

AUTOMAÇÃO DE TESTES EM API REST

Objetivo: Capacitar os Alunos a se tornarem proficientes na automação de testes de APIs Rest ao compreenderem e aplicarem os princípios e práticas fundamentais para automatizar testes em APIs Rest.

Prof. MSc. Taynara Luana



Ementa

| API | API - ESTRUTURA | AUTOMAÇÃO DE TESTES DE API | ENCERRAMENTO |
|--|--|--|--|
| <ul style="list-style-type: none">• Introdução a APIs• Estilo arquitetural REST | <ul style="list-style-type: none">• Métodos• Endpoints• JSON• Requisições• Respostas• Status Code | <ul style="list-style-type: none">• Fundamentos e Conceitos sobre automação de testes para APIs REST• Bibliotecas, primeiro código de testes de API• Logs e métodos• Headers e autenticações• Asserções e validações da estrutura da resposta em JSON• Boas práticas no desenvolvimento de testes automatizados | <ul style="list-style-type: none">• Quais forças externas podem impactar negativamente seu negócio?• Há empresas com potencial de se tornar concorrentes? |

STATUS CODE

Status retornado de um servidor ao cliente onde é realizada uma requisição HTTP, que irá indicar por sua vez o resultado dessa requisição:

 mdn web docs

Códigos de status de respostas HTTP - HTTP | MDN

Os códigos de status de resposta HTTP indicam se uma solicitação HTTP específica foi concluída com êxito. As respostas são agrupadas em cinco classes

 MDN Web Docs

Curiosidade

Um dos erros comuns que podemos identificar em uma API, é a utilização de status 200 para todas as respostas que temos, mesmo essas respostas sendo de erro.

Boa prática

Utilizar o código de status mais adequado ao processo, para retornar a sua devida resposta.

As mensagens que você fornecer ou estiver testando em uma API, devem ser claras e detalhadas, e por sua vez devem ser apresentadas no corpo da resposta.



STATUS CODE

Falando em boas práticas, então seria uma boa prática os retornos que a nossa API está retornando ao cadastrar um registro de usuários por exemplo?

200

Atualmente a nossa API de testes está nos retornando o código 200 e um corpo de resposta, mas essa resposta indica que o resultado depende do método HTTP.

201

Como a própria documentação nos diz:

Created:

A requisição foi bem sucedida e um novo recurso foi criado como resultado. Esta é normalmente a resposta enviada após as solicitações **POST** ou algumas solicitações **PUT**.



Hora da prática

Projeto

Agora que você já tem uma primeira request desenvolvida em seu projeto de testes automatizados, para cadastro de novos usuários, execute novamente o seu projeto com a nova versão do projeto (projeto backend).

- Para quem não estiver usando git, entrar no repositório e fazer download do projeto atualizado na versão da main.
- Para quem estiver usando o git, basta executar o git pull, no projeto principal.

Depois que a aplicação estiver de pé, você irá de fato rodar o teste realizado na aula anterior.

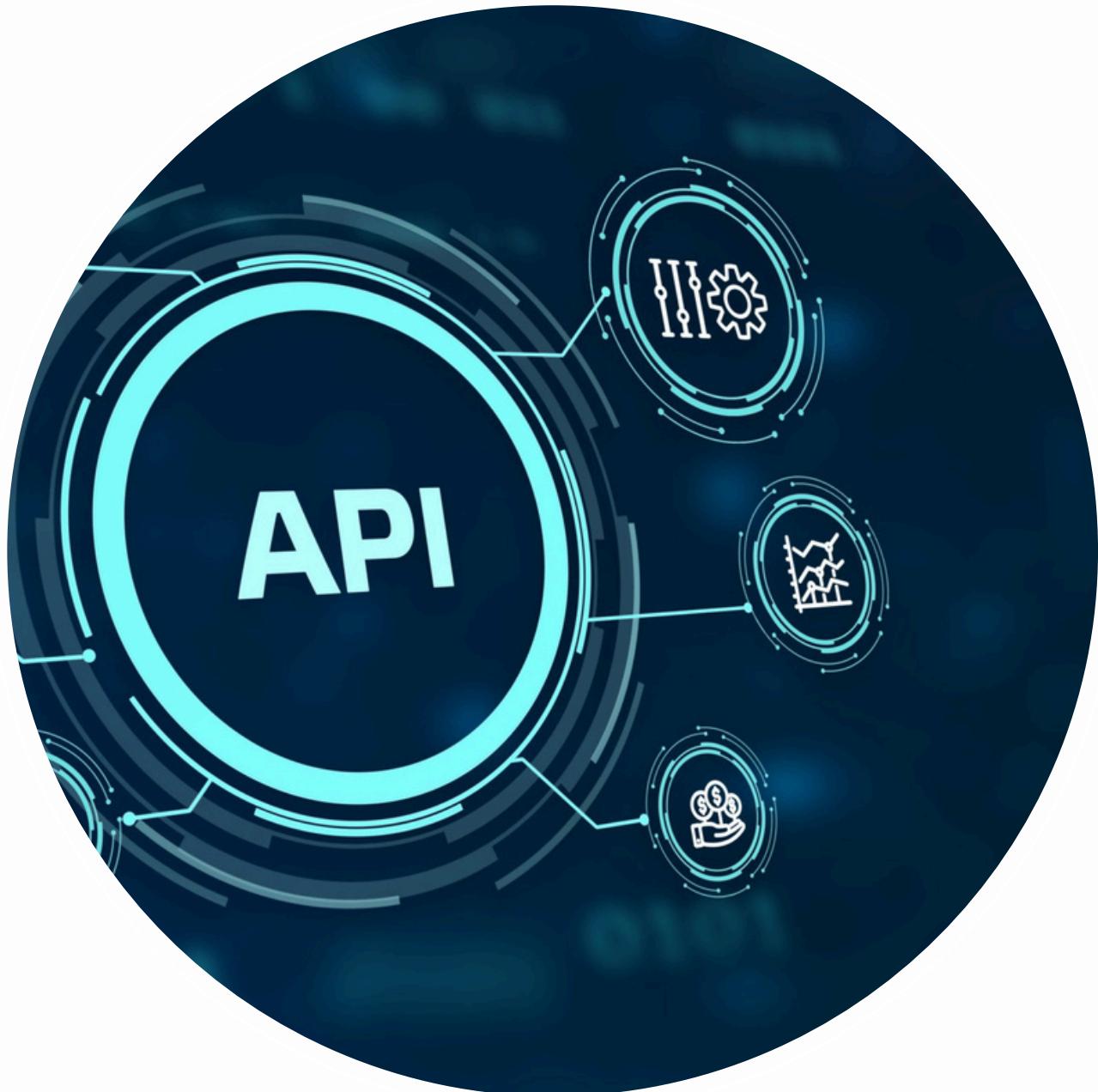
Verifique o resultado do teste.



