



Desenvolver aplicações Backend para WEB

Prof. Renan Ponick



Lema

**"Nenhuma quantidade de ansiedade faz
qualquer diferença para qualquer coisa que vai
acontecer"**

Allan watts

Motivacional



Agenda

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

15/09/2023 - Exercícios + Testes

21/09/2023 - **Avaliação**

22/09/2023 - CRUD

28/09/2023 - ORM - MySQL

29/09/2023 - Exercícios + Testes

04/10/2023 - Async - Await

05/10/2023 - **Review - Kahoot Avaliativo**

06/10/2023 - Desafios

11/10/2023 - Autenticação

12/10/2023 - FERIADO

13/10/2023 - FERIADO

18/10/2023 - Desafios

19/10/2023 - **Avaliação**

20/10/2023 - Revisão

Antes de avançar, falaremos de

ERRO

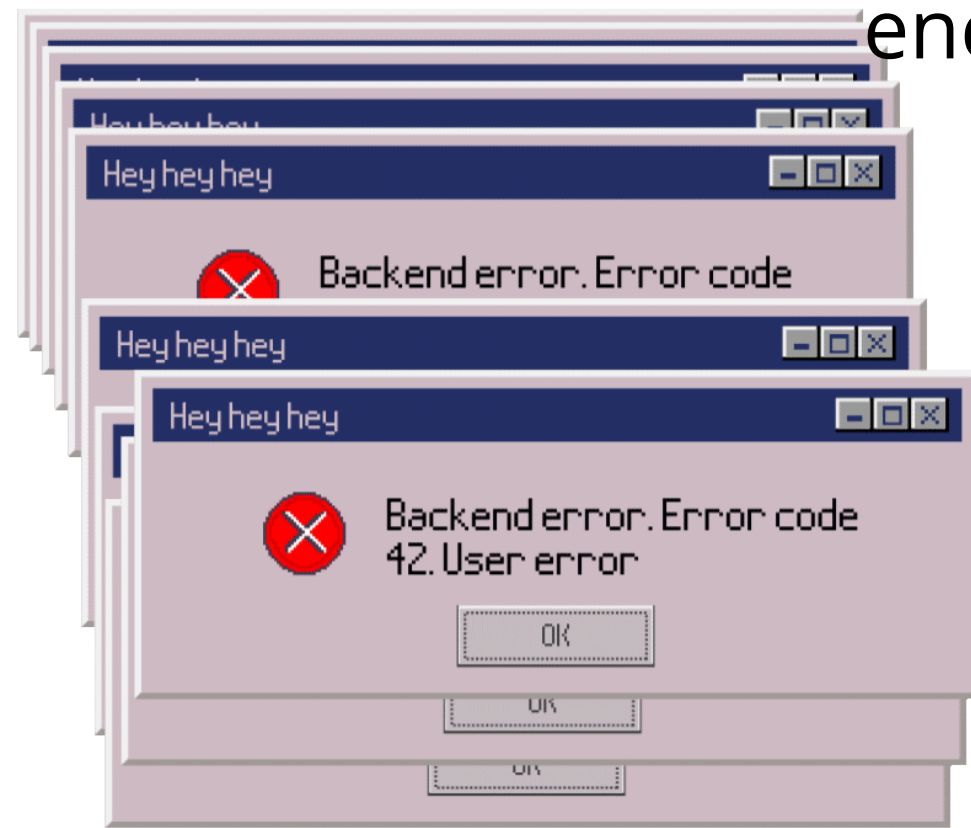
Tratativa de erro

Try Catch
Throw new Error()

Throw new Error()

throw é uma instrução que é usada para lançar exceções ou erros durante a execução do código.

Quando um erro é lançado usando throw, ele interrompe o fluxo normal do programa e busca por uma cláusula **catch** adequada para lidar com esse erro. Se nenhuma cláusula catch correspondente for encontrada, o programa pode parar de funcionar e um erro não tratado será exibido no console.



Esperar o inesperado

O throw é útil para lidar com situações em que algo inesperado acontece durante a execução do código.

Ele permite que você crie e comunique erros personalizados para indicar problemas específicos, facilitando a depuração e o gerenciamento de exceções em seu código.

