

Web Fullstack na Prática: Sistema de Cadastro de Vídeos

Sumário

O que vamos construir?	3
O que você vai aprender:	3
Frontend (O Salão)	3
Backend (A Cozinha)	3
Como eles trabalham juntos?.....	4
Começando com o BackEnd	4
Extensões do VSCode que iremos usar no backend.....	4
Organização das pastas do projeto	4
Acessando o terminal de linha de comando pelo VSCODE.	5
Iniciando o gerenciador do GIT na pasta raiz	5
Iniciando o gerenciador de pacotes do NodeJS na pasta raiz.....	6
Instalando todas as bibliotecas	6
Configurando o servidor com Fastify.....	7
Arquivo ENV.....	8
Como usar no Node.js.....	9
Configurando o Git na pasta	10
Ignorando pastas e arquivos no commit.....	10
Commit do projeto no Git e publicando no GitHub	10
Persistência com o banco de dados.....	11
Criando o arquivo de conexão com o banco de dados	12
Criando a tabela vídeos.....	13
Pausa para o Commit.....	14
Criando o arquivo de persistência com o banco	14
Criando as Rotas	15
Arquivo de Rotas.....	17

Web Fullstack na Prática:

Sistema de Cadastro de Vídeos

Você já se perguntou como plataformas de streaming ou redes sociais gerenciam milhares de arquivos e informações? Neste curso, você vai sair da teoria e colocar as mãos na massa para construir, do zero, uma aplicação web completa de **Cadastro de Vídeos**.

O que vamos construir?

Vamos desenvolver um sistema onde o usuário pode cadastrar, listar, atualizar e remover vídeos (o famoso **CRUD**). Para isso, utilizaremos as tecnologias mais requisitadas pelo mercado de trabalho atual:

- **Frontend (A Interface):** Usaremos o **React**, a biblioteca de JavaScript mais popular do mundo, para criar uma interface moderna, rápida e responsiva.
- **Backend (Os Bastidores):** Vamos construir o coração da aplicação com **Node.js**, lidando com a lógica de negócios e as rotas do sistema.
- **Banco de Dados (O Armazenamento):** Utilizaremos o **MySQL** para aprender como salvar e organizar as informações de forma profissional e segura.

O que você vai aprender:

1. **Conectar as pontas:** Entender como o Frontend "conversa" com o Backend.
2. **Lógica Fullstack:** Criar uma API para gerenciar os dados dos vídeos.
3. **Persistência de Dados:** Dominar comandos SQL para ler e gravar informações no banco de dados.
4. **Componentização:** Criar interfaces reutilizáveis e organizadas com React.

Frontend (O Salão)

O **Frontend** é tudo o que o cliente vê, toca e interage. No restaurante, seria a decoração, o cardápio, as mesas, as cadeiras e a forma como o garçom te atende.

- **Na programação:** É a interface do site ou aplicativo. São as cores, os botões, as fontes, as animações e o layout que aparece na sua tela.
- **Tecnologias comuns:** HTML (estrutura), CSS (estilo) e JavaScript (interação).

Backend (A Cozinha)

O **Backend** é o que acontece "nos bastidores". No restaurante, é a cozinha. Você não vê o que está acontecendo lá dentro, mas é onde a mágica acontece: o pedido é processado, os ingredientes são retirados da despensa (banco de dados) e o prato é preparado.

- **Na programação:** É a parte lógica que o usuário não vê. Ele processa os dados, verifica se sua senha está correta, salva suas compras e comunica-se com o banco de dados.
 - **Tecnologias comuns:** Python, Node.js, PHP, Java e Bancos de Dados (como MySQL).
-

Como eles trabalham juntos?

Quando você clica em um botão "Comprar" (**Frontend**), o site envia um pedido para a "cozinha" (**Backend**). O Backend verifica se você tem saldo, retira o item do estoque e, depois de tudo pronto, envia uma mensagem de "Sucesso" de volta para o Frontend mostrar na sua tela.

CRUD é um acrônimo que representa as quatro operações fundamentais utilizadas em aplicações que gerenciam dados, geralmente em um banco de dados.