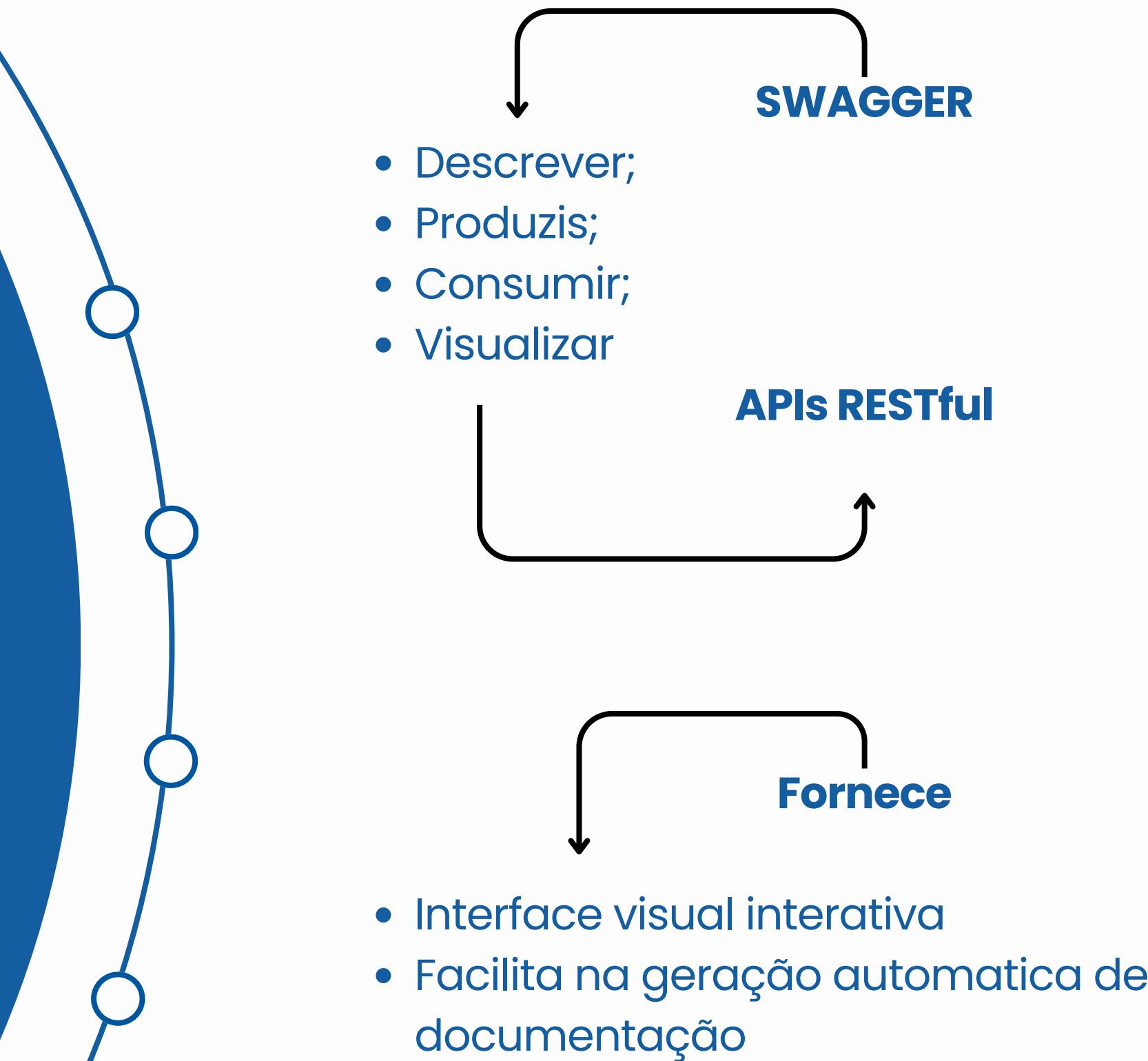
Atividade 1

Com a aplicação rodando, você pode identificar que os testes de inserção de um novo registro, quebraram, pois o status code agora passa a ser 200. Com isso, altere os testes que você já desenvolveu de inserção de registros para que ele agora valide para nós se o status code retornado foi 201.

Rode novamente o projeto, e confira o resultado.



