



Desenvolver aplicações Backend para WEB

Prof. Renan Ponick

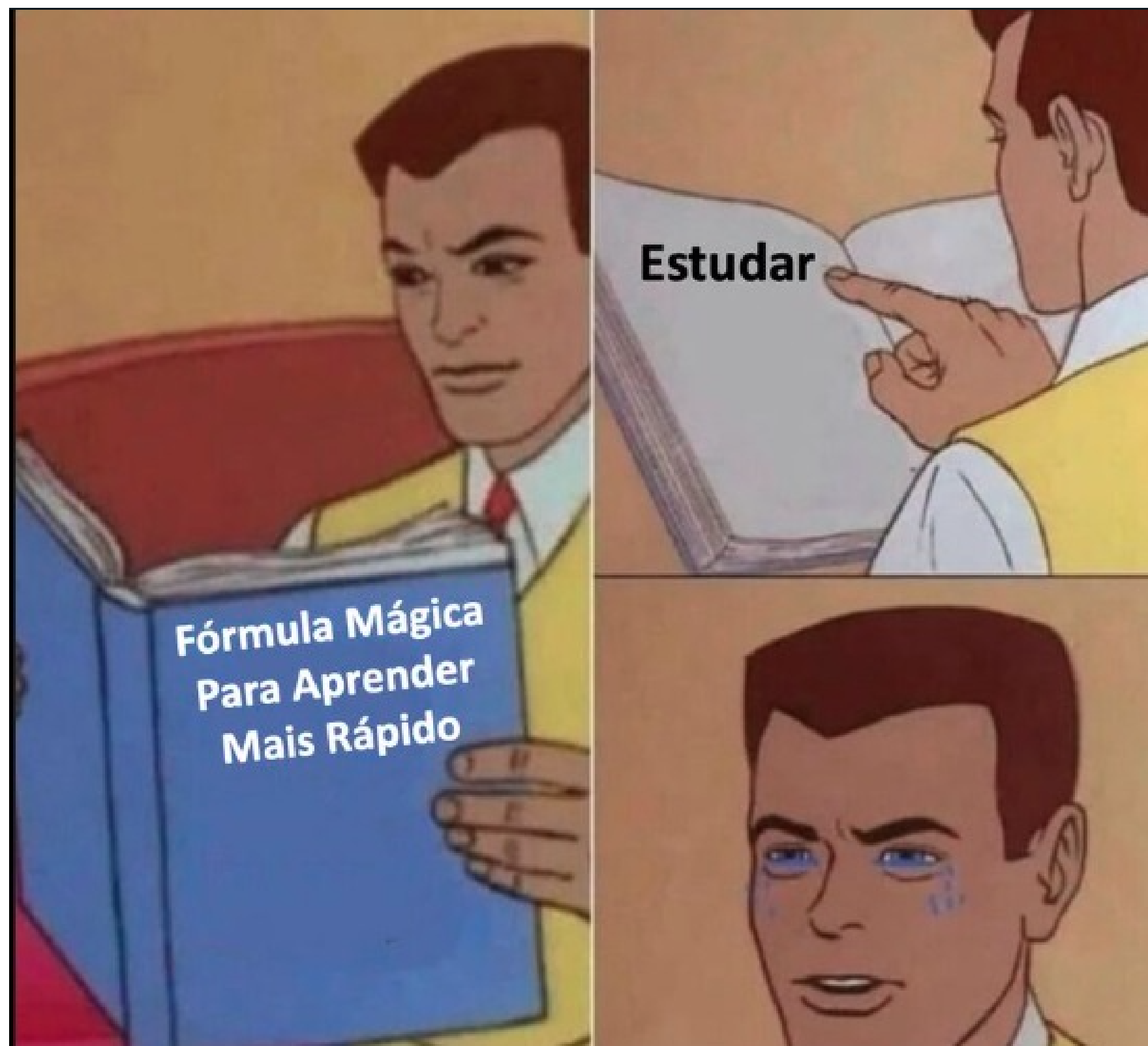


Lema

"Menor erro é uma tarefa mal executada do que a falta de decisão. É mais desejado errar do que não fazer"

Baltasar Gracián

Motivacional



Agenda

17/08/2023 - Postman + Post

18/08/2023 - Review 1

24/08/2023 - Teste Unitário

25/08/2023 - Exercícios + Testes

31/08/2023 - Review 2

01/09/2023 - Exercícios + Testes

07/09/2023 - FERIADO

08/09/2023 - FERIADO

14/09/2023 - Boas Práticas + Arq. REST e RESTFull

15/09/2023 - Exercícios + Testes

21/09/2023 - Avaliação

22/09/2023 - CRUD

28/09/2023 - ORM - MySQL

29/09/2023 - Exercícios + Testes



Seja esperto, evite retrabalho.

Um dos grandes benefícios da automação é prover feedback de forma frequente desde o início das atividades de desenvolvimento.

Ferramentas de automação.

