

Uma conversa (nem tão) séria sobre testes

Oi :)



Gabriel Ramos

Desenvolvedor @ Pluto.TV | Mentor @ Laboratória

Já fiz bugs em vários lugares e quebrei alguns sistemas por aí.

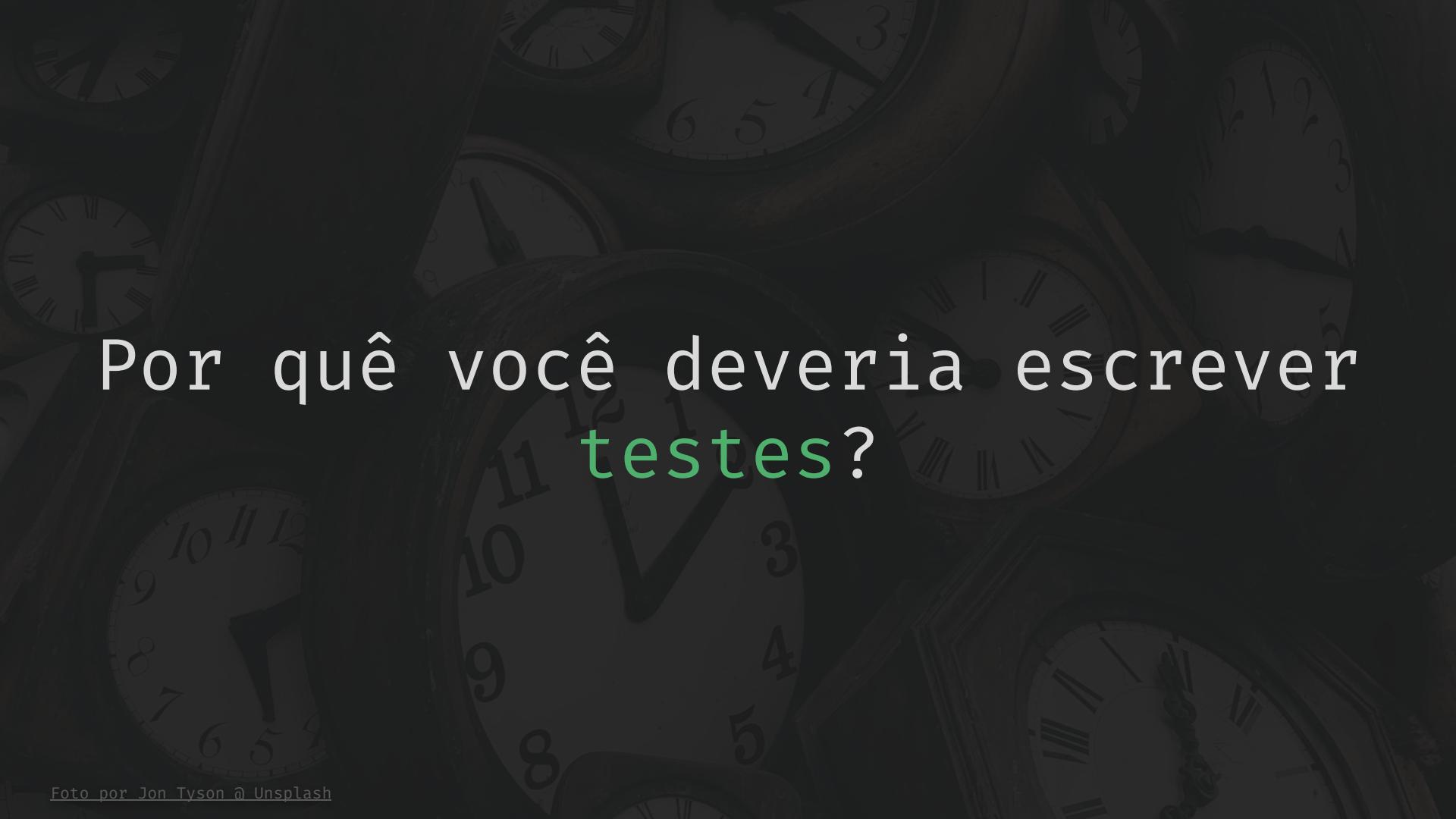
Desenvolvedor de gastrite, fotógrafo por hobby e "o doido dos gatos" (só 3).

@gabrieluizramos

gabrieluizramos.com.br

Conversar sobre o quê, maluco?

- Por quê escrever testes?
- O que são, onde vivem e do que se alimentam?
- Alguns dos tipos de testes
 - Testes unitários
 - Testes de integração
 - Testes de carga
 - Testes de regressão (visual)
 - Testes de ponta-a-ponta
- Como escrever um código testável
- Profissões e algumas ferramentas



Por quê você deveria escrever
testes?

Vamos imaginar uma cena ...