Documentação API



Documentação API

BENEFÍCIOS

- Clareza;
- Interatividade;
- Facilitação de Testes Automatizados;

APIs RESTful

Projeto SWAGGER

 Swagger
Supported by SMARTBEAR

API de CRUD para testes automatizados 1.0.0 OAS 3.0

Documentação da API de CRUD para testes Automatizados

Usuarios API para gerenciamento de usuários

^

GET /users/{id} Retorna um usuário pelo ID

PUT /users/{id} Atualiza um usuário pelo ID

DELETE /users/{id} Exclui um usuário pelo ID

GET /users Retorna a lista de todos os usuários

POST /users Cria um novo usuário

<http://localhost:3000/api-docs/>

Atividade 2

Vamos realizar a instalação do restante das dependências para que possamos ter acesso a documentação da swagger.

- Com a nova versão já baixada do projeto crud, você deverá executar o seguinte comando:
 - npm install
- Este comando fará a instalação das dependências para o projeto corretamente.
- Após a instalação das dependências, navegue até a pasta do backend, e execute o seguinte comando:
 - node app.js
 - Pronto! Sua aplicação estará sendo executada na porta 3000.

<http://localhost:3000/api-docs/>



Atividade 3

- Criação de um novo usuário com dados válidos: Deve retornar um objeto do usuário criado e um status 201.
- Criação de um usuário com dados ausentes: Deverá lançar um erro e retornar um status 422 apresentando uma mensagem descritiva e detalhada do erro ocorrido.



Atividade 4

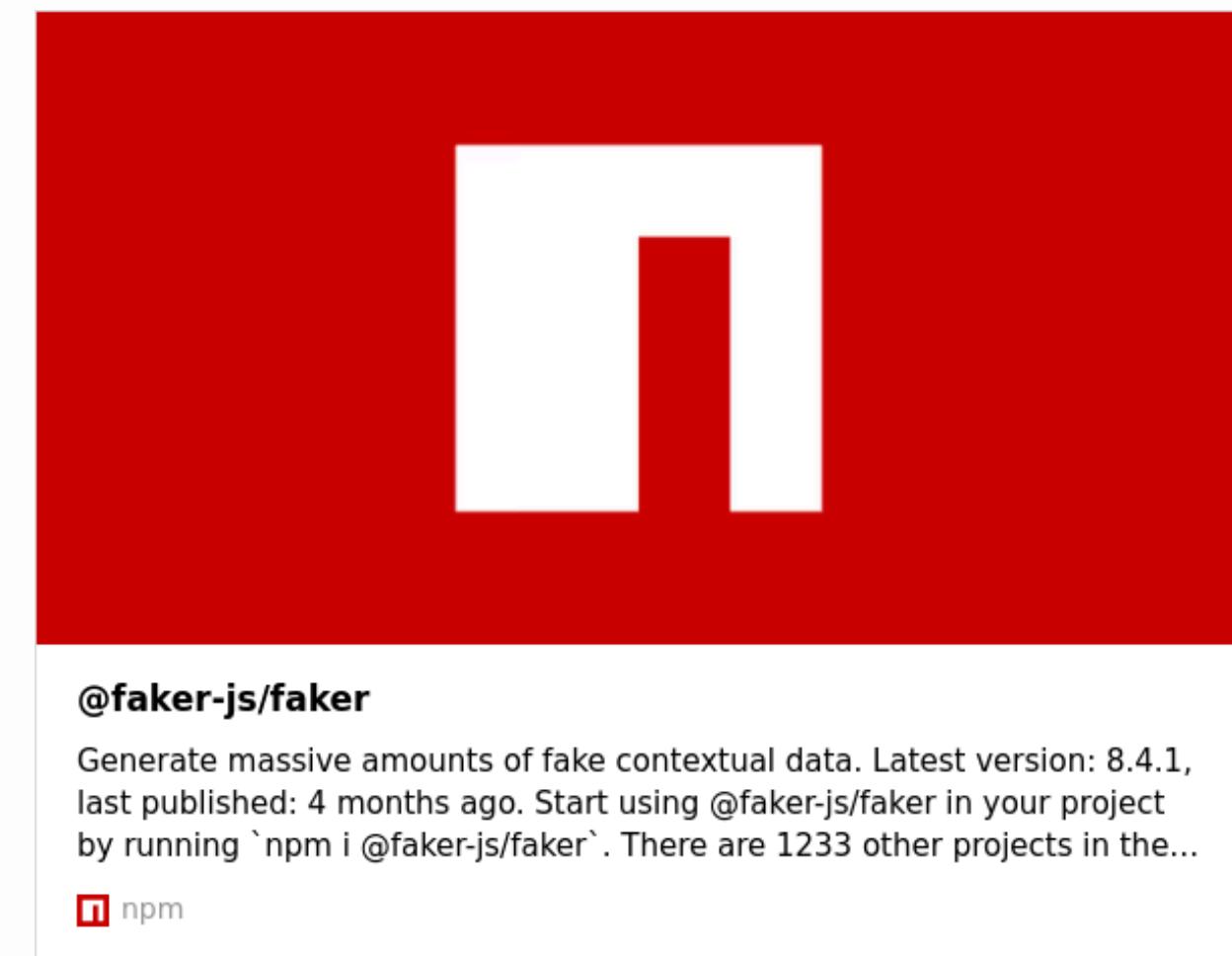
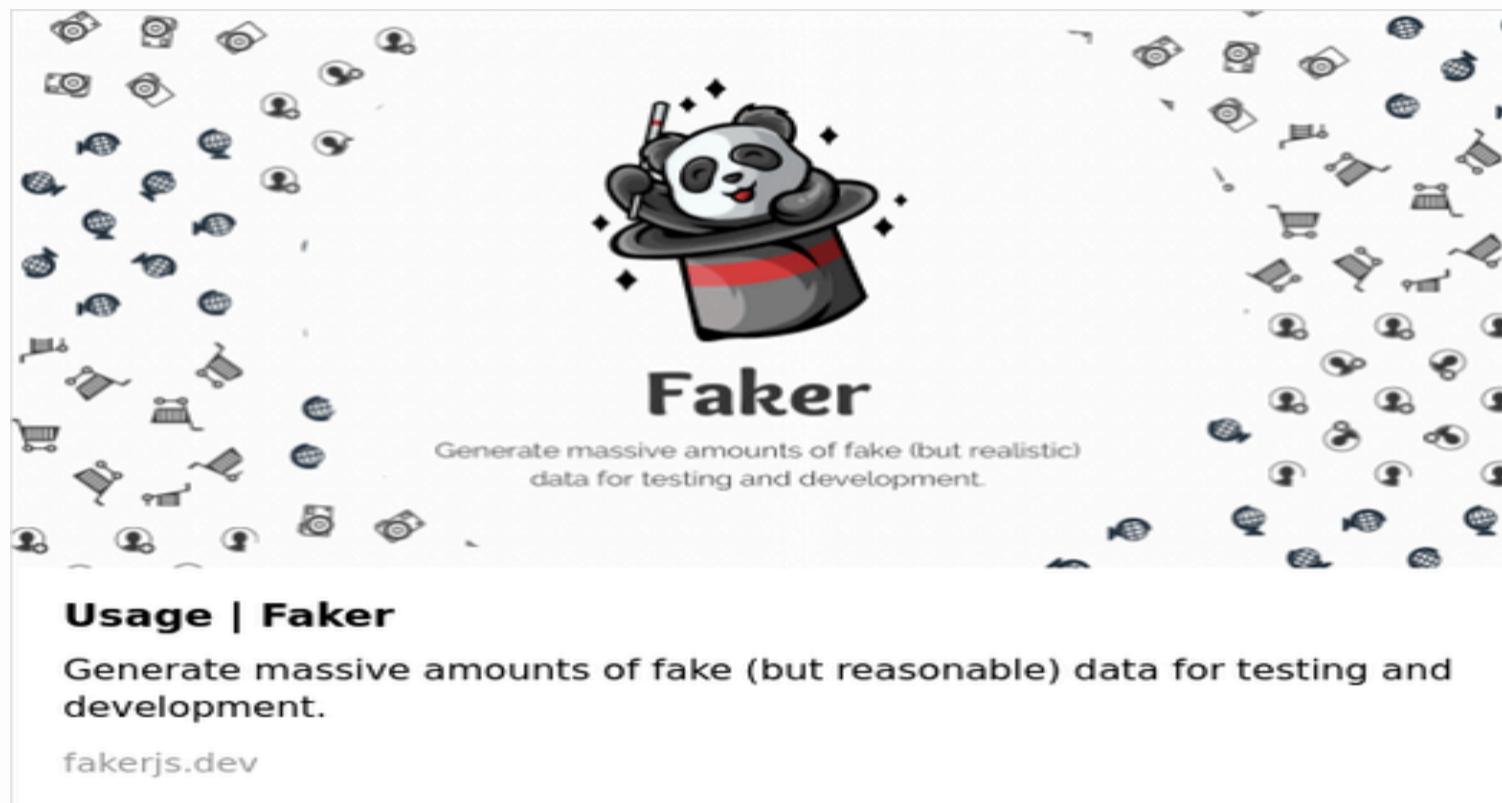
- Criação de um novo usuário com dados válidos: Deve retornar um objeto do usuário criado e um status 201, e consultando o registro gerado anteriormente para verificar que o mesmo foi devidamente cadastrado.