1. Disponibiliza uma estrutura de testes (**Mocha**, Jasmine, **Jest**, **Cucumber**);
2. Prove funções de validação (assert) (**Chai**, Jasmine, **Jest**, Unexpected);
3. Gera, mostra (display) e observa (watch) o resultado dos testes (**Mocha**, Jasmine, **Jest**, Karma);
4. Gera e compara snapshots do componente e estrutura de dados, para garantir que as mudanças anteriores foram como o planejado (**Jest**, Ava);
5. Prove mocks, spies, e stubs (Sinon, Jasmine, enzyme, **Jest**, testdouble)
6. Gera relatórios de cobertura de código (code coverage) **Istanbul**, **Jest**, Blanket);
7. Prove um navegador, ou ambiente parecido, com controle sobre seus cenários de execução (Protractor, Nightwatch, Phantom, Casper).

(Cypress, Groovy, Katalon)

JEST



A biblioteca Jest é uma estrutura de teste em JavaScript amplamente utilizada, projetada principalmente para testar código JavaScript e aplicações construídas com tecnologias como Node.js, React, Vue.js e muito mais. Ela fornece um conjunto abrangente de recursos e ferramentas para facilitar a criação, execução e análise de testes automatizados.

JEST

Instale o JEST em desenvolvimento

```
19  },
20  "homepage": "https://github.com/renanponick/javascript#readme",
21  "dependencies": {
22    "express": "^4.18.2"
23  },
24  "devDependencies": {
25    "jest": "^29.6.2",
26    "nodemon": "^3.0.1"
27  }
28 }
29
```

PROBLEMS GL OUTPUT DEBUG CONSOLE GL TERMINAL

```
● renan@ng0502:~/Documents/senac/JP/javascript$ npm install jest --save-dev

added 271 packages, and audited 363 packages in 18s

41 packages are looking for funding
  run `npm fund` for details

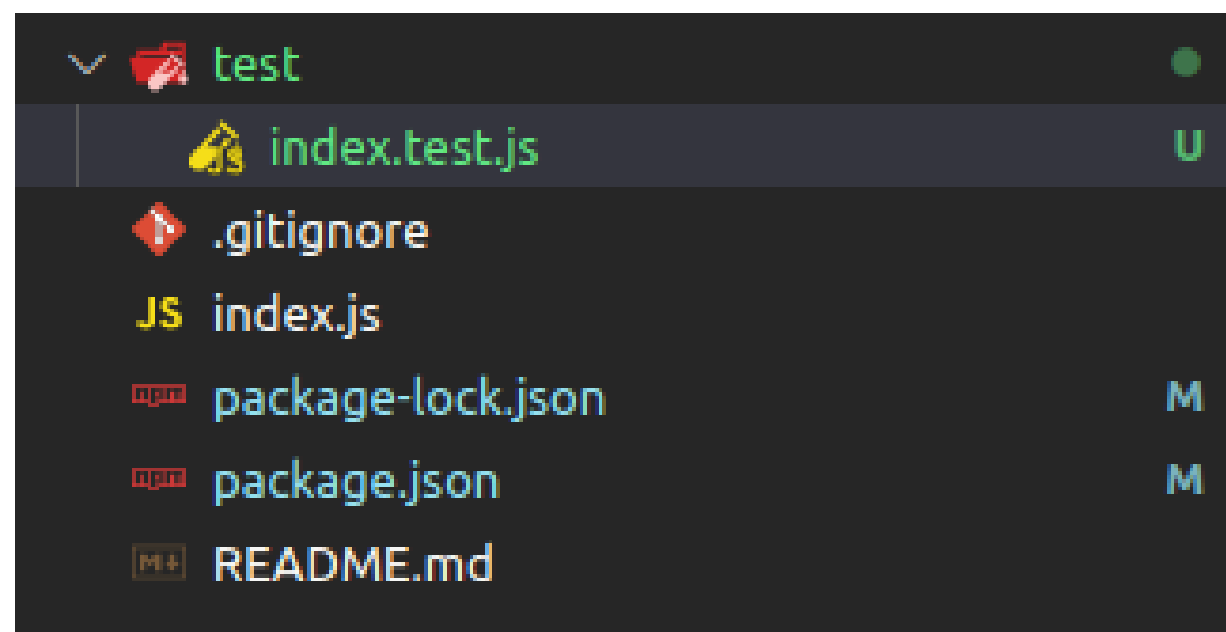
found 0 vulnerabilities
○ renan@ng0502:~/Documents/senac/JP/javascript$
```


JEST

Altere o script para executar o teste:

```
7  "scripts": {  
8    "start": "nodemon index.js",  
9    "dev": "jest test/index.test.js",  
10   "test": "yarn node --experimental-vm-modules $(yarn bin jest)"  
11 },
```

Crie uma pasta **teste** e um arquivo **index.test.js** dentro do projeto



JEST

Dentro do arquivo escreva o seguinte:

```
1 import { describe, expect, it } from '@jest/globals';
2 import { exercicio1 } from "../exercicios/exercicios-parte-um";
3
4 describe('Testes do primeiro exercício', () => {
5   // Executado antes de TODOS os testes
6   beforeAll(async () => {
7     console.info('Iniciando TDD com jest!');
8   });
9
10  // Executado após TODOS os testes
11  afterAll(() => {
12    console.info('Encerrados os testes');
13  });
14
15  it('Should sum two numbers', () => {
16    const result = exercicio1(1, 2)
17
18    expect(result).toBe(3);
19  })
20
21 })
```

