js
PASS __tests__/infinite_timer_game.test.js
-----|-----|-----|-----|-----|-----|
File    | %Stmts | %Branch | %Funcs | %Lines | Uncovered Line #s |
-----|-----|-----|-----|-----|-----|
All files | 87.5 | 100 | 80 | 87.5 |
infiniteTimerGame.js | 80 | 100 | 66.67 | 80 | 12 |
timerGame.js | 100 | 100 | 100 | 100 |
-----|-----|-----|-----|-----|-----|
Test Suites: 2 passed, 2 total
Tests:      4 passed, 4 total
Snapshots:  0 total
Time:       0.878s, estimated 1s
Ran all test suites.
```

```
• toHaveBeenCalledWith two arguments
expect(jest.fn()).toHaveBeenCalledWith(...expected)
Expected: "last", 10
Received:
 1: "first", 1
 2: "second", 2
 3: "third", 3
Number of calls: 9
```

## SIMULAÇÕES FÁCEIS

Jest uses a custom resolver for imports in your tests, making it simple to mock any object outside of your test's scope. You can use mocked imports with the rich [Mock Functions API](#) to spy on function calls with readable test syntax.

# Jest: como instalar e configurar

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

- Para instalar, basta executar o comando acima no terminal;
- Se for um projeto React iniciado com *create-react-app*, o **Jest** já estará presente por padrão;
- Se for um projeto com *typescript*, será necessário também instalar o *babel-jest* e outras libs auxiliares:
  - *npm install --save-dev babel-jest @babel/core @babel/preset-env*
  - *npm install --save-dev @babel/preset-typescript*
  - e configurar o arquivo *babel.config.js*.

<https://jestjs.io/pt-BR/docs/getting-started#usando-typescript>

<https://jestjs.io/docs/testing-frameworks>

# Jest: como instalar e configurar

- Depois de instalado, o Jest já está pronto para uso. Contudo, para facilitar o seu uso, é interessante defini-lo como *runner* padrão no script de "*test*" do package.json, da seguinte forma:
  - Assim, ao rodar "*npm test*", o Jest irá agir e executar todos os testes presentes no projeto.

```
"scripts": {  
    "start": "nodemon -e js,json,ts,yaml src/index.ts",  
    "start:prod": "node build/index.js",  
    "build": "npx tsc",  
    "tsc:status": "tsc --diagnostics",  
    "test": "jest"  
},
```

# Jest: como instalar e configurar

```
PS C:\Users\julia\Documents\web_academy_uf\test\LojaVirtualWA\backend> npx jest --init

The following questions will help Jest to create a suitable configuration for your project

✓ Would you like to use Jest when running "test" script in "package.json"? ... yes
✓ Would you like to use Typescript for the configuration file? ... yes
✓ Choose the test environment that will be used for testing » node
✓ Do you want Jest to add coverage reports? ... yes
✓ Which provider should be used to instrument code for coverage? » v8
✓ Automatically clear mock calls, instances, contexts and results before every test? ... yes

📝 Modified C:\Users\julia\Documents\web_academy_uf\test\LojaVirtualWA\backend\package.json

📝 Configuration file created at C:\Users\julia\Documents\web_academy_uf\test\LojaVirtualWA\backend\jest.config.ts
PS C:\Users\julia\Documents\web_academy_uf\test\LojaVirtualWA\backend>
```

# Jest: como instalar e configurar

Vamos praticar?



1. Clonar repositório [https://github.com/julialuiza/myapp\\_test](https://github.com/julialuiza/myapp_test)
2. Instalar Jest utilizando npm
3. Inserir script para rodar testes no package.json
4. Conferir se script está rodando corretamente.
5. Resultado esperado no terminal:

```
@julialuiza → /workspaces/myapp_test (main) $ npm test  
> myapp_test@1.0.0 test  
> jest  
  