Utilizando bibliotecas de dados fakers

Em alguns casos, precisamos manipular os dados de forma automática para que possamos ter valores randomicos, e facilitem os nossos testes. Com isso utilizamos algumas bibliotecas que fornecem essa possibilidade. Uma delas é faker-js

`npm install @faker-js/faker --save-dev`



Atividade 5

- Agora o seu teste deverá validar o cadastro de um registro e validando se os dados que enviamos é o mesmo dado que estamos recebendo.



Atividade 6

Inserir um teste automatizado, onde valida a ausencia de um dos campos, e valida se a mensagem retornada é de fato a mensagem que estamos esperando.



Atividade 7

Agora iremos inserir um teste automatizado onde queremos alterar o registro que fizemos a consulta anteriormente.

- Deverá ser alterado somente o nome.
- Execute o teste e valide se o mesmo obteve o resultado esperado.



Atividade 8

Após inserir um registro, consultar e alterar, agora quero que vocês desenvolva um teste onde deverá ser removido o registro cadastrado anteriormente, e um teste para garantir que realmente esse usuário não existe mais.

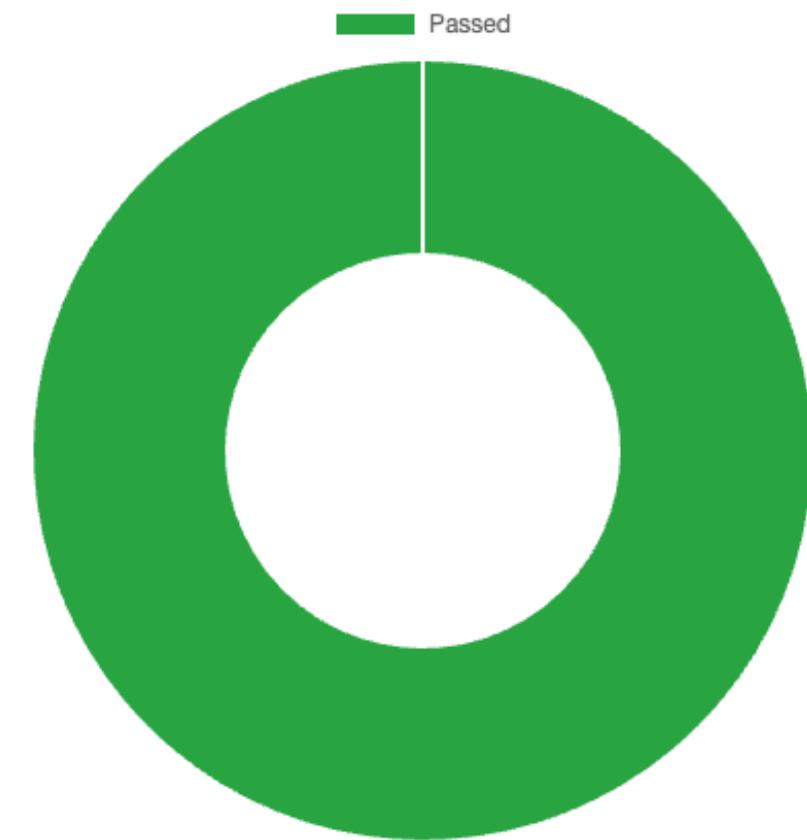


Atividade 9

DESAFIO: Desenvolva um caso de testes onde você deverá inserir uma alteração de registros, e deverá garantir que os registros que você solicitou alteração de fato foram alterados, como resposta.

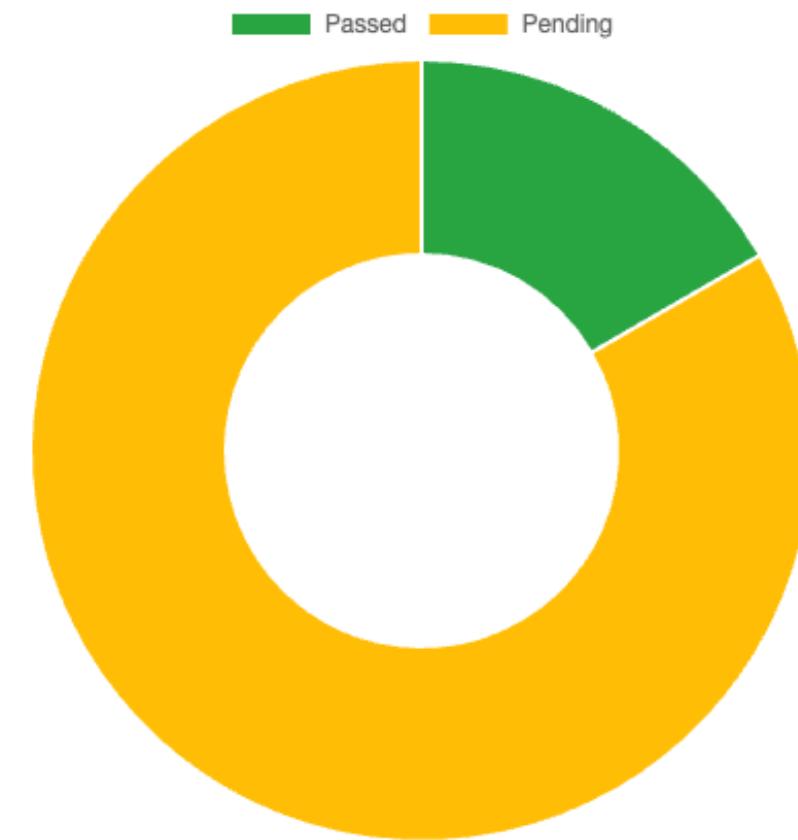


Relatório de Testes



Test Suites

1 passed, 1 total



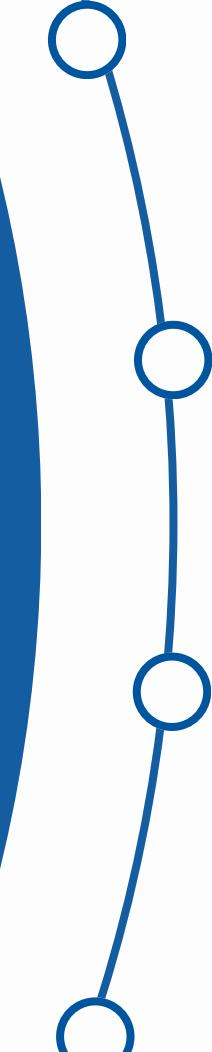
Tests

2 passed, 12 total

/tests/user-test.test.js

2 0 10 0

Relatório de Testes



BENEFÍCIOS

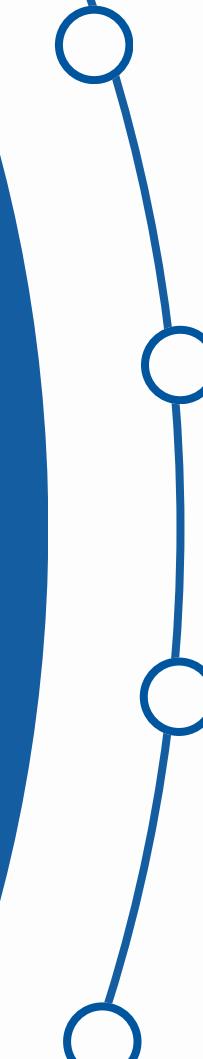
- Visibilidade;
- Interatividade;
- Colaboração;
- Documentação;
- Diagnóstico.