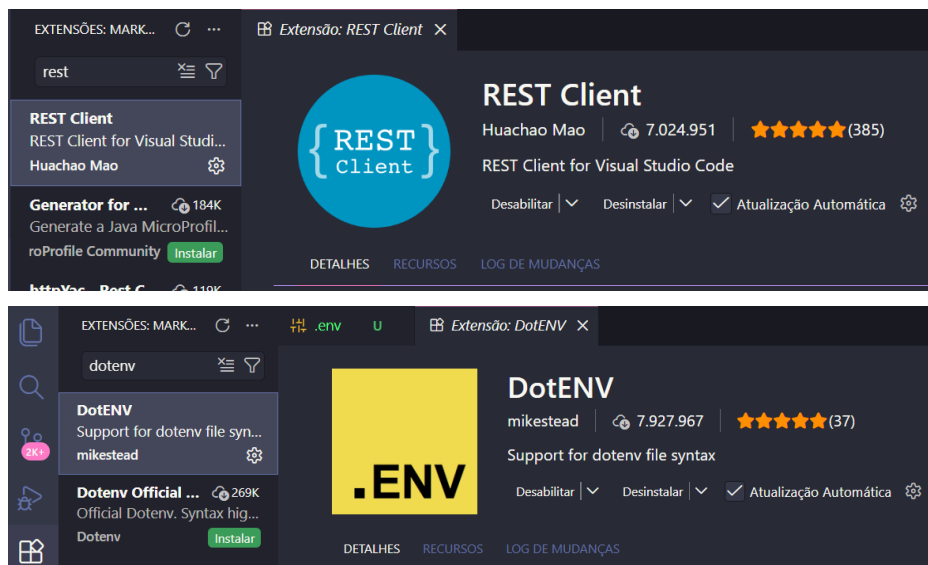
O significado de cada letra é:

- **C - Create (Criar):** Inserir novos registros ou informações no sistema.
- **R - Read (Ler):** Consultar ou visualizar dados que já estão armazenados.
- **U - Update (Atualizar):** Modificar ou editar informações existentes.
- **D - Delete (Deletar):** Remover ou excluir registros do banco de dados.

Começando com o BackEnd

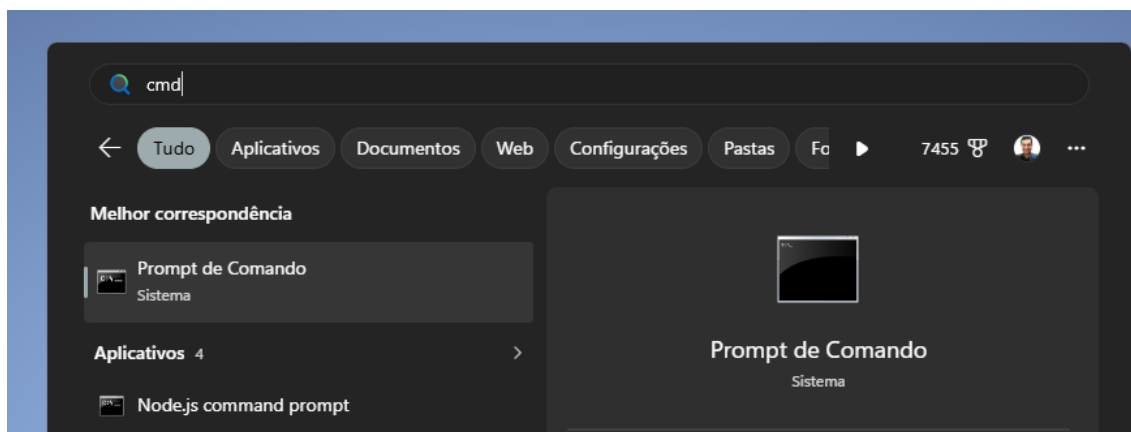
Vamos criar toda a parte de **backend**, logo após faremos a interface no **frontend** e em uma terceira etapa faremos a comunicação entre os dois.