No tests found, exiting with code 1  
Run with `--passWithNoTests` to exit with code 0  
In /workspaces/myapp_test  
  5 files checked.  
  testMatch: **/_tests_/**/*.[jt]s?(x), **/?(*.)+(spec|test).[tj]s?(x) - 0 matches  
  testPathIgnorePatterns: /node_modules/ - 5 matches  
  testRegex: - 0 matches  
  Pattern: - 0 matches
```

# Cronograma: Aula 01

## Visão geral

- Por que é importante testar?
- Diferentes tipos de teste;
- Ferramentas disponíveis.

## Jest

- O que é e vantagens;
- Instalação utilizando npm;
- Configurações adicionais.

## Como escrever testes

- Sintaxe do Jest;
- Cobertura de código;
- Prática com testes para funções;
- Boas práticas.



ATENÇÃO

Dúvidas?

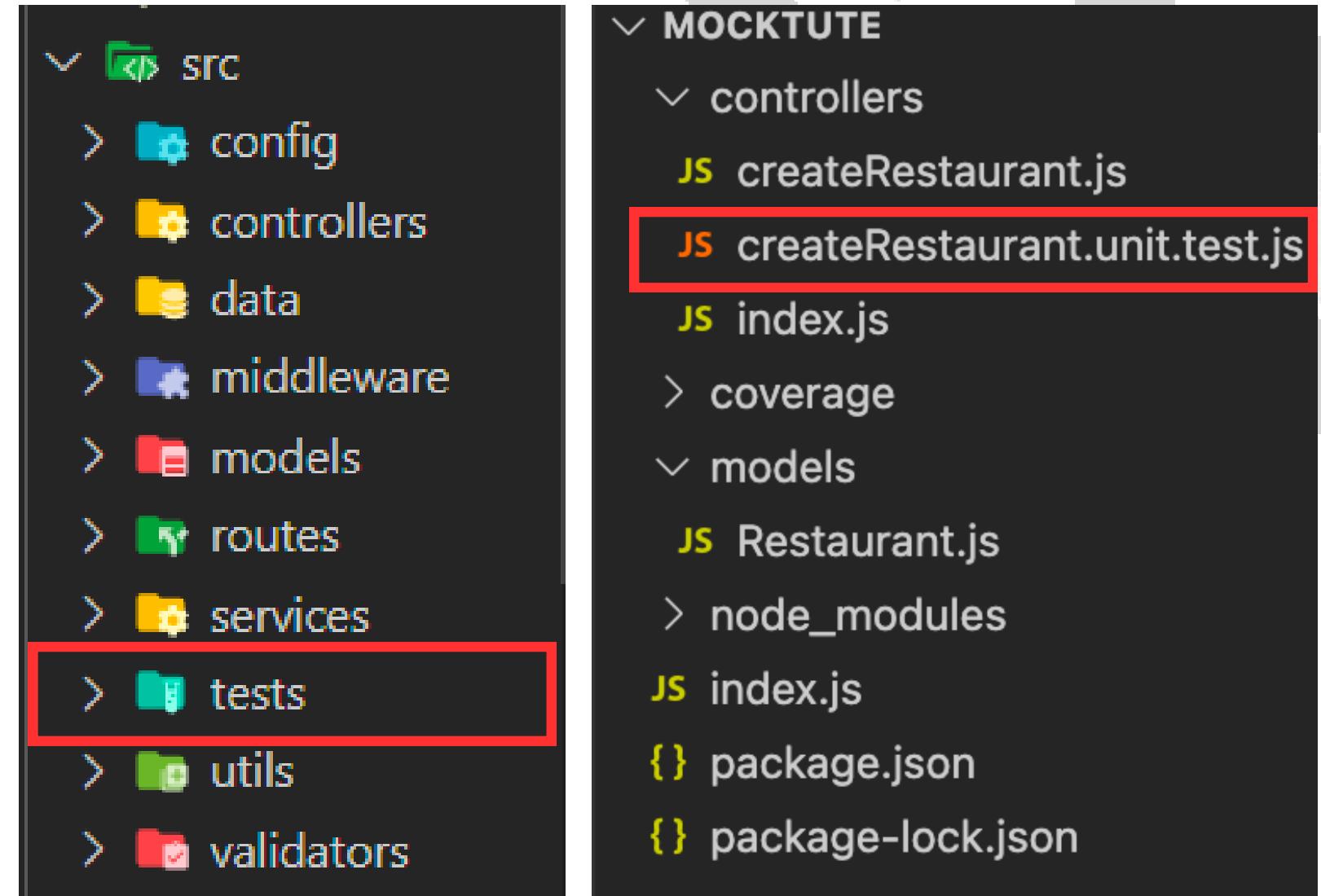
# Como escrever testes

- **Organização:**

- *Usualmente*, os arquivos de teste localizam-se perto dos módulos que estão sendo testados;
- Ou em uma pasta dedicada para os testes do projeto;
- No geral, segue-se padrão definido pelo time, empresa, etc.

- **Arquivo de teste:**

- *Usualmente*, cada arquivo de teste comporta testes para um contexto específico do projeto;
- Por padrão, costuma-se nomear com o mesmo nome do arquivo ou função que será testada;
  - Ex: itemCarrinho.js -> itemCarrinho.**test.js**
- No caso do **Jest**, para que o arquivo seja identificado como sendo de teste, deverá possuir a extensão `'.test'` ou estar dentro de uma pasta `__tests__`.