INSTALAÇÃO

- JEST-STARE;
- Na pasta raiz do seu projeto de teste automatizado, você deverá executar o seguinte comando:

npm install jest-stare --save-dev

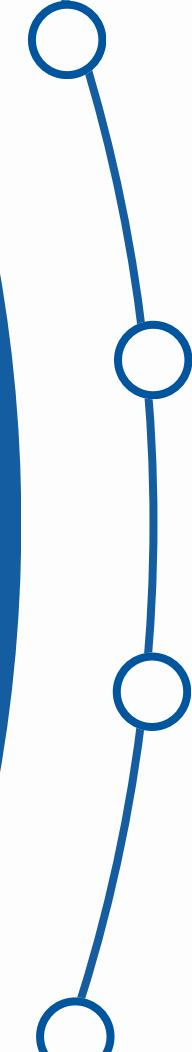
Relatório de Testes



CONFIGURAÇÃO

- Deverá ser configurado o arquivo do jest.config para que possamos dizer ao projeto que queremos gerar um relatório após a execução dos nossos testes.
será apresentado.

Relatório de Testes



EXECUÇÃO 01

- Após executar o projeto jest, então será criado um novo diretório da biblioteca instalada, antes de abrir o arquivo index.html, instale uma extensão para visualizarmos os registros em um servidor local.

LIVE SERVER

EXECUÇÃO P2

- Abra o arquivo no vsCode, (index.html), e clique em Live na nova extensão. Seu relatório será gerado e apresentado corretamente.

Atividade 10

Execute o seu projeto completo, e veja o relatório gerado com as informações que refletem o resultado.



Atualização do Projeto

Agora, precisamos atualizar nosso projeto para pegar as últimas atualizações de código, com isso, para quem estiver utilizando o git, apenas dar o comando **git pull na versão main.**

Para quem está utilizando o projeto baixado de outra maneira, entre no repositório e faça novamente o download.

Após baixar, executar na raiz npm install
Execute novamente o seu projeto acessando a pasta do backend:
node app.js



Utilizando arquivo .env

Corresponde a um arquivo de configuração usado para armazenar variáveis de ambiente para sua aplicação.

Instale a biblioteca do dotenv no seu projeto de testes automatizados.

npm install dotenv



Atividade 11

Agora iremos executar apenas alguns cenários onde o ID de consulta, remoção, e também de alteração de um registro, não existe.

Deve ser retornado 404 para consulta ou alteração ou remoção de registros que já não existem na API.



Configuração de Projeto, boas práticas

ORGANIZAÇÃO DO PROJETO

- Agora que já passamos por todos os nossos métodos propostos a aula, iremos configurar da maneira correta nosso projeto.