Describe = Grupo de testes
it = Cada testes em específico

BeforeAll = Roda antes dos it s
AfterAll = Roda depois dos it s

Expect = valor recebido
toBe = deve ser



Instale o yarn

```
npm install -g yarn
```

JEST

Execute `npm test`

```
renan@ng0502:~/Documents/senac/JP/javascript$ npm test

> javascript@1.0.0 test
> yarn node --experimental-vm-modules $(yarn bin jest)

yarn node v1.22.19
console.info
  Iniciando TDD com jest!

    at Object.info (test/index.test.js:6:15)

console.info
  Encerrados os testes

    at Object.info (test/index.test.js:11:15)

(node:17228) ExperimentalWarning: VM Modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
FAIL test/index.test.js
  Testes do primeiro exercício
    ✕ Should sum two numbers (5 ms)

● Testes do primeiro exercício › Should sum two numbers

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

Expected: 4
Received: 3

   15 |         const result = exercicio1(1, 2)
   16 |
>  17 |         expect(result).toBe(4);
      |                           ^
   18 |       })
   19 |
   20 |     })
```



Cobertura de testes

Cobertura de testes

A cobertura de testes, também conhecida como "cobertura de código", é uma métrica que indica a porcentagem de código fonte do seu software que é exercida por testes automatizados.

1. Identificação de Áreas Não Testadas;
2. Aumento da Confiabilidade;
3. Facilita a Manutenção;
4. Melhoria na Qualidade do Código;
5. Feedback Rápido;
6. Documentação Viva;
7. Apoio a Processos de Integração Contínua (CI) e Entrega Contínua (CD);

Coverage

Adicione o script "test:cov" para executar o a cobertura de teste:

```
7  "scripts": {  
8    "start": "nodemon index.js",  
9    "dev": "jest test/index.test.js",  
10   "test": "yarn node --experimental-vm-modules $(yarn bin jest)",  
11   "test:cov": "yarn node --experimental-vm-modules $(yarn bin jest) --coverage"  
12 },
```

Coverage

Execute npm run test:cov

```
● renan@ng0502:~/Documents/senac/JP/javascript$ npm run test:cov

> javascript@1.0.0 test:cov
> yarn node --experimental-vm-modules $(yarn bin jest) --coverage

yarn node v1.22.19
console.info
  Iniciando TDD com jest!

    at Object.info (test/index.test.js:6:15)

console.info
  Encerrados os testes

    at Object.info (test/index.test.js:11:15)


(node:17921) ExperimentalWarning: VM Modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
PASS test/index.test.js
  Testes do primeiro exercício
    ✓ Should sum two numbers (3 ms)

-----
File                                     % Stmts   % Branch   % Funcs   % Lines   Uncovered Line #s
-----
All files                               100        100        100        100
exercicios-parte-um.js                  100        100        100        100
-----

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.355 s, estimated 1 s
Ran all test suites.
```




10 min para praticar



Bugs e Erros

Como identificá-los no meio de códigos gigantes?

Solução 1

```
escreva("entrou aqui");
```

```
echo "funciona";
```

```
printr("batata");
```

```
console.log("palavrão horrendo devido ao stress do  
bug não encontrado");
```