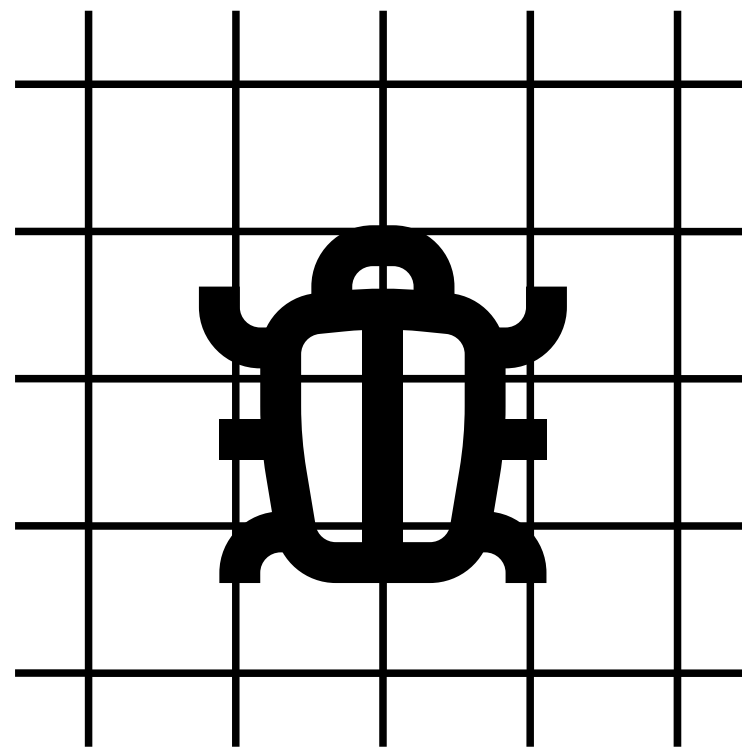


Na prática

```
1 export function exercicio1(num1, num2) {  
2   if(isNaN(num1) || isNaN(num2)) {  
3     throw new Error("Favor informar números");  
4   }  
5   return num1 + num2;  
6 }
```

Try Catch

tente pegar



É usado para envolver um bloco de código onde você espera que erros possam ocorrer, e então você pode **capturar** e tratar esses erros de forma a evitar que eles interrompam completamente a execução do programa

Estrutura base

Dentro do bloco try, é colocado o código que pode potencialmente gerar exceções. Se ocorrer um erro dentro desse bloco, a execução será interrompida e o fluxo de controle será transferido para o bloco catch. O erro é capturado na variável error que pode ser utilizada dentro do bloco catch.

```
1  try {  
2      // Bloco de código onde você espera que erros possam ocorrer  
3      // ...  
4  } catch (error) {  
5      // Bloco de código para tratar o erro  
6      // ...  
7  }
```

Na prática

```
7 app.post("/api/exercicio1", (req, res) => {  
8   try{  
9     const num1 = req.body.num1;  
10    const num2 = req.body.num2;  
11    const result = exercicio1(num1, num2);  
12  
13    res.status(201).json({ message: `Resultado: ${result}` });  
14  } catch(error) {  
15    res.status(500).json({ message: error.message });  
16  }  
17  });  
18
```

Funcionou?

REST

REST

REST não é um protocolo ou padrão, mas sim um conjunto de restrições de arquitetura.

É um estilo arquitetural que define um conjunto de princípios para projetar sistemas distribuídos e APIs da web.

Permitindo, por exemplo, que aplicações se comuniquem.

O estilo REST enfatiza a escalabilidade, a simplicidade e a interoperabilidade entre sistemas.

REST

Para uma API ser considerada do tipo RESTful, ela precisa estar em conformidade com os seguintes critérios:

- Ter uma arquitetura **cliente/servidor** formada por clientes, servidores e recursos, com solicitações gerenciadas por **HTTP**.
- Estabelecer uma comunicação **stateless**(independente) entre cliente e servidor.
- Ter uma **interface uniforme** entre os componentes para as informações serem transferidas em **um formato padronizado**.
- Ter um sistema em **camadas** que organiza os tipos de servidores (responsáveis pela segurança, pelo carregamento de carga e assim por diante) envolvidos na recuperação das informações solicitadas em **hierarquias** que o cliente não pode ver.

<https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>

e RESTFul?

REST e RESTful

- REST: conjunto de princípios de arquitetura
- RESTful: capacidade de determinado sistema aplicar os princípios de REST.