```
MOCKTUTE
├── controllers
│   ├── createRestaurant.js
│   └── createRestaurant.unit.test.js
├── index.js
└── models
    └── Restaurant.js
├── node_modules
└── package.json
└── package-lock.json

src
└── config
└── controllers
    └── index.js
    └── tests
        └── createRestaurant.unit.test.js
    └── data
    └── middleware
    └── models
    └── routes
    └── services
    └── utils
    └── validators
```

# Como escrever testes

---

- **Estrutura:**
  - O Jest dispõe de diversos recursos para estruturação dos testes, então **a estrutura final irá depender do cenário a ser testado;**
  - De modo geral, um teste possui:
    - Um ou mais blocos "**describe**" (agrupamento de testes)
    - Que comportam um ou mais blocos "**it**" ou "**test**" (o teste em si)

# Como escrever testes

```
const myBeverage = {  
  delicious: true,  
  sour: false,  
};  
  
describe('my beverage', () => {  
  test('is delicious', () => {  
    expect(myBeverage.delicious).toBeTruthy();  
  });  
  
  test('is not sour', () => {  
    expect(myBeverage.sour).toBeFalsy();  
  });  
});
```

```
describe('binaryStringToNumber', () => {  
  describe('given an invalid binary string', () => {  
    test('composed of non-numbers throws CustomError', () => {  
      expect(() => binaryStringToNumber('abc')).toThrow(CustomError);  
    });  
  
    test('with extra whitespace throws CustomError', () => {  
      expect(() => binaryStringToNumber(' 100')).toThrow(CustomError);  
    });  
  });  
  
  describe('given a valid binary string', () => {  
    test('returns the correct number', () => {  
      expect(binaryStringToNumber('100')).toBe(4);  
    });  
  });  
});
```

<https://jestjs.io/docs/api#describename-fn>

# Como escrever testes

---

- **Estrutura:**
  - O Jest dispõe de diversos recursos para estruturação dos testes, então **a estrutura final irá depender do cenário a ser testado;**
  - De modo geral, um teste possui:
    - Um ou mais blocos "**describe**" (agrupamento de testes)
    - Que comportam um ou mais blocos "**it**" ou "**test**" (o teste em si)
- Além disso, existem os blocos:
  - **beforeEach()** -> executado sempre antes de cada bloco it
  - **afterEach()** -> executado sempre após cada bloco it
  - **beforeAll()** -> executado uma vez antes de todos os blocos it existentes executarem
  - **afterAll( )** -> executado uma vez depois de todos os blocos it existentes executarem
  - Obs.: podem ser usados no topo do arquivo de teste ou dentro de cada bloco describe

# Como escrever testes