Extensões do VSCode que iremos usar no backend



Organização das pastas do projeto

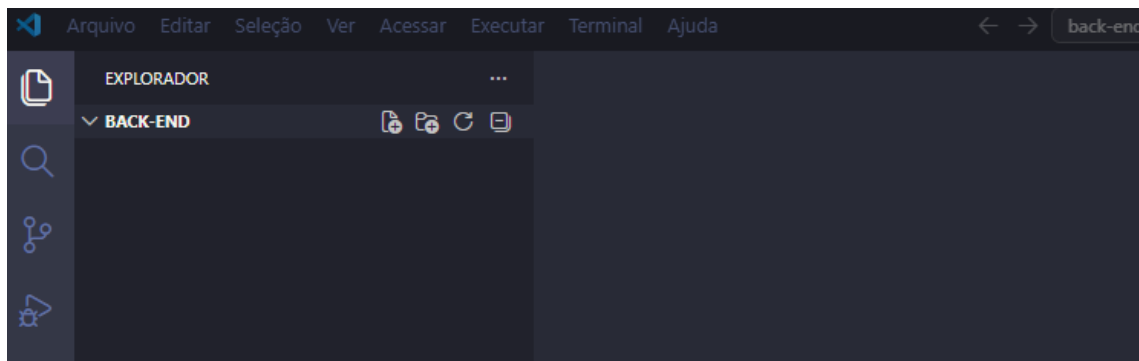
Vamos criar a estrutura de pastas do projeto a começar pela pasta raiz (principal), esta pasta se chamara "gestor-videos" e dentro dela será criado a pasta back-end, vamos usar o terminal **cmd** (Prompt de Comandos).



Escolha qual pasta do Windows irá criar a pasta Raiz do projeto e em seguida digite no terminal os comandos a seguir;

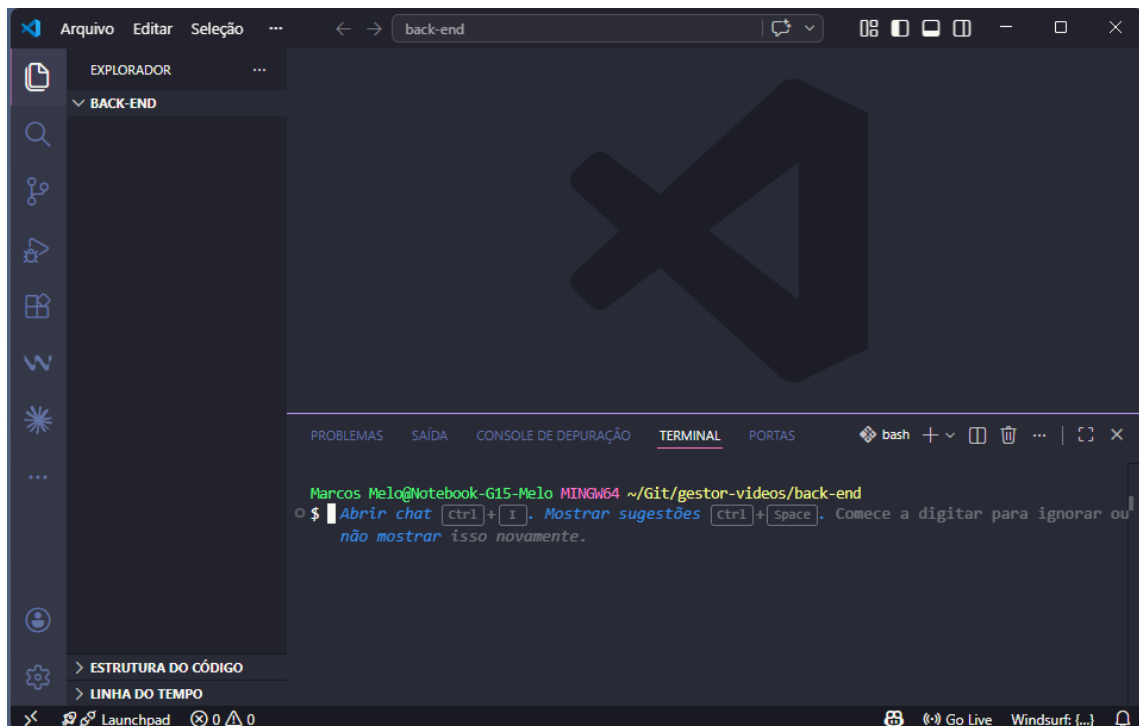
```
md gestor-videos
cd gestor-videos
md back-end
cd back-end
code .
```

O ultimo comando **code .** abre o VSCODE para digitarmos dentro da pasta o código do projeto.



Acessando o terminal de linha de comando pelo VSCODE.

Vamos ter que executar comandos no terminal várias vezes durante o desenvolvimento do projeto, para acessar o terminal no VSCODE digite **ctrl + j**.



Iniciando o gerenciador do GIT na pasta raiz

Antes de começar a programa vamos ativar o git para gerenciar, versionar e publicar o projeto no GITHUB.