Você trabalha em um e-commerce e testava todo fluxo de pagamento manualmente. Até então, só era possível pagar com boleto ou cartão de débito ou crédito. Você precisa então simular um usuário:

- Comprando no boleto ("boletando")
- Comprando no débito
- Comprando no crédito

Nas próximas sprints seu time precisará desenvolver a funcionalidade de salvar um cartão de crédito para reutilizar em compras futuras. Após esse trabalho, você vai precisar testar um usuário:

- Comprando no boleto
- Comprando no débito
- Comprando no crédito sem usar um cartão salvo
- Comprando no crédito com um cartão salvo

Testes manuais tendem a
acompanhar o crescimento do
produto que você desenvolve.

Eu não sei vocês, mas se eu faço uma tarefa umas 3 ou 4 vezes, já dá uma canseira ...

As diversas formas de encarar

"Pra que **perder tempo** com isso?"

X

"Eu não quero meu celular apitando **quando
isso quebrar.**"

As diversas formas de encarar

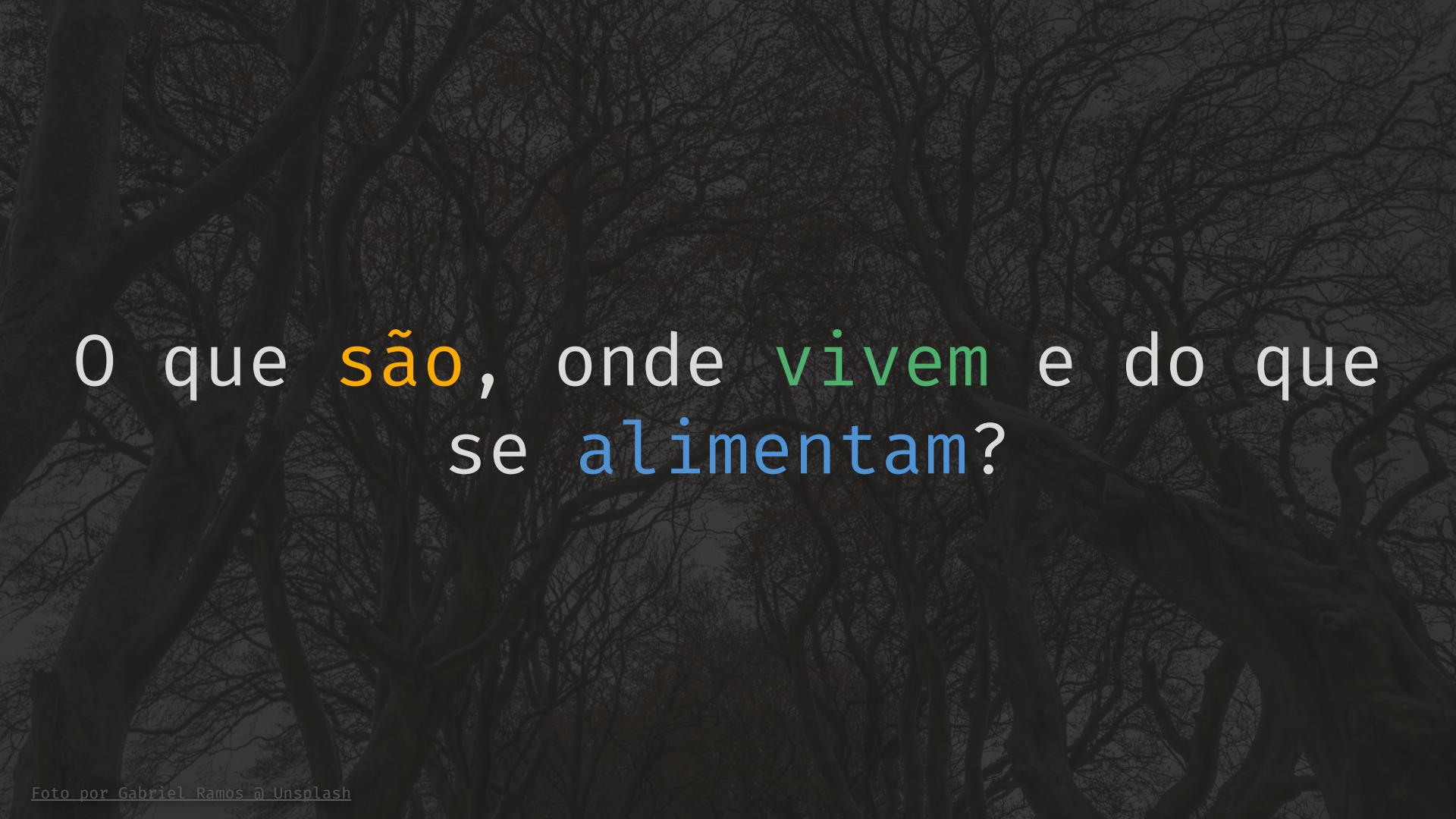
"Eu escrevo código bom, não preciso
disso."

X

"Eu confio nos testes que eu escrevi."