```
beforeEach(() => {
  initializeCityDatabase();
});

afterEach(() => {
  clearCityDatabase();
});

test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy();
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy();
});
```

```
beforeAll(() => {
  return initializeCityDatabase();
});

afterAll(() => {
  return clearCityDatabase();
});

test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy();
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy();
});
```

# Como escrever testes

- **Sintaxe:**
  - Para realizar o teste em si, utilizamos o "**expect**", "**modifiers**" e "**matchers**" que o Jest provê;
  - São muitos, então o ideal é estar sempre com a documentação ao lado durante o desenvolvimento dos testes.
- Expect
  - `expect(value)`
- Modifiers
  - `.not`
  - `.resolves`
  - `.rejects`
- Matchers
  - `.toBe(value)`
  - `.toHaveBeenCalled()`
  - `.toHaveBeenCalledTimes(number)`
  - `.toHaveBeenCalledWith(arg1, arg2, ...)`
  - `.toHaveBeenCalledWith(arg1, arg2, ...)`
  - `.toHaveBeenCalledWith(nthCall, arg1, arg2, ....)`
  - `.toHaveBeenCalled()`
  - `.toHaveBeenCalledTimes(number)`
  - `.toHaveBeenCalledWith(value)`

# Como escrever testes

- **Sintaxe:**

- Às vezes, mais de um matcher irá servir para a conferência do teste, mas vale conferir qual das opções é a mais semântica para o teste em questão:
  - Exemplo: **toBe(null)** e **toBeNull()** podem produzir o mesmo resultado, mas possuem semântica diferente.

```
function bloop() {  
  return null;  
}  
  
test('bloop returns null', () => {  
  expect(bloop()).toBeNull();  
});
```

<https://jestjs.io/pt-BR/docs/expect#tobenull>

# Jest Matchers

```
test('dois mais dois é quatro', () => {
  expect(2 + 2).toBe(4);
});
```

```
test('atribuição de objeto', () => {
  const data = {one: 1};
  data['two'] = 2;
  expect(data).toEqual({one: 1, two: 2});
});
```

```
test('dois mais dois', () => {
  const value = 2 + 2;
  expect(value).toBeGreaterThan(3);
  expect(value).toBeGreaterThanOrEqual(3.5);

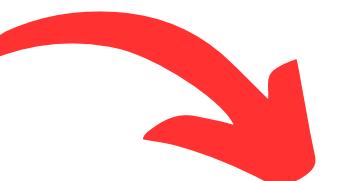
test('não existe I em team', () => {
  expect('team').not.toMatch(/I/);
});
```

```
test('resolves to lemon', async () => {
  await expect(Promise.resolve('lemon')).resolves.toBe('lemon');
  await expect(Promise.resolve('lemon')).resolves.not.toBe('octopus');
});
```

```
test('rejects to octopus', async () => {
  await expect(Promise.reject(new Error('octopus'))).rejectstoThrow('octopus');
});
```

- Matchers

- `.toBe(value)`
- `.toHaveBeenCalled()`
- `.toHaveBeenCalledTimes(number)`
- `.toHaveBeenCalledWith(arg1, arg2, ...)`
- `.toHaveBeenCalledWithLastCalledWith(arg1, arg2, ...)`
- `.toHaveBeenCalledWithNthCalledWith(nthCall, arg1, arg2, ....)`
- `.toHaveReturned()`
- `.toHaveReturnedTimes(number)`
- `.toHaveReturnedWith(value)`
- `.toHaveLastReturnedWith(value)`
- `.toHaveNthReturnedWith(nthCall, value)`
- `.toHaveLength(number)`



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

# Cobertura de código

- Pode ser incorporado em pipelines de CI/CD para manter uma cobertura mínima de código;
- Geralmente, usa-se a porcentagem mínima por volta de 80%, que é considerada boa.
- Para NPM:
  - **npm jest -- --coverage**