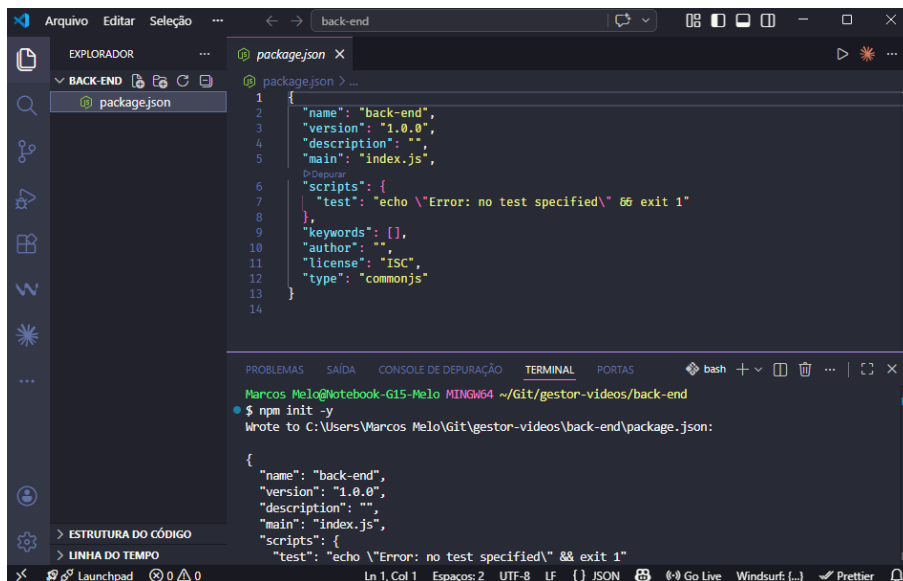
```
git init
```

Iniciando o gerenciador de pacotes do NodeJS na pasta raiz

No terminal digite o comando abaixo para inicializar o gerenciador de pacotes NPM dentro da pasta raiz.

```
npm init -y
```

Este comando irá criar o arquivo json chamado package.json contendo as configurações do projeto.

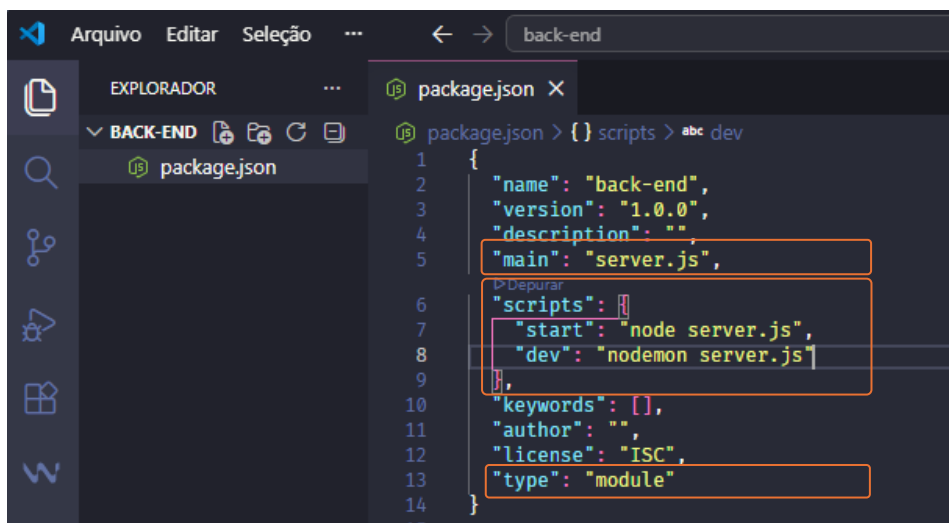


The screenshot shows the Visual Studio Code editor with the 'package.json' file open. The file content is as follows:

```
1 {
2   "name": "back-end",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "type": "commonjs"
13 }
14
```

The terminal at the bottom shows the command `$ npm init -y` and the output: `Write to C:\Users\Marcos Melo\Git\gestor-videos\back-end\package.json:` followed by the same JSON content.

Altere as informações do tipo de **importação / exportação** e **script** do arquivo package.json como na imagem a seguir;



The screenshot shows the Visual Studio Code editor with the 'package.json' file open. The file content is as follows:

```
1 {
2   "name": "back-end",
3   "version": "1.0.0",
4   "description": "",
5   "main": "server.js",
6   "scripts": {
7     "start": "node server.js",
8     "dev": "nodemon server.js"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "type": "module"
14 }
```

Orange boxes highlight the changes: `"main": "server.js",`, the `"scripts"` block, and `"type": "module"`.