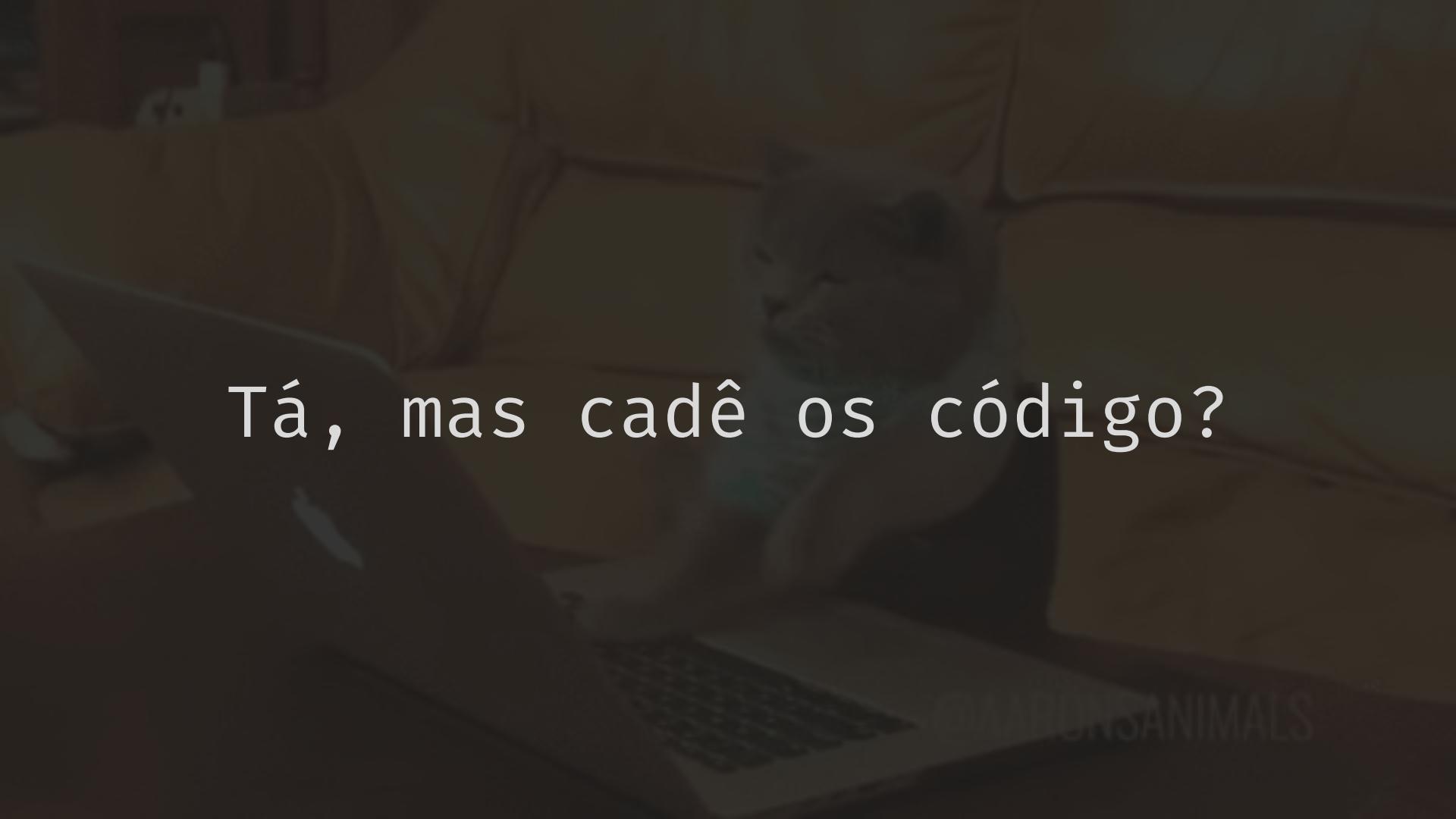
Investir tempo em testes automatizados vai **economizar muito mais tempo** a longo prazo.

E servem até como uma pequena documentação quando você começa a trabalhar em um projeto novo :)



O que são, onde vivem e do que
se alimentam?

Testes automatizados, no fim
do dia, podem ser tão simples
quanto um **if**

A dark, low-light photograph of a person sitting at a desk. The person is facing a laptop computer, which is open and visible in the lower right foreground. The background is mostly obscured by deep shadows, but some faint text is visible on the wall behind them.

Tá , mas cadê os código?