```
~/d/p/j/j/e/timer $ yarn jest --coverage
yarn run v1.12.3
$ /Users/ortatherox/dev/projects/jest/jest/examples/timer/node_modules/.bin/jest --coverage
PASS  __tests__/timer_game.test.js
PASS  __tests__/infinite_timer_game.test.js
-----|-----|-----|-----|-----|-----|
File           | %Stmts | %Branch | %Funcs | %Lines | Uncovered Line #s |
-----|-----|-----|-----|-----|-----|
All files     |   87.5 |    100 |     80 |   87.5 |          |
infiniteTimerGame.js |   80 |    100 | 66.67 |   80 | 12 |
timerGame.js  |   100 |    100 |   100 |   100 |          |
-----|-----|-----|-----|-----|-----|
Test Suites: 2 passed, 2 total
Tests:        4 passed, 4 total
Snapshots:   0 total
Time:         0.878s, estimated 1s
Ran all test suites.
```

# Como escrever testes

---

Em resumo, para escrever testes com Jest você deve:

- 1. Avaliar o alvo do teste.**
  - a. Se é uma função isolada, um arquivo com várias funções, etc;
- 2. Decidir como organizar seus testes** do projeto:
  - a. Podem ser todos em um diretório /tests, um teste por arquivo, etc;
- 3. Observar os principais casos de uso** do alvo a ser testado
  - a. No caso de uma função, por exemplo, observar quais são os seus parâmetros e retornos possíveis.
- 4. Consultar documentação do Jest** procurando pelos *matchers* mais semânticos para a construção do teste;
- 5. Implementar o teste em si, de forma organizada;**
- 6. Executar os testes e observar os resultados.**
  - a. Ao executar um teste com opção de *coverage*, o Jest informa dados interessantes sobre o que está sendo testado e se ainda há casos de uso não cobertos.
  - b. Sempre é possível que hajam resultados falso positivos, onde o teste está passando, mas não está funcionando corretamente.

# Como escrever testes: Exemplo

Considere a seguinte função abaixo e os exemplos de entrada e saída:

```
/**  
 * Extracts the first name from a full name string.  
 *  
 * @param {string} fullName - The full user name separated by blank spaces.  
 * @returns {string} - The first name extracted from the full name, or the name itself if no blank space is found.  
 */  
function firstName(fullName) {  
    const fullNameTrim = fullName.trim();  
    const blankSpace = fullNameTrim.indexOf(' ');  
  
    if (blankSpace === -1) return fullNameTrim;  
    else return fullNameTrim.slice(0, blankSpace);  
}  
  
console.log('firstName() outputs:');  
console.log(firstName(' Test')); // Output will be 'Test'  
console.log(firstName('Test T')); // Output will be 'Test'  
console.log(firstName('Test T T T')); // Output will be 'Test'  
console.log(firstName(' Test T    ')); // Output will be 'Test'
```

```
    julialuiza, 4 months ago | 1 author (julialuiza)
1  const {
2    |   firstName,
3    } = require("../validations");
4
5  describe("firstName()", () => {
6    it("should return the first name when the full name is given", () => {
7      const fullName = "John Doe Etc";
8      const result = firstName(fullName);
9      expect(result).toBe("John");
10   });
11
12  it("should return the same name when no blank space is found", () => {
13    const name = "Alice";
14    const result = firstName(name);
15    expect(result).toBe(name);
16  });
17
18  it("should return the first name correctly when theres blank space in the start", () => {
19    const name = " Alice Test";
20    const result = firstName(name);
21    expect(result).toBe("Alice");
22  });
23
24  it("should return the first name correctly when theres blank space in the end", () => {
25    const name = "Alice Test ";
26    const result = firstName(name);
27    expect(result).toBe("Alice");
28  });
29});
```

Um possível grupo de **testes unitários** seria o seguinte:

# Como escrever testes: Exemplo

Ao executar "npm test" no terminal, o resultado será o seguinte:

```
[> myapp_test@1.0.0 test /Users/juliaconceicao/Documents/side_projects/myapp_test
> jest