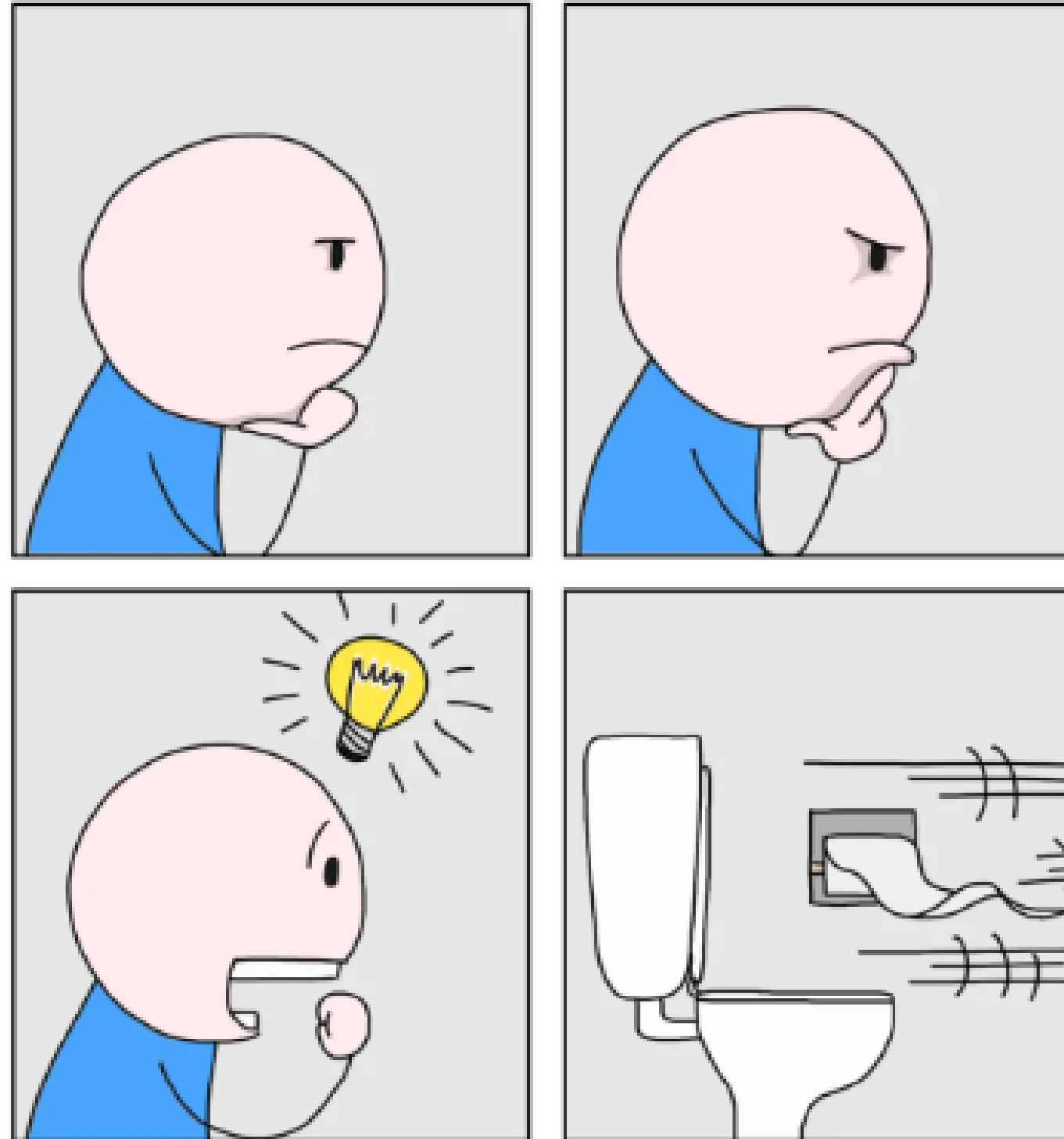


Solução 2

Debug ou Debugging
Depurar ou Depuração

Solução 2

DEBUGGING



Solução 2

The image shows a VS Code editor window with a JavaScript file named `app.js` and a `launch.json` file. The code in `app.js` is as follows:

```
2 const app = express();
3 const http = require('http').Server(app);
4 const io = require('socket.io')(http);
5 const path = require('path');
6
7 //Serve public directory
8 app.use(express.static(path.join(__dirname, 'public')));
9
10 app.get('/', function(req, res) {
11   res.sendFile(path.join(__dirname, 'public/index.html'));
12 });
13
14 io.on('connection', function(socket) {
15   console.log('a new user connected ');
16
17   socket.on('disconnect', () => {
18     console.log('user disconnected');
19   });
20 });
```

The left sidebar shows the following panels:

- VARIABLES**: Shows local variables like `socket` and `this`.
- WATCH**: Shows variables like `usernameInput`, `messages`, and `socket`.
- CALL STACK**: Shows the call stack with the current function being `(anonymous function) app.js 15:2`.
- BREAKPOINTS**: Shows a breakpoint set at line 15 of `app.js`.

The bottom panel shows the **DEBUG CONSOLE** with the following output:

```
Debugging with legacy protocol because Node.js v7.10.0 was detected.
/usr/local/bin/node --nolazy --debug-brk=60065 app.js
(node:20173) DeprecationWarning: node --debug is deprecated. Please use node --inspect instead.
Debugger listening on 127.0.0.1:60065
listening on port 3000
```