Como é a cara de um teste?

```
● ● ●  
1  function soma (a, b) {  
2      return a - b;  
3  }  
4  
5  const a = 1;  
6  const b = 1;  
7  const esperado = 2;  
8  const resultado = soma(a, b);  
9  
10 if (resultado !== esperado) {  
11     throw new Error("Soma tá zuada, xamps");  
12 }
```

Bom, depende... Podem sem bem simples

Como é a cara de um teste?

```
● ● ●  
1  function soma(a, b) {  
2      return a - b;  
3  }  
4  
5  
6  test('soma', () => {  
7      const a = 1;  
8      const b = 2;  
9      const esperado = 2;  
10     const resultado = soma(a, b);  
11  
12     expect(resultado).toEqual(esperado);  
13  });
```

Mas podem ser mais rebuscadinhos

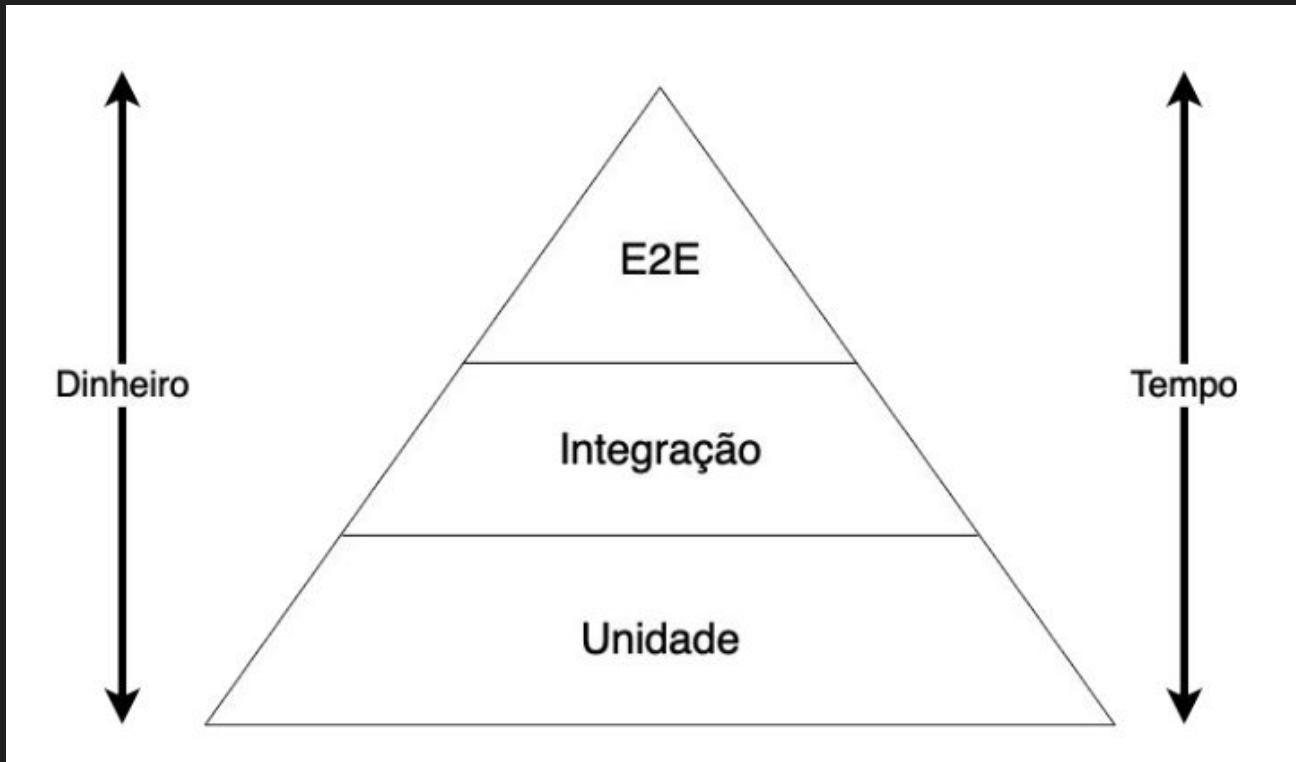
Nossa automação não é completa
se for **executada de forma manual**

Desenvolve → **Sobe** → **Executa testes** → **Produção**

Existem diversas ferramentas de integração e entrega
contínua que, por si só, seriam uma conversa a parte

Alguns tipos de testes

Você já viu essa pirâmide?



E esse troféu?



análise estática

Análise estática tem como objetivo validar seu código até mesmo em tempo de desenvolvimento, indicando possíveis variáveis esquecidas, métodos e funções não utilizadas ou inexistentes.

Geralmente, ferramentas que analisam seu código de forma estática dão algumas dicas até mesmo no seu editor/IDE enquanto você escreve.

```
const idade = 24;
const novaIdade = idad + 1;

function soma(a, b) {
    return a + b;
}

function subtracao(a, b) {
    return a - b;
}
```

```
const idade = 24;  
const novaIdade = idad + 1;
```

```
function      function subtracao(a: any, b: any): number  
  return      'subtracao' is defined but never used. eslint(no-unused-vars)  
}  
          Peek Problem (⌞F8) Quick Fix... (⌘.)  
function subtracao(a, b) {  
  return a - b;  
}
```

```
/Users/gabrieluizramos/Development/personal/javascriptassertivo.com.br/exemplos  
1:7  error  'idade' is assigned a value but never used      no-unused-vars  
2:7  error  'novaIdade' is assigned a value but never used  no-unused-vars  
2:19 error  'idad' is not defined                         no-undef
```