O que faz uma api ser boa?

Tenha em mente

- Separação de Responsabilidades
- Nomenclatura Significativa
- Estrutura de Diretórios
- Use Middlewares (autenticação, validação de entrada ou erros)
- Validação de Entrada
- Utilize Status HTTP Adequados
- Utilize Verbos HTTP Adequados
- Versionamento da API
- Tratamento de Erros Adequado
- Utilize Status de Resposta Claros
- Possua testes
- Realize a documentação



Estrutura

Muito parecido com o MVC que vimos em C#, trabalharemos com uma estrutura um pouco maior:

- |—— controllers
- |—— models
- |—— repositories
- |—— routes
- |—— services
- |—— middlewares
- |—— index.js ou app.js



Routes

As rotas definem os endpoints da sua API e especificam qual função do **controlador** será chamada quando um cliente fizer uma requisição a uma determinada URL.

Middleware

Middlewares são funções que podem ser usadas para pré-processar ou pós-processar uma requisição antes que ela chegue ao controlador ou após o controlador ter gerado a resposta. Eles são úteis para tarefas como autenticação, autorização, validação de entrada e manipulação de erros.

Controllers

Controladores são responsáveis por lidar com as requisições HTTP recebidas pelas rotas e coordenar a lógica de negócios. Eles recebem os dados da requisição, interagem com os **serviços e modelos**, e retornam as respostas apropriadas.



Services

Eles são responsáveis por lidar com operações mais complexas que podem envolver múltiplos modelos, operações assíncronas ou regras de negócios específicas. Isso ajuda a manter os controladores mais limpos e focados.



Models

Modelos representam as estruturas de dados que serão usadas para interagir com o banco de dados ou outras fontes de dados.