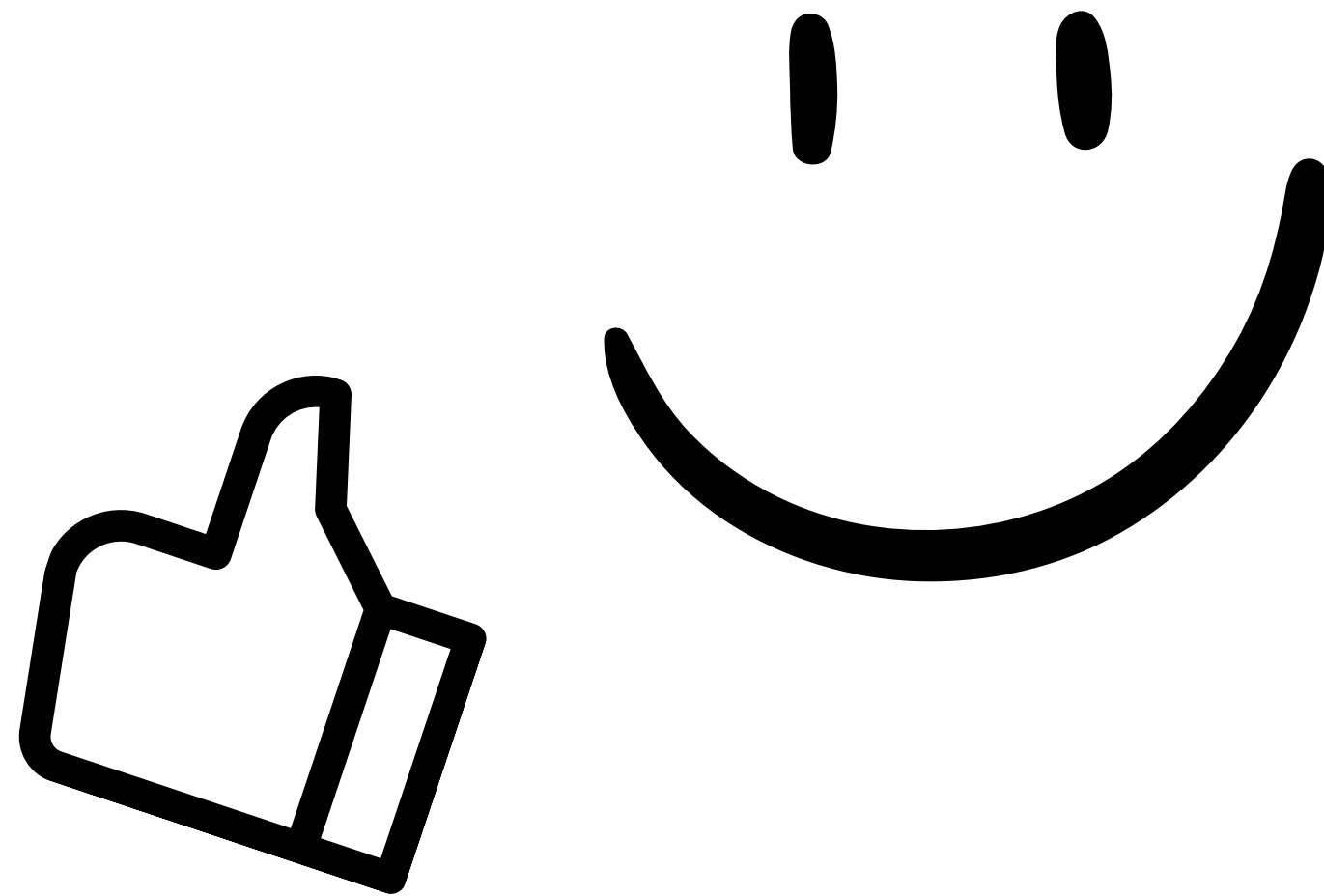


Definição

É o processo voltado para identificar e remover erros existentes no código.

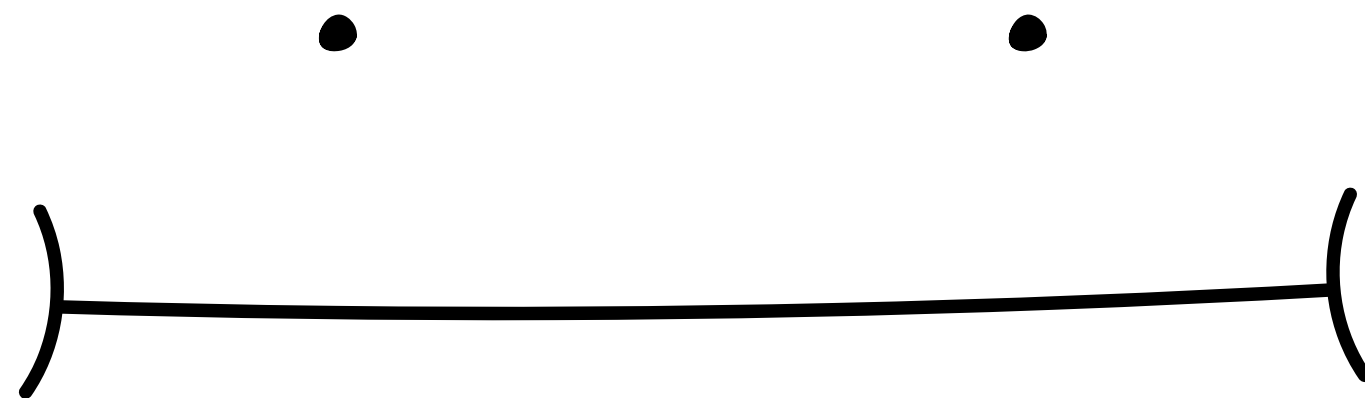
Dica

Ele não descobre erros sozinho!



Dica

Ele não analisa erros sozinho!



Dica

Ele não corrige erros sozinho!





e agora?

Você vai precisar reconhecer o bug, encontrar exatamente em qual parte do código ele esconde, para só então ativar o modo debug e DETALHAR o erro.