unidade

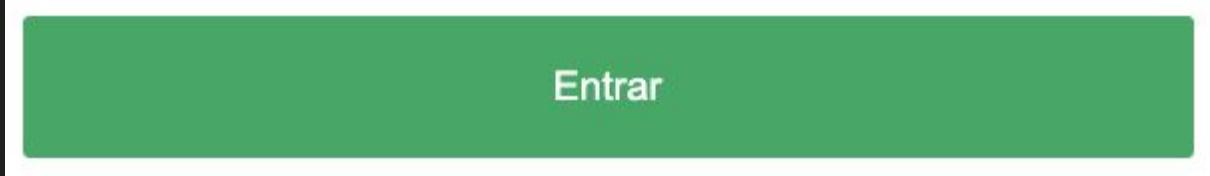
Visam testar pequenos pedaços do seu código,
isolados como uma simples unidade.

A própria palavra **unidade** pode ser algo difícil
de entender, então vamos ver alguns cenários.

Unidade pode ser uma **função**

```
1 // funções a serem testadas
2 const somaHorasExtras = (salario, valorHorasExtras) => {
3     return salario + valorHorasExtras;
4 };
5
6 // variáveis com os valores
7 let esperado = 10;
8 let retorno = somaHorasExtras(5, 5);
9
10 // testa a função somaHorasExtras
11 if (esperado === retorno) {
12     console.log(`✅ Teste passou`);
13 } else {
14     console.error(`🔴 Uh, deu ruim...`);
15 }
```

Unidade pode ser um **componente**



Entrar

Unidade pode ser um **componente**

```
● ● ●  
1 it('Renderiza um botão corretamente', () => {  
2   renderWithTheme(<Button>conteudo</Button>);  
3   const button = screen.getByRole('button');  
4   expect(button).toBeInTheDocument();  
5   expect(button).toMatchSnapshot();  
6 });  
7  
8 it('Executa ações de click ao receber a função por prop', () => {  
9   const onClick = jest.fn();  
10  renderWithTheme(<Button onClick={onClick}>conteudo</Button>);  
11  const button = screen.getByRole('button');  
12  
13  userEvent.click(button);  
14  
15  expect(onClick).toHaveBeenCalledTimes(1);  
16});
```

É geralmente nos testes unitários que lidamos com simulações de eventos e de acontecimentos da nossa aplicação, mais conhecidas como "**mocks**".

Usamos essas simulações para evitar que operações reais (uma operação de compra, salvar dados em um banco, uma requisição HTTP) aconteça ao longo dos nossos testes.

Um **mock** tem mais ou menos essa carinha



```
1 it('Executa ações de click ao receber a função por prop', () => {
2   const onClick = jest.fn();
3   renderWithTheme(<Button onClick={onClick}>conteudo</Button>);
4   const button = screen.getByRole('button');
5
6   userEvent.click(button);
7
8   expect(onClick).toHaveBeenCalledTimes(1);
9 });
```

integração

Visam testar as diversas unidades da sua aplicação funcionando de forma conjunta.

Assim como **unidade, diversas unidades em conjunto** pode ser algo subjetivo de entender.

Pode ser uma **tela inteira**, com diversos componentes

The image shows a screenshot of a web application's login page. The title 'JavaScript Assertivo' is displayed prominently at the top. Below it, a text instruction reads 'Preencha suas informações para acessar a área logada'. There are two input fields: one for 'usuario' and one for 'sua senha super secreta', which includes an eye icon for password visibility. A large green button labeled 'Entrar' is at the bottom.

JavaScript Assertivo

Preencha suas informações para acessar a área logada

usuario

sua senha super secreta

Entrar

Pode ser uma **tela inteira**, com diversos componentes



```
1 it('Exibe mensagem de erro ao realizar login com dados inválidos', async () => {
2   auth.logIn.mockRejectedValueOnce();
3
4   renderWithProviders(<LoginPage />);
5
6   userEvent.type(screen.getByPlaceholderText('usuario'), 'foo');
7   userEvent.type(screen.getByPlaceholderText('sua senha super secreta'), 'bar');
8   userEvent.click(screen.getByText('Entrar'));
9
10  expect(await screen.findByText(MESSAGES.AUTHENTICATE.ERROR)).toBeInTheDocument();
11  expect(Redirect).not.toHaveBeenCalled();
12});
```

Pode ser uma **API**, com suas rotas



```
1 let server;
2 beforeAll(() => {
3   server = app();
4 });
5
6 let client;
7 beforeEach(() => {
8   client = clientHTTP.create(server);
9 });
10
11 afterAll(() => {
12   server.close();
13 });
14
15 it('Não consegue logar usuário com dados inválidos', async () => {
16   const { userName: username, password } = createUser();
17   const user = { username, password };
18   const { response } = await client.post('/auth/login', user);
19
20   expect(response.status).toEqual(404);
21 });
```

carga

Visam indicar como sua aplicação se comporta com uma determinada carga de acesso, identificando potenciais problemas de performance e pontos de melhoria.

```
● ○ ●  
1 config:  
2   target: http://localhost:8080/api  
3   phases:  
4     - duration: 15  
5       arrivalRate: 50  
6  
7 scenarios:  
8   - flow:  
9     - post:  
10       url: /auth/login  
11       json:  
12         username: admin  
13         password: admin  
14
```