Instalando todas as bibliotecas

Instale a biblioteca nodemon

```
npm install nodemon -D
```

Instale a biblioteca dotenv

```
npm install dotenv -D
```

Instale a biblioteca do Fastify

```
npm install fastify
```

Instale a biblioteca de conexão com o banco de dados MySQL

```
npm install mysql2
```

The screenshot shows the Visual Studio Code interface with the Explorer on the left showing a project structure under 'BACK-END' with files 'node_modules', 'package-lock.json', and 'package.json'. The main editor displays the content of 'package.json' with the following code:

```
1 {
2   "name": "back-end",
3   "version": "1.0.0",
4   "description": "",
5   "main": "server.js",
6   "scripts": {
7     "start": "node server.js",
8     "dev": "nodemon server.js"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "type": "module",
14  "devDependencies": {
15    "dotenv": "^17.4.2",
16    "nodemon": "^3.1.14"
17  },
18  "dependencies": {
19    "fastify": "^5.8.5",
20    "mysql2": "^3.22.2"
21  }
22 }
```

Configurando o servidor com Fastify

Criaremos agora o arquivo principal da nossa API do backend, o **server.js**, neste arquivo será configurado o servidor http usando a biblioteca **Fastify**.

Digite o código como a seguir;

The screenshot shows the Visual Studio Code interface with the Explorer on the left showing a file named 'server.js'. The main editor displays the content of 'server.js' with the following code:

```
1 import { fastify } from 'fastify';
2 const PORT = 3333;
3
4 console.log('Variáveis de ambiente carregadas:', { PORT });
5
6 const server = fastify();
7
8 server.get('/', async (request, reply) => {
9   return { message: 'API server - Gestor de Vídeos' };
10 });
11
12 server.listen({port:PORT}, (err, address) => {
13   if (err) {
14     console.error(err);
15     process.exit(1);
16   }
17   console.log(`Servidor rodando em ${address}`);
18 });
```

Após digitar o código teste o servidor executando o código a seguir no terminal;

```
npm run dev
```

```
server.js
1 import { fastify } from 'fastify';
2
3 const PORT = 3333;
4
5
6

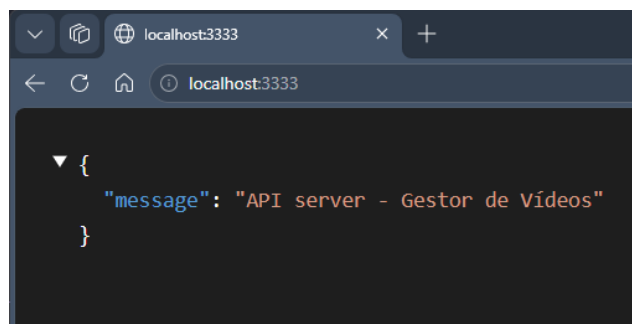
Marcos Melo@Notebook-G15-Melo MINGW64 ~/Git/gestor-videos/back-end
o $ npm run dev

> back-end@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.14
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Servidor rodando em http://[::1]:3333
[]
```

Acesse no navegador pelos endereços

[\[::1\]:3333](http://[::1]:3333) ou <http://localhost:3333>

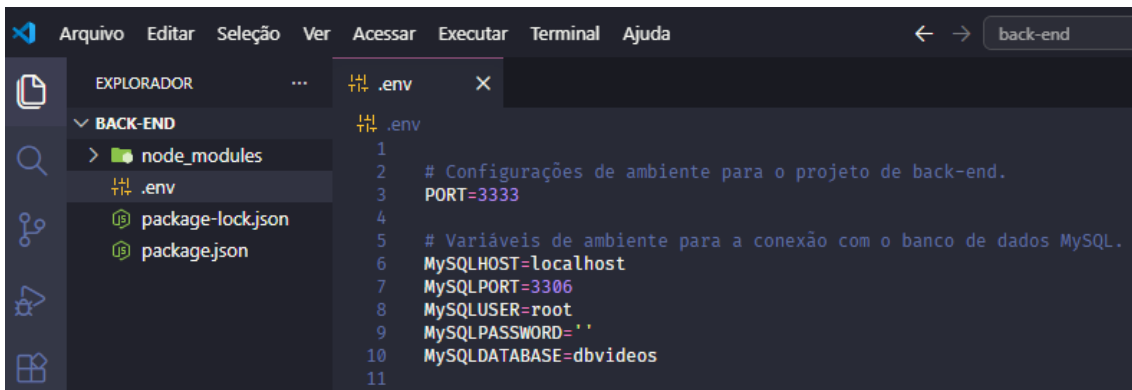


Arquivo ENV

O arquivo `.env` é um arquivo de texto simples usado em aplicações [Node.js](#) para armazenar **variáveis de ambiente**. Ele permite separar as configurações e dados sensíveis da aplicação do código-fonte propriamente dito.