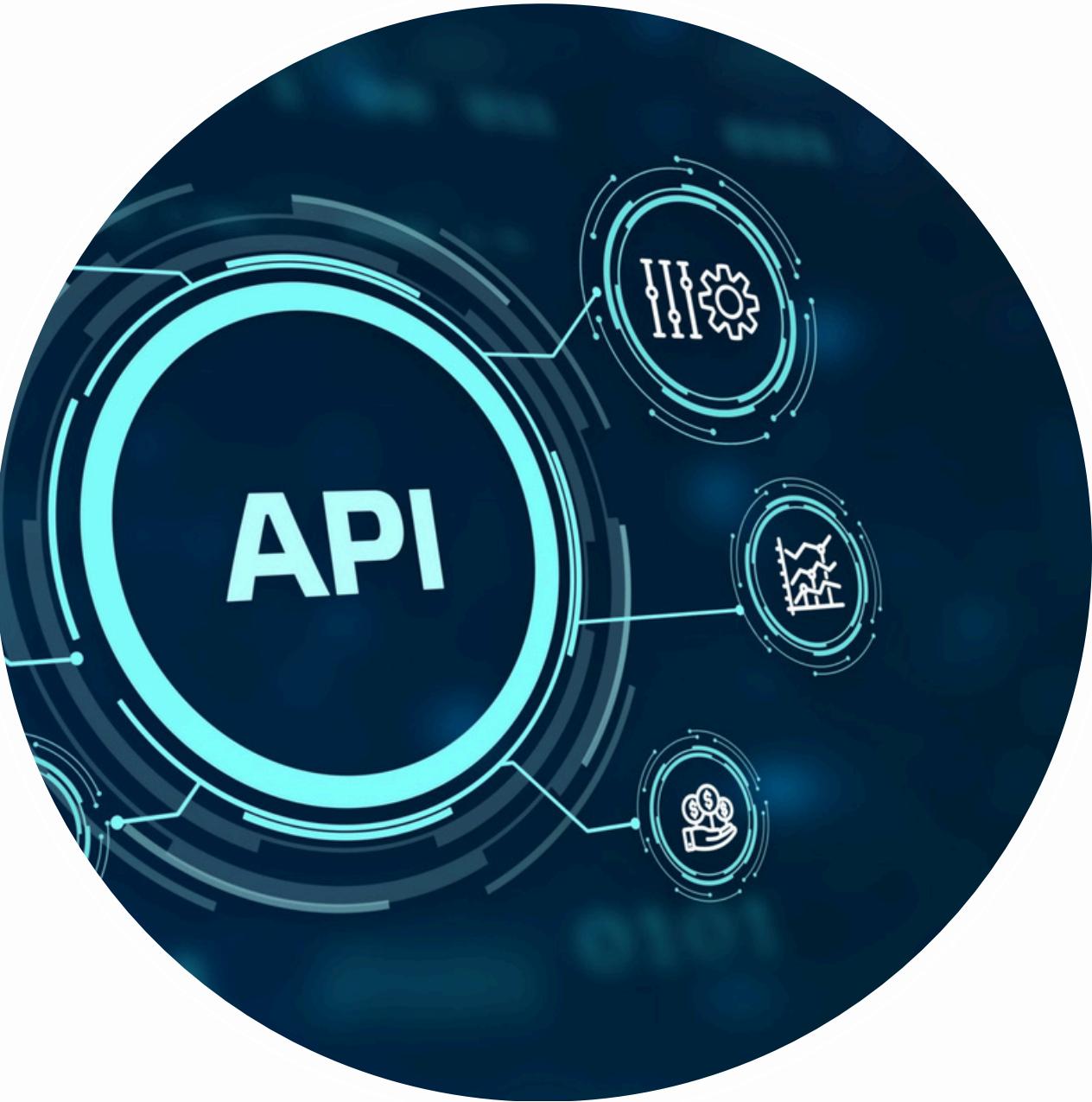
Atividade 12

Após a explicação da estrutura de pastas, agora insira a estrutura de pastas em seu projeto de acordo com o que foi ensinado.

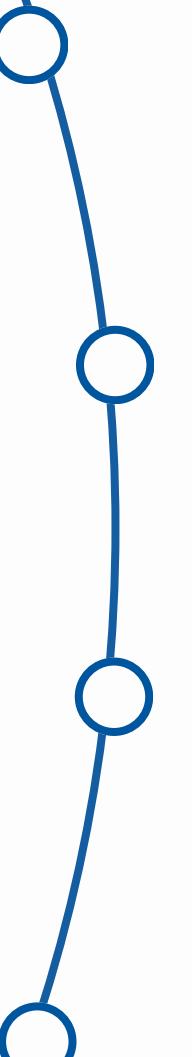


Atividade 13

Separe os testes que você criou por pastas e contextos.



Funções assíncronas e síncronas



Funções síncronas

**Executam uma tarefa de forma linear,
terminando uma ação antes de iniciar a
próxima.**

Execução Linear

A execução é sequencial,
sem interrupções.

Bloqueio de Processo

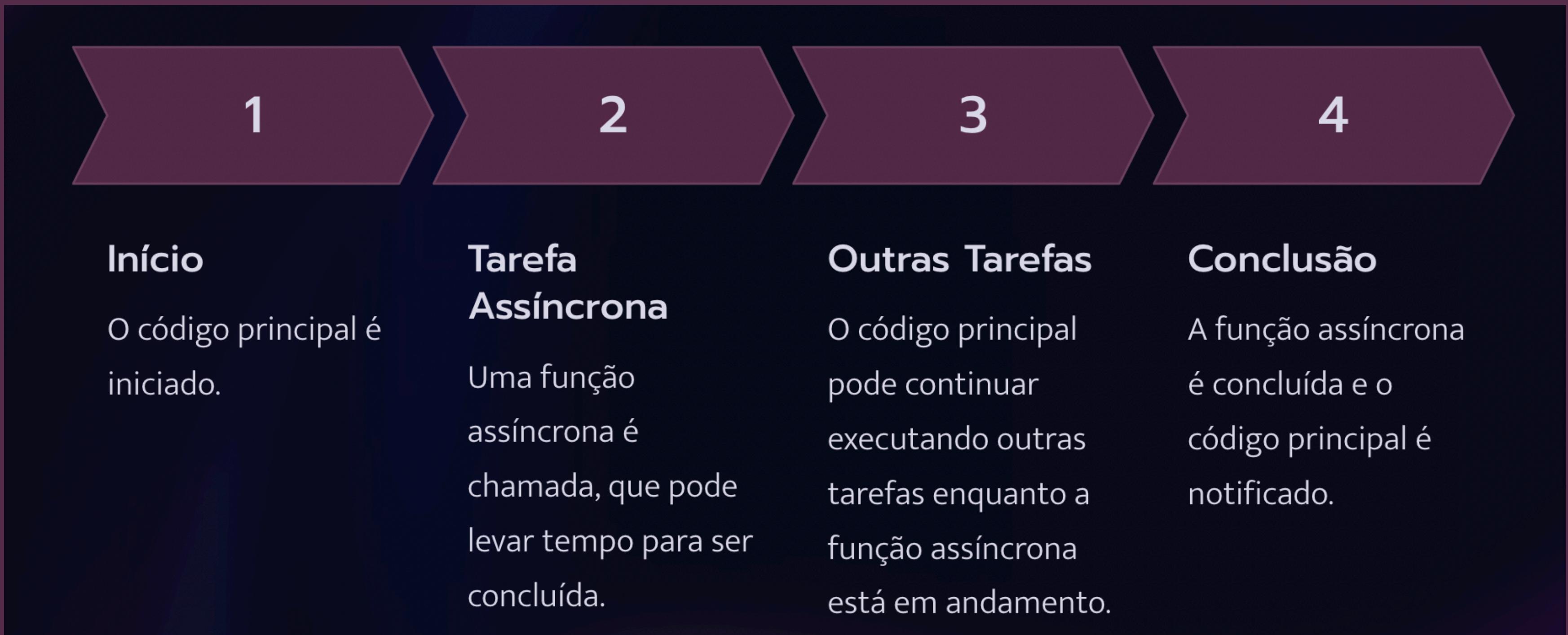
→ A função síncrona bloqueia
o processo até que a tarefa
seja concluída.