FAIL  src/utils/tests/validations.test.js
firstName()
  ✘ should return the first name when the full name is given (7 ms)
  ✓ should return the same name when no blank space is found
  ✘ should return the first name correctly when theres blank space in the start (1 ms)
  ✘ should return the first name correctly when theres blank space in the end

● firstName() > should return the first name when the full name is given

expect(received).toBe(expected) // Object.is equality

Expected: "John"
Received: "John Doe"
```

Qual o 'bug' da função anterior?

# Como escrever testes: Exemplo

E ao executar "npm test -- --coverage", o resultado será o mesmo que o anterior porém com uma tabela de informações mais detalhadas:

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Line #s
All files	33.33	50	33.33	33.33	
validations.js	33.33	50	33.33	33.33	23-55

```
Test Suites: 1 failed, 1 total
Tests:       3 failed, 1 passed, 4 total
Snapshots:   0 total
Time:        0.714 s, estimated 1 s
Ran all test suites.
npm ERR! Test failed. See above for more details.
```

# Como escrever testes: Exemplo

Após corrigir a função e executar "npm test -- --coverage" no terminal novamente, agora o resultado será o seguinte:

```
> myapp_test@1.0.0 test /Users/juliaconceicao/Documents/side_projects/myapp_test
> jest "--coverage"

PASS  src/utils/tests/validations.test.js
  firstName()
    ✓ should return the first name when the full name is given (3 ms)
    ✓ should return the same name when no blank space is found
    ✓ should return the first name correctly when theres blank space in the start (1 ms)
    ✓ should return the first name correctly when theres blank space in the end

-----|-----|-----|-----|-----|-----|
File      | % Stmt | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----|
All files | 37.5 | 50 | 33.33 | 38.46 |
validations.js | 37.5 | 50 | 33.33 | 38.46 | 25-57
-----|-----|-----|-----|-----|-----|
Test Suites: 1 passed, 1 total
Tests: 4 passed, 4 total
Snapshots: 0 total
Time: 0.71 s, estimated 1 s
Ran all test suites.
```

# Como escrever testes: Trabalho Prático 01

Vamos praticar?



1. Voltar ao projeto clonado [https://github.com/julialuiza/myapp\\_test](https://github.com/julialuiza/myapp_test)
2. Criar arquivo de teste para as funções do arquivo src/utils/validations.js
  - a. Organizar da forma que preferir
    - i. em arquivo único ou separado para cada função;
    - ii. utilizando describes e it, somente it, mais de um describe, etc.
3. Considerando os exemplos mostrados e a documentação de matchers do Jest, criar **testes unitários** para cada função
4. Identificar, por meio dos testes, bugs existentes nas funções, e corrigi-las;
5. Obter os testes passando 100%.

# Como escrever testes com Mock

## Mock:

- Às vezes, durante a construção de um **teste unitário**, precisamos simular o comportamento de serviços dos quais nosso teste dependem, mas que **não são** o foco do teste em si. Por exemplo, considere uma função **verificaBairro(CEP: String)** que:
  - Recebe um CEP como argumento;
  - Consulta os detalhes de endereço utilizando a API do viaCEP e retorna informações de rua, bairro, cidade, etc;
  - E verifica se está na lista de bairros atendidos pela loja, retornando *true* ou *false*;
- Nesse caso, o que verdadeiramente interessa para nosso teste é se nossa função está conferindo corretamente a lista de bairros autorizados e retornando *true* ou *false*
- Mas para isso, precisamos obter as informações de retorno da API do viaCEP;
- E agora?



# Como escrever testes com Mock

## Mock:

- Às vezes, durante a construção de um **teste unitário**, precisamos simular o comportamento de serviços dos quais nosso teste dependem, mas que **não são** o foco do teste em si. Por exemplo, considere uma função **verificaBairro(CEP: String)** que:
  - Recebe um CEP como argumento;
  - Consulta os detalhes de endereço utilizando a API do viaCEP e retorna informações de rua, bairro, cidade, etc;
  - E verifica se está na lista de bairros atendidos pela loja, retornando *true* ou *false*;
- Nesse caso, o que verdadeiramente interessa para nosso teste é se nossa função está conferindo corretamente a lista de bairros autorizados e retornando *true* ou *false*.
- ~~Mas para isso~~



## Funções de Simulação

Funções de simulação ( mocks em inglês ) permitem que você teste os links entre códigos, apagando a implementação real de uma função, capturando chamadas para a função (e os parâmetros passados nessas chamadas), capturar instâncias do construtor de funções quando instanciado com `new`, e permitindo configuração em tempo de teste de valores de retorno.

# Mock: Funções de simulação

Em resumo, **Mocks** no Jest funcionam da seguinte forma:

- É possível sobreescrever a definição inteira de um módulo, um método específico ou apenas determinar qual será o retorno, sem necessariamente especificar uma nova implementação;
- Por conta dessa flexibilidade, torna-se possível realizar testes que dependem de módulos de terceiros, por exemplo;
- Contudo, é importante atentar-se ao fato de que o **funcionamento no teste com mocks não garante o funcionamento dos módulos na 'vida real'**, em ambiente de produção;
- Portanto, **mocks são principalmente úteis para isolar o código que está sendo testado**, permitindo que você se concentre em testar uma parte específica do seu código sem depender diretamente de funcionalidades externas.

# Como escrever testes com Mock