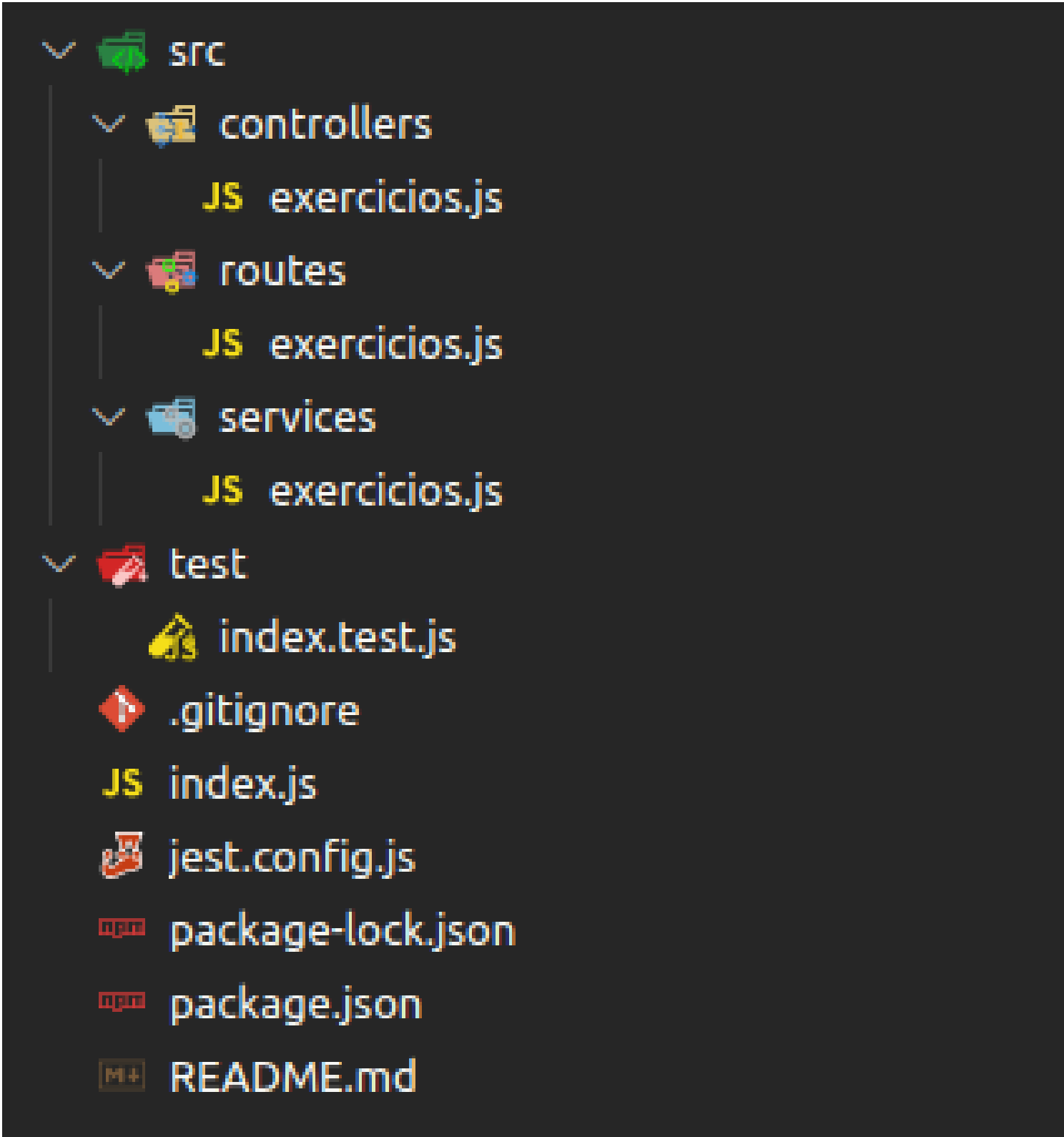
Repositories

Repositórios servem como uma camada intermediária entre a lógica de negócios da aplicação e a camada de acesso a dados.

Iniciaremos com

— controllers	Try Catch, status e send;
— routes	Métodos Get, Post (put e delete);
— services	Método em si e validações (throw);
— tests	Testes;
— index.js ou app.js	Inicialização do sistema.

Iniciaremos com



A screenshot of a file explorer interface with a dark background. The file tree is expanded to show the 'src' directory. Inside 'src', there are three subdirectories: 'controllers', 'routes', and 'services', each containing a file named 'exercicios.js'. Below the 'src' directory is a 'test' directory containing 'index.test.js'. At the root level of the project, there are files: '.gitignore', 'index.js', 'jest.config.js', 'package-lock.json', 'package.json', and 'README.md'. Each file or directory is preceded by a small icon representing its type (e.g., folder, JavaScript file, Git ignore file, etc.).

- src
 - controllers
 - JS exercicios.js
 - routes
 - JS exercicios.js
 - services
 - JS exercicios.js
- test
 - index.test.js
- .gitignore
- JS index.js
- jest.config.js
- package-lock.json
- package.json
- README.md



Bora melhorar nosso código

Lembra da classe do C#?

Service

```
...  
1  export default class ServicoExercicio {  
2      Calcular(num1, num2) {  
3          if (isNaN(num1) || isNaN(num2)) {  
4              throw new Error("Favor informar números");  
5          }  
6          return num1 + num2;  
7      }  
8  }  
9  }
```

Controller

```
1  import ServicoExercicio from "../services/exercicios.js";
2
3  const servico = new ServicoExercicio()
4
5  export default class ControllerExercicio {
6
7      Calcular(req, res) {
8          try {
9              const num1 = req.body.num1;
10             const num2 = req.body.num2;
11
12             const result = servico.Calcular(num1, num2);
13             res.status(201).json({ message: `Resultado: ${result}` });
14         } catch (error) {
15             res.status(500).json({ message: error.message });
16         }
17     }
18
19 }
```


Router

```
1  import express from "express";
2  import ControllerExercicio from "../controllers/exercicios.js";
3
4  const router = express.Router();
5
6  const controllers = new ControllerExercicio()
7
8  router.post("/api/exercicio1", controllers.Calcular);
9
10 export default router;
```

Index.js

```
1  import express from "express";
2  import routers from "../src/routes/exercicios.js";
3
4  const app = express();
5
6  app.use(express.json());
7
8  app.use(routers);
9
10 const PORT = process.env.PORT || 3000;
11
12 app.listen(PORT, () => {
13   console.log(`Servidor rodando na porta ${PORT}`);
14 });
15
```

Funcionou?



Exercícios

Ajuste os teste unitário para que possuam também verificações de erro. Tratativa de erros com throw new error

Ajuste a estruturação dos arquivos dos exercícios conforme o que vimos durante a aula.