Exemplo 1

Termos um código que calcula a soma de dois números. A função síncrona aguarda a finalização da operação de adição antes de prosseguir.

Funções assíncronas

Permitem que o processo continue a executar outras tarefas enquanto uma ação específica está sendo concluída.



EXEMPLO

Uma chamada HTTP GET para uma URL especificada.

Caso a chamada seja bem-sucedida, ela retorna os dados recebidos da API.

Caso a chamada falhar (por exemplo, devido a um problema de rede ou uma resposta inválida da API), a função captura o erro e o lança novamente.

Algumas comparações

| Característica | Síncrona | Assíncrona |
|----------------------|----------|------------|
| Fluxo de execução | Linear | Paralelo |
| Bloqueio de processo | Sim | Não |
| Complexidade | Baixa | Alta |

Promises

Promessas (Promises) recurso fundamental no JavaScript para lidar com operações assíncronas.

]Permitem controlar o fluxo de código e lidar com resultados e erros de forma eficiente.





Significado

Resultado futuro de uma operação assíncrona.

1 Promessa Pendente

A promessa ainda não foi resolvida ou rejeitada.

2 Promessa Resolvida

A promessa foi concluída com sucesso e retorna um valor.

3 Promessa Rejeitada

A promessa falhou e retorna um erro.

```
const minhaPromise = new Promise((resolve, reject) => {
    // Código assíncrono aqui...
    if (sucesso) {
        resolve('Sucesso!');
    } else {
        reject('Erro!');
    }
});

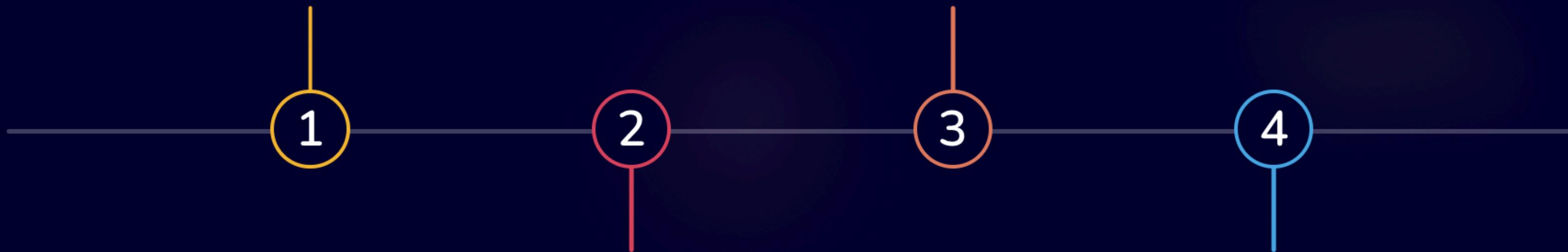
minhaPromise
    .then(resultado => {
        // Código a ser executado se a promessa for resolvida
    })
    .catch(erro => {
        // Código a ser executado se a promessa for rejeitada
});
```

Operação Assíncrona

A operação assíncrona é iniciada
(ex: requisição HTTP).

Resolvendo/Rejeitando

A promessa é resolvida se a operação for bem-sucedida ou rejeitada em caso de erro.



Criação da Promessa

Uma promessa é criada para representar o resultado da operação.

Gerenciando o Resultado

O `then` e `catch` são usados para lidar com o resultado da promessa.

Desafio Final

Antes de iniciar

Antes de iniciar um projeto de testes automatizados, temos que realizar a configuração do projeto. (Para quando já temos a stack de desenvolvimento estabelecida).

Devemos inicializar o projeto; E também instalar as dependências necessárias do que iremos utilizar.

Outro ponto, é que após toda configuração e instalação, temos que inserir as configurações para execução do projeto no arquivo package.js

Desafio Final

Antes de iniciar

Vimos também que podemos realizar pequenos testes unitários em Jest para testarmos as rotas de nossa API.

Um exemplo foi o teste onde enviamos dados válidos e validamos se a nossa aplicação retornou sucesso, e também realizamos algumas validações de cenários onde a nossa aplicação nos retornou erro.

Para esses testes utilizamos a biblioteca supertest do jest.
(permite que você faça requisições HTTP (GET, POST, PUT, DELETE, etc.) para sua aplicação e verifique as respostas de maneira fácil e eficiente).



How to test Express.js with Jest and Supertest

I recently changed from Mocha, Chai to Jest. Pretty lovely experiences. It took a while to figure out the...

By Through the binary · May 24, 2017

Desafio Final

Antes de iniciar

Até aqui desenvolvemos um teste unitário e também um teste de integração (crud) completo do cadastro de usuários:

Estrutura do Teste de Integração utilizada:

- Criar um novo usuário
- Buscar o usuário pelo ID
- Atualizar o usuário
- Deletar o usuário

Vimos também a importância de termos uma documentação detalhada para desenvolvermos os nossos testes automatizados. Bem como os relatórios simplificados.

Método de entrega

Será enviado um formulário e qualquer dúvida sobre a entrega do trabalho, pode ser enviada no canal de comunicação do discord, que a equipe entrará em contato comigo, para que eu possa ajudá-los da melhor maneira possível.

Agradecimentos

Obrigada por permitirem essa experiência enriquecedora, se eu consegui passar pelo menos 10% do conhecimento que vocês me proporcionaram, já valeu demais!



Taynara Luana

Quality Assurance Analyst | Automated Test Lead |
Lead QA Engineer | QA Engineer | Quality Assuranc...