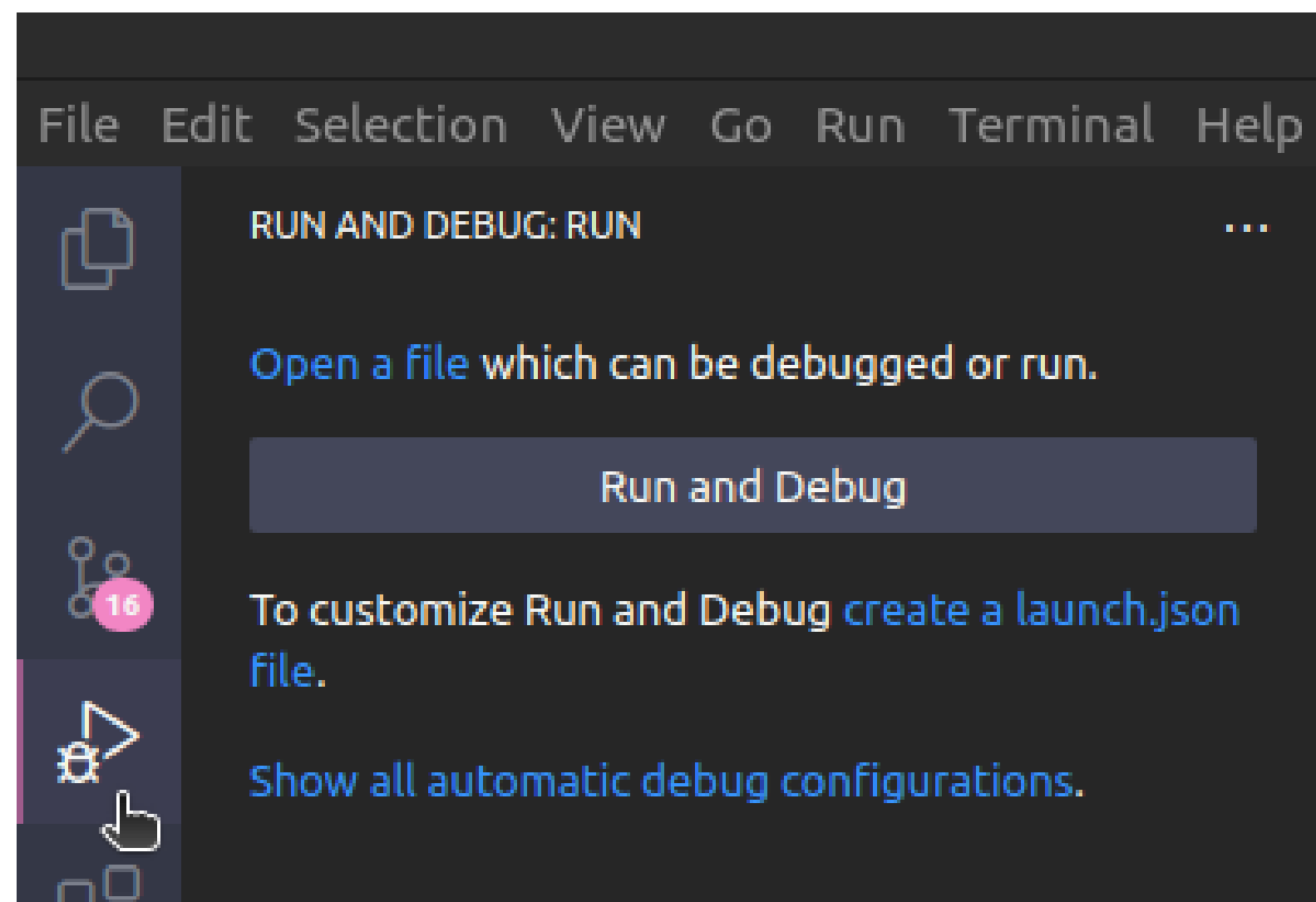


Importancia

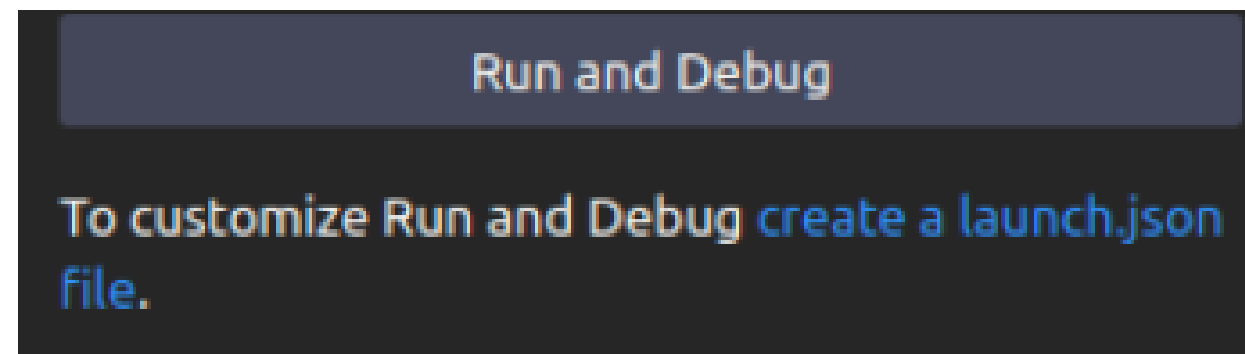
O debug melhora a gestão de tempo e produtividade. É nítido que corrigir bugs manualmente pode ser uma tarefa demorada e frustrante.