```
import axios from 'axios';

class Users {
  static all() {
    return axios.get('/users.json').then(resp => resp.data);
  }
}

export default Users;
```

<https://jestjs.io/pt-BR/docs/mock-functions#simulando-m%C3%B3dulos>

users.test.js

```
import axios from 'axios';
import Users from './users';

jest.mock('axios');

test('deve buscar os usuários', () => {
  const users = [{name: 'Bob'}];
  const resp = {data: users};
  axios.get.mockResolvedValue(resp);

  // ou você poderá usar o código abaixo dependendo do seu caso de uso:
  // axios.get.mockImplementation(() => Promise.resolve(resp))

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

# Como escrever testes com mock: Exemplo

Vamos praticar?



1. Voltar ao projeto clonado [https://github.com/julialuiza/myapp\\_test](https://github.com/julialuiza/myapp_test)
2. Criar arquivo de teste para a função do arquivo `src/utils/cep.js` utilizando Mocks do Jest para simular a chamada ao módulo `node-correios`.
3. Dicas:
  - a. É possível utilizar **mockImplementation** do Jest para sobreescrivar a função `consultaCEP()` do `node-correios`;
  - b. Por ser uma função assíncrona, é necessário considerar que o mock precisa retornar *promises*;
  - c. Da mesma forma, no `expect()` dos testes, é necessário considerar que as promises talvez ainda precisem ser resolvidas (`rejects/resolves`);
  - d. Consultar a implementação da função `consultaCEP()` original:  
<https://github.com/vitorleal/node-correios/blob/master/lib/correios.js#L59>

# Boas práticas em testes unitários

- **Nomear bem os testes:** é essencial nomear de forma semântica os blocos de testes e os testes em si; dessa forma, quando algum teste falhar será fácil identificar qual parte do código está com problemas.
- **Rápido:** não é incomum projetos maduros terem milhares de testes de unidade. A execução dos testes de unidade deve demorar pouco tempo.
- **Isolado:** testes de unidade são autônomos, podem ser executados em isolamento e geralmente não têm dependências em nenhum fator externo, como um sistema de arquivos ou o banco de dados. Caso seja necessário resolver dependências externas, utilize recursos como mock.
- **Repetível:** a execução de um teste de unidade deve ser consistente com seus resultados, ou seja, sempre retornará o mesmo resultado se você não alterar nada entre execuções.
- **Não ignore os testes.**

<https://blog.mandic.com.br/artigos/10-dicas-para-escrever-bons-testes-de-unidade/>

<https://learn.microsoft.com/pt-br/dotnet/core/testing/unit-testing-best-practices>

# Obrigada!

Dúvidas?

- Slack
- Email: [jlslc@icomp.ufam.edu.br](mailto:jlslc@icomp.ufam.edu.br)