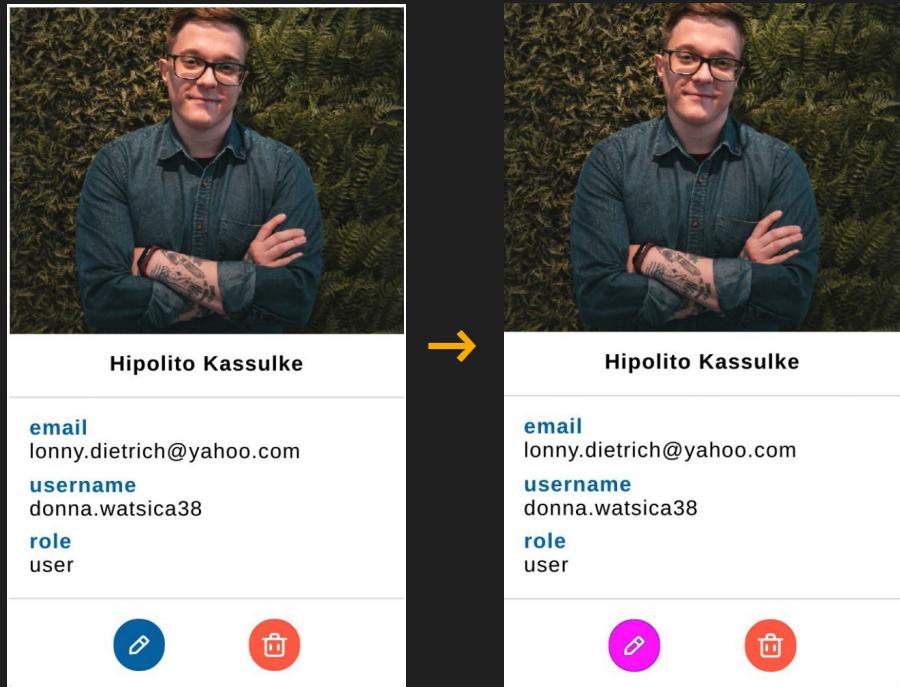
```
All virtual users finished
Summary report @ 20:18:11(-0300) 2021-03-06
Scenarios launched: 750
Scenarios completed: 750
Requests completed: 750
Mean response/sec: 48.48
Response time (msec):
  min: 1.6
  max: 39.9
  median: 2.6
  p95: 5
  p99: 17.6
Scenario counts:
  0: 750 (100%)
Codes:
```

Geralmente sendo executado em ambientes próximos aos ambientes onde sua aplicação vai ser acessada por usuários reais.

regressão

Visam garantir que, ao modificar um código um funcionalidade, o restante da sua aplicação continuará funcionando sem mudanças não esperadas.

Em regressões visuais isso fica bem claro.



Baseline

This is a post title!



Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

Change

This is a post title!



Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

Diff

This is a post title!

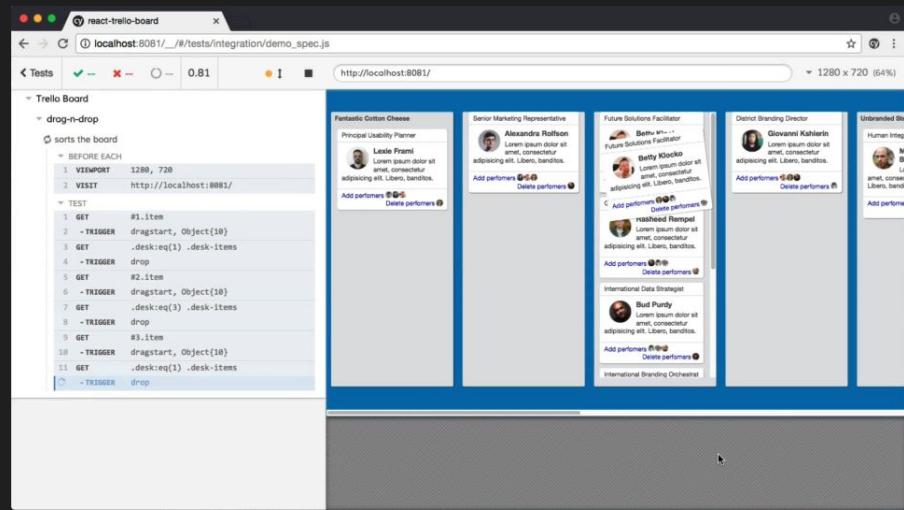


LOREM IPSUM IS SIMPLY DUMMY TEXT OF THE PRINTING AND TYPESETTING INDUSTRY. LOREM IPSUM HAS BEEN THE INDUSTRY'S STANDARD DUMMY TEXT EVER SINCE THE 1500S, WHEN AN UNKNOWN PRINTER TOOK A GALLEY OF TYPE AND SCRAMBLED IT TO MAKE A TYPE SPECIMEN BOOK.

ponta-a-ponta

Simulam o comportamento da sua aplicação por completo (por isso o nome).