Na prática

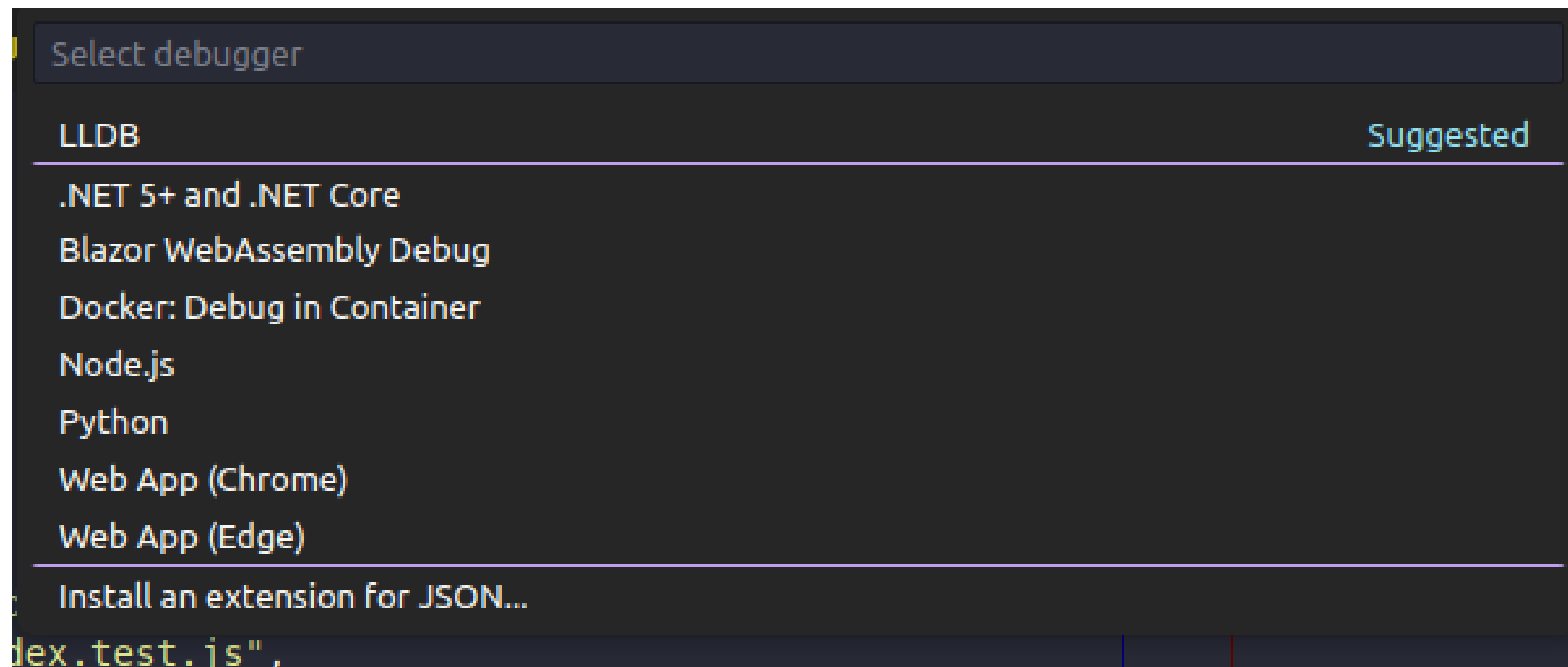
Dentro do VsCode encontre o ícone do debug



Na prática



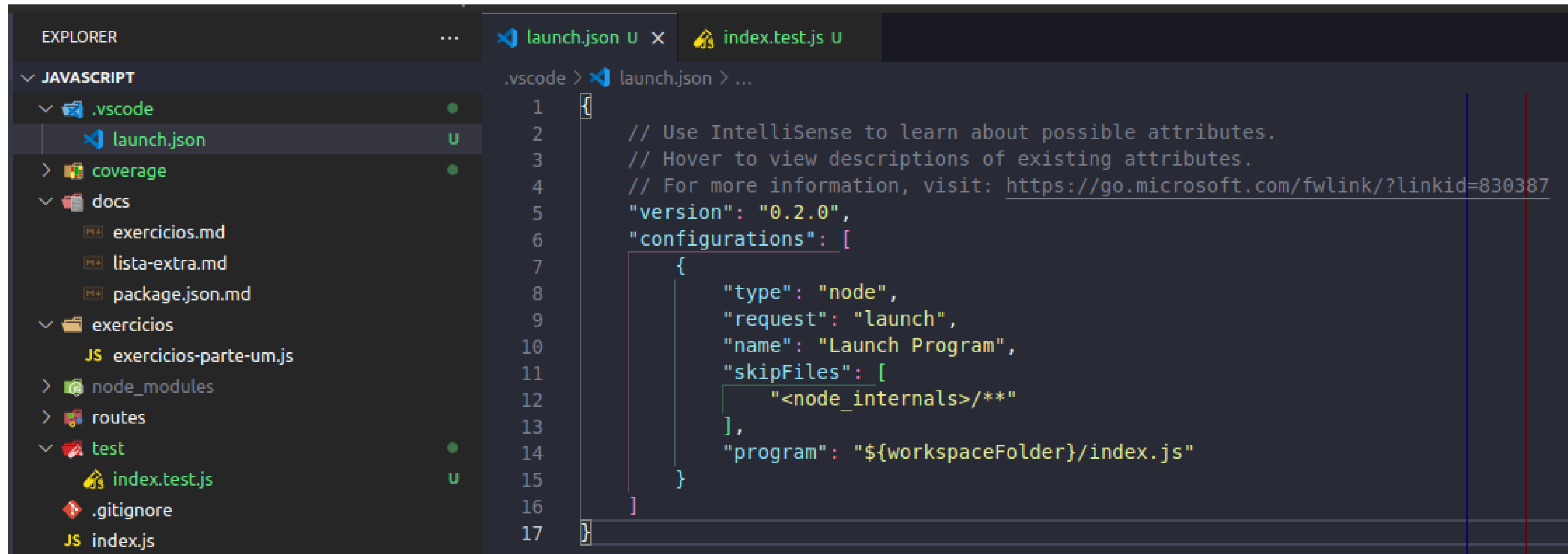
Abaixo do botão Run and Debug, clique na msg em azul



