# Testes Unitários

Técnico em Desenvolvimento de Sistemas

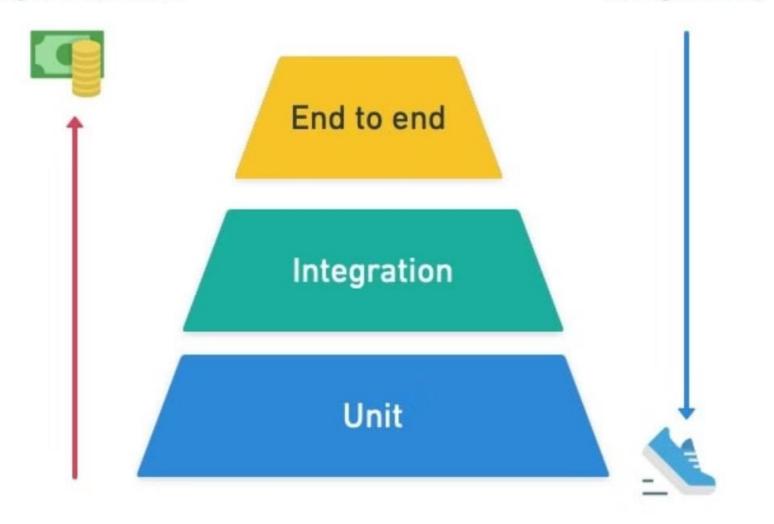
Prof. Daniel Ferreira





- More effort
- Slower
- Higher maintenace

- Faster
- Low effort
- More granularity



## Entregas da aula passada - eslint

https://github.com/daniellferreira/aula-publicacao-teste-apps-web

Pasta 2-static-code-analysis

- 1. Configurar o arquivo eslint.config.js para encontrar esses problemas e executar o comando npm run lint
- 2. Criar script lint:fix no package.json para corrigir automaticamente
- 3. Configurar o editor para corrigir automaticamente ao salvar
- 4. Desabilitar o linter um arquivo inteiro de cada vez para ter os resultados por arquivo
- 5. Desabilitar por linha os comandos console.log
- 6. DESAFIO: criar um arquivo ex-challenge.js e um arquivo de configs com o maior número de regras e problemas para consertar exemplificados, quem configurar e exemplificar a maior quantidade de regras irá ganhar um prêmio
- => Ajuda para subir no repositório?

### Testes Unitários

#### 1 Conceito

Os testes unitários são responsáveis por verificar o funcionamento de unidades individuais de código, como funções ou métodos. O objetivo é garantir que cada componente funcione corretamente de forma isolada.

### 3 Benefícios

Ao implementar testes unitários, os desenvolvedores podem ter mais confiança na integridade do código, facilitando futuras manutenções e melhorias. Isso contribui para a entrega de software de alta qualidade.

### 2 Objetivos

Facilitar a identificação de erros específicos e tornar o processo de depuração mais eficiente. Além disso, os testes unitários servem como documentação viva do comportamento esperado do código.

### 4 Isolamento e Mocks

Para testar unidades de código de forma isolada, é comum o uso de mocks e stubs para simular as dependências. Isso permite focar nos testes da unidade em questão, sem precisar lidar com interações complexas entre componentes.

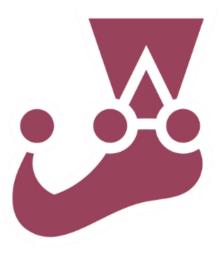
## Ferramentas para Testes Unitários em Node.js



O Mocha é um framework flexível e extensível para testes unitários em Node.js. Ele permite uma estrutura modular e customizável, facilitando a organização e execução dos testes.



O Chai é uma biblioteca de asserções que pode ser usada com diversos frameworks de testes. Ela oferece uma sintaxe clara e expressiva, tornando as asserções mais legíveis e fáceis de entender.



O Jest é conhecido por sua simplicidade e funcionalidades integradas, como mocks automáticos. Essa ferramenta facilita a configuração e a escrita de testes unitários, acelerando o desenvolvimento.

A escolha da ferramenta ideal para testes unitários em Node.js depende das necessidades específicas do projeto e das preferências da equipe de desenvolvimento. Cada uma dessas ferramentas possui suas próprias características e vantagens, oferecendo soluções flexíveis e eficientes para a implementação de testes unitários.

## Escrevendo Testes Unitários em Node.js com Jest

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

```
> node_modules

> src
> 1-math
```

```
function multiplicar(a, b) {
   return a * b;
}

module.exports = { multiplicar };
```

math.js

```
const { multiplicar } = require('./math');

test('multiplicar dois números positivos', () => {
   expect(multiplicar(2, 3)).toBe(6);
});

test('multiplicar por zero', () => {
   expect(multiplicar(5, 0)).toBe(0);
});
```

math.test.js

```
"scripts": {
   "test": "jest"
},
```

package.json

```
npm run test
```

## Escrevendo Testes Unitários em Node.js com Jest

```
> node_modules

> src

> 1-math

> 2-email

• .gitignore

{} package-lock.json

{} package.json
```

```
const emailClient = require('./emailClient');
function enviarEmail(destinatario, mensagem) {
   const remetente = "email@email.com"
   return emailClient.send(remetente, destinatario, mensagem);
}
module.exports = { enviarEmail };
```

emailService.js

```
function send(remetente, destinatario, mensagem) {
    // chama API para envio de email
}
module.exports = { send };
```

emailClient.js

```
const emailClient = require('./emailClient')
const { enviarEmail } = require('./emailService')

jest.mock('./emailClient');

test('envia o email com os parametros corretos', () => {
    enviarEmail('destination@email.com', 'Olá Mundo');
    expect(emailClient.send).toHaveBeenCalledWith('email@email.com', 'destination@email.com', 'Olá Mundo');
    expect(emailClient.send).toHaveBeenCalledTimes(1);
})
```

npm run test

### Melhores Práticas em Testes Unitários

### Testes Pequenos e Focados

Ao escrever testes unitários, é importante mantê-los pequenos e focados em um único aspecto do código. Dessa forma, fica mais fácil identificar problemas e os testes ficam mais legíveis e compreensíveis.

#### Isolamento dos Testes

Certifique-se de que os testes sejam independentes entre si, evitando que a falha de um afete os outros. Isso ajuda a manter a integridade e a confiabilidade do conjunto de testes.

### Nomeação Descritiva

Nomeie os testes de maneira clara e descritiva, de modo que outros desenvolvedores entendam facilmente o que está sendo verificado. Isso facilita a manutenção e a atualização dos testes ao longo do desenvolvimento.

#### Uso de Mocks e Stubs

Utilize mocks e stubs quando necessário para isolar a unidade de código sendo testada, especialmente quando ela depende de recursos externos ou de estado compartilhado. Isso ajuda a garantir que os testes sejam focados e reproduzíveis.

### Exercício Prático: Palíndromo

- Implementar a função isPalindrome(string)
- Implementar os requisitos abaixo e testar:
  - Reconhece palíndromos em strings simples
  - Reconhece palíndromos com letras maiúsculas e minúsculas
  - Reconhece palíndromos com espaços e pontuação
  - Função se previne contra strings vazias, nulas, etc...
  - Informa quando a string não é um palíndromo

DICA: implementar como TDD (primeiro teste, depois implementa)

```
"Após a sopa." "A torre da derrota"
"Socorram-me Subi no onibus em Marrocos".
"Lá tem metal."
                       # reviver
                Ana
                       # rir
  Luz azul
                      # rodador
  "Ame o poema"
"A cara rajada da jararaca"
                        "A sacada da casa."
```