Geralmente utilizam alguns motores de navegador e realizam o passo a passo de uma pessoa interagindo com seu software.



A photograph of a person walking up a large set of wide, light-colored stone steps. The steps lead upwards through a series of arches, creating a perspective that converges towards the top right of the frame. The person is seen from behind, wearing a dark cap and a striped shirt. The overall atmosphere is architectural and minimalist.

Como escrever um bom código **testável**?

Não existe fórmula secreta

Para escrever código testável de forma clara, é preciso praticar, errar, escrever código ruim, escrever teste ruim e ir se acostumando. Só com a prática que os acertos chegam.

Mas, algumas coisinhas sempre podem **ajudar**:

- Ter uma separação de responsabilidades clara entre as camadas da sua aplicação
- Ler bases de código que já possuem testes
- Começar com pequenos exercícios (soma, calculadoras, coisas simples) até acostumar
- Entender que teste é só código

O que eu devo **testar**?

Também depende muito do seu cenário.

Geralmente as partes mais críticas da sua aplicação são sempre as mais preocupantes, começar por elas pode ser uma boa. Se você está num projeto sem testes, essas partes críticas são boas candidatas para um primeiro passo.

Caso você esteja começando um projeto novo, pode já começar escrevendo testes também. Dessa forma, você garante que toda funcionalidade ou componente está devidamente **coberto por testes**.

Cobertura?

All files operacoes.js

80% Statements 4/5 100% Branches 0/0 50% Functions 1/2 80% Lines 4/5

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

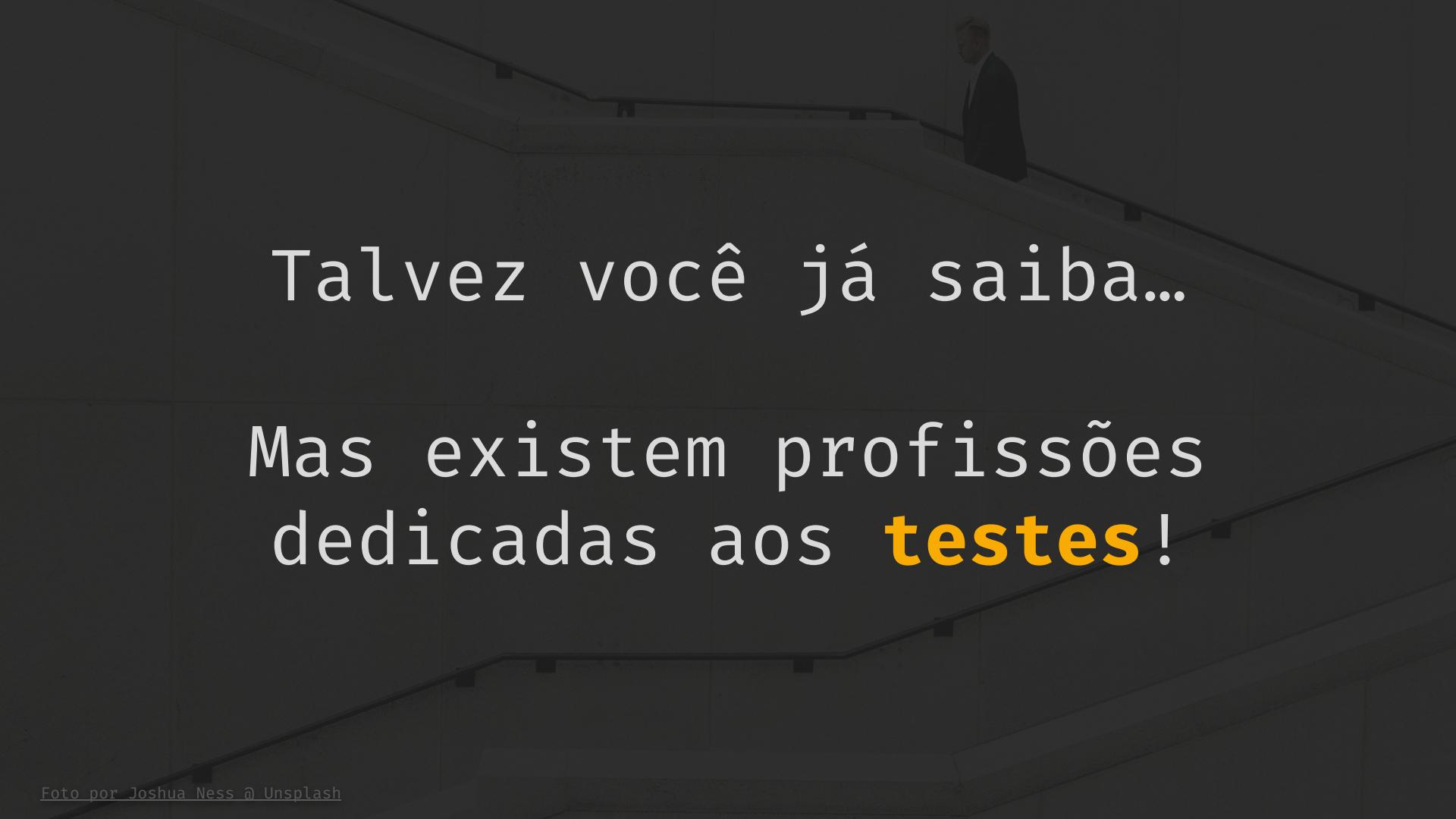
```
1 1x const somaHorasExtras = (salario, valorHorasExtras) => {
2 1x   return salario + valorHorasExtras;
3 };
4
5 1x const calculaDesconto = (salario, descontos) => {
6   return salario - descontos;
7 };
8
9 1x module.exports = {
10   somaHorasExtras, calculaDesconto
11 }
12
```