Selecione Node.js

Na prática

Será gerado um arquivo de configuração para debugar aplicações node:

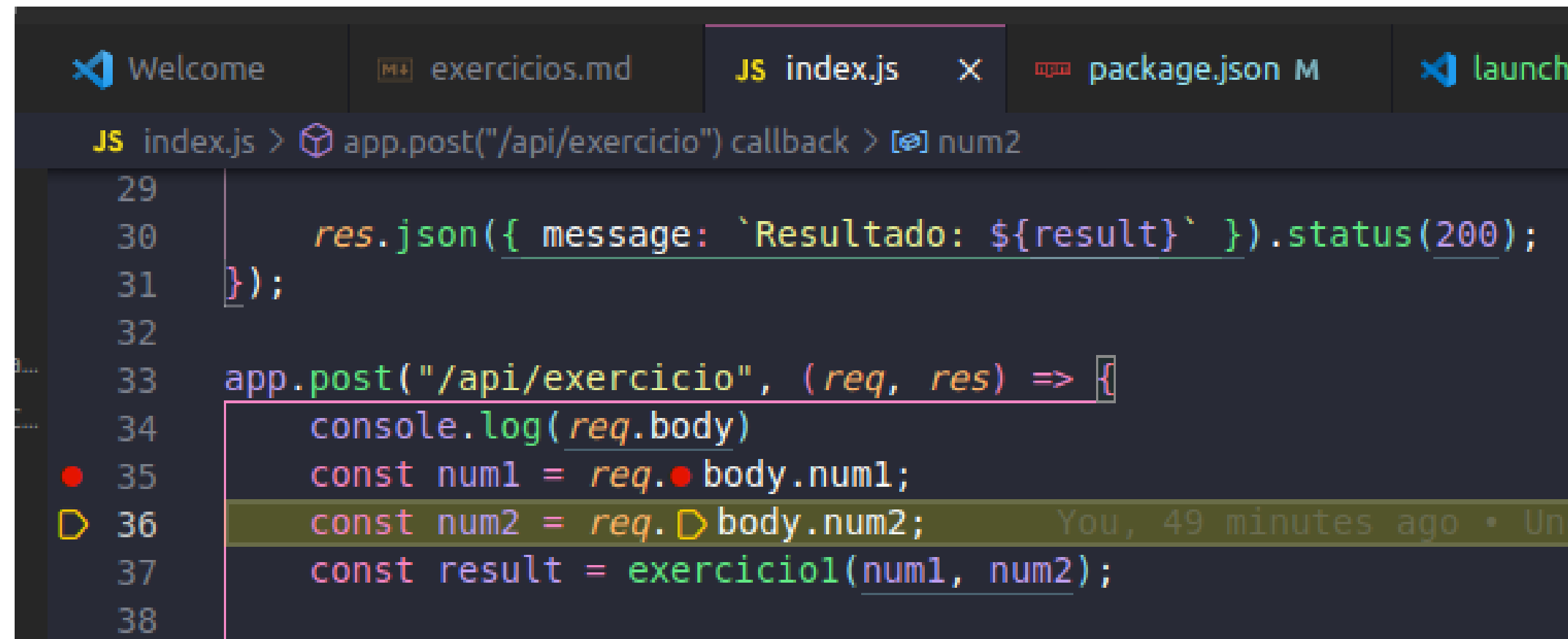


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the file structure of a project, including folders like .vscode, coverage, docs, and test, and files like launch.json, index.test.js, and index.js. The main editor area shows the launch.json file, which is a JSON configuration for debugging Node.js applications. The file contains comments and a configuration for a 'Launch Program'.

```
1 {  
2   // Use IntelliSense to learn about possible attributes.  
3   // Hover to view descriptions of existing attributes.  
4   // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387  
5   "version": "0.2.0",  
6   "configurations": [  
7     {  
8       "type": "node",  
9       "request": "launch",  
10      "name": "Launch Program",  
11      "skipFiles": [  
12        "<node_internals>/**"  
13      ],  
14      "program": "${workspaceFolder}/index.js"  
15    }  
16  ]  
17 }
```

Na prática

Adicione breakpoints no código, faça uma requisição e veja a mágica acontecer.



The screenshot shows the Visual Studio Code editor with a file named `index.js` open. The code is as follows:

```
29  
30     res.json({ message: `Resultado: ${result}` }).status(200);  
31 });  
32  
33 app.post("/api/exercicio", (req, res) => {  
34     console.log(req.body)  
35     const num1 = req.body.num1;  
36     const num2 = req.body.num2;  
37     const result = exercicio1(num1, num2);  
38
```

Breakpoints are indicated by red dots on line 35 and a yellow square on line 36. The breadcrumb at the top of the editor shows the file path: `JS index.js > app.post("/api/exercicio") callback > num2`.

Esses são os controladores do debug





Exercício

Continue realizando as listas de exercícios,
porém agora faça testes para cada uma
desses exercícios.