Quantos %?

Ferramentas de teste também analisam seu código para validar que trechos são "cobertos" (executados) pelos seus testes e geram alguns relatórios com algumas porcentagens.

Uma alta cobertura de teste não quer dizer que seu código não possui erros ou está completamente "blindado" (é até fácil driblar essas ferramentas).

Vale a pena decidir uma porcentagem confortável. Sempre tem um trecho de código que é difícil de ser testado ou que o esforço para isso é muito maior do que a confiança que seu teste vai gerar.

A dark, slightly grainy photograph of a man in a dark suit and tie standing on a modern staircase with black railings. He is looking down at the steps. The background is a bright, possibly overexposed area.

Talvez você já saiba...

Mas existem profissões
dedicadas aos **testes**!

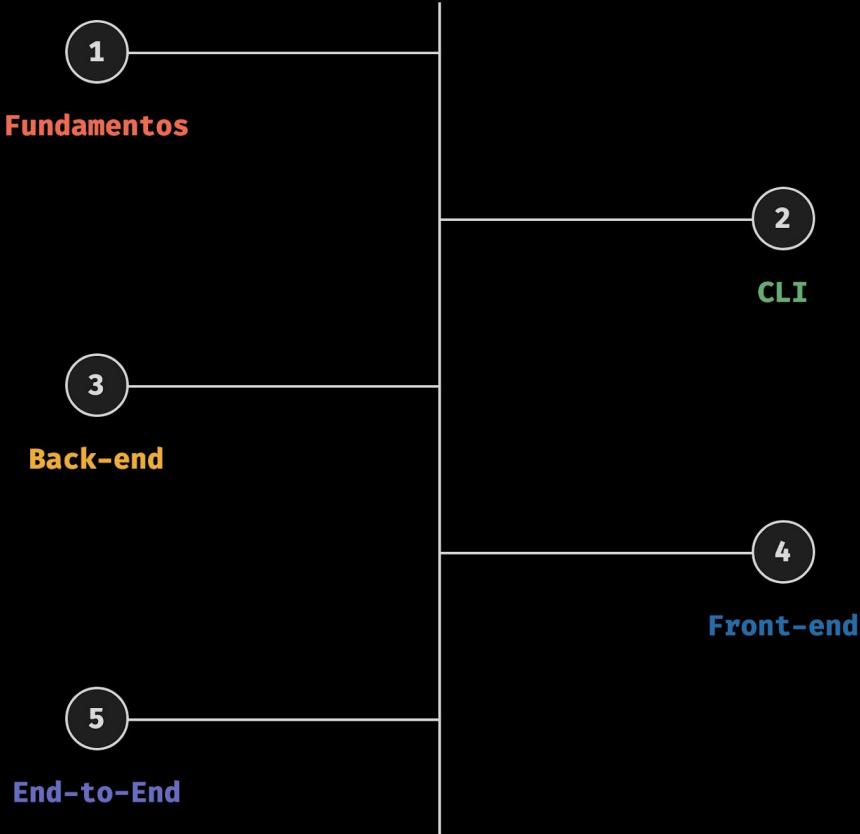
Algumas **ferramentas**

- [Eslint](#): linter de código JavaScript, utilizado nos exemplos de análise estática;
- [Jest](#): framework de testes, usados nos exemplos de testes unitários e de integração;
- [Artillery](#): ferramenta utilizada no exemplo de testes de carga;
- [Loki](#): ferramenta utilizada no exemplo de testes de regressão visual;
- [Cypress](#): ferramenta utilizada nos exemplos de testes de ponta-a-ponta.

Existem diversas outras que trabalham de forma similar e também de forma totalmente diferente. O **oceano** de testes é muito vasto ...

Se você tem interesse em se
aprofundar em testes ...

JavaScript Assertivo





@gabrieluizramos

gabrieluizramos.com.br

Dúvidas?

slides: bit.ly/conversa-sobre-testes
javascriptassertivo.com.br