Para que serve?

- **Segurança:** Protege informações sensíveis, como senhas de banco de dados, chaves de API e segredos de tokens, evitando que sejam expostos no código-fonte.
- **Configuração por ambiente:** Facilita o uso de diferentes configurações para desenvolvimento, teste e produção sem precisar alterar o código.
- **Centralização:** Armazena todas as configurações em um único local no formato CHAVE=valor.



```
Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda
EXPLORADOR
BACK-END
node_modules
.env
package-lock.json
package.json
.env
1
2 # Configurações de ambiente para o projeto de back-end.
3 PORT=3333
4
5 # Variáveis de ambiente para a conexão com o banco de dados MySQL.
6 MySQLHOST=localhost
7 MySQLPORT=3306
8 MySQLUSER=root
9 MySQLPASSWORD=''
10 MySQLDATABASE=dbvideos
11
```

Como usar no Node.js

Para acessar esses valores, a aplicação lê o arquivo e popula o objeto global `process.env`.

A forma que iremos acessar as informações deste arquivo é através da **Biblioteca dotenv**: É a solução mais comum para versões anteriores do Node.js.

Digite no terminal o comando a seguir para instalar a biblioteca `dotenv`.

```
npm install dotenv -D
```

Dica importante: Nunca envie seu arquivo `.env` para o controle de versão (como o Git). Adicione-o ao seu arquivo `.gitignore` para garantir que seus segredos não sejam compartilhados publicamente.

Após criar o arquivo `.env` contendo as variáveis de ambiente, edite o arquivo `server.js` para acessar as variáveis acrescentando as linhas de código a seguir;



```
server.js
server.js > ...
1 import { fastify } from 'fastify';
2 import 'dotenv/config';
3 const { PORT } = process.env;
4 console.log('Variáveis de ambiente carregadas:', { PORT });
5
6 const server = fastify();
7
8 server.get('/', async (request, reply) => {
9   return { message: 'API server - Gestor de Vídeos' };
10 });
11
12 server.listen({port:PORT}, (err, address) => {
13   if (err) {
14     console.error(err);
15     process.exit(1);
16   }
17   console.log(`Servidor rodando em ${address}`);
18 });
19
20
```

Execute o servidor novamente e observe que a variável de ambiente `PORT` está sendo acessada no arquivo `server.js`

```
server.js
1 import { fastify } from 'fastify';
2 import 'dotenv/config';
3 const { PORT } = process.env;
4
5 console.log('Variáveis de ambiente carregadas:', { PORT });
6
7 const server = fastify();
8
9 server.get('/', async (request, reply) => {
10   return { message: 'API server - Gestor de Vídeos' };
11 });
12
13 server.listen({port:PORT}, (err, address) => {
14   if (err) {
15     console.error(err);
16     process.exit(1);
17   }
18   console.log(`Servidor rodando em ${address}`);
19 });
20
21
```

```
Terminal
Marcos Melo@Notebook-G15-Melo MINGW64 ~/Git/gestor-videos/back-end
$ npm run dev
> back-end@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.14
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Variáveis de ambiente carregadas: { PORT: '3333' }
Servidor rodando em http://[::1]:3333
```

Configurando o Git na pasta

Se há a intenção de colocar este projeto no github, é preciso configurar o GIT na pasta backend para poder fazer os commits do projeto.

Digite no terminal o comando a seguir para ativar o GIT na pasta;

```
git init
```

Ignorando pastas e arquivos no commit

Para ignorar arquivos e pastas que não devem ser colocados no repositório do github, crie o arquivo **.gitignore** na raiz e digite em cada linha o nome de cada arquivo e pasta que deseja ignorar nos commits do GIT, assim pastas muito grandes que não precisam ir para o repositório, exemplo, a pasta **node_modules** e arquivos com informações sensíveis (sigilosas), por exemplo, o arquivo **.env** serão ignorados.

```
EXPLORADOR
GESTOR-VIDEOS
  back-end
  node_modules
  .env
  .gitignore
  package-lock.json
  package.json
  server.js
```

```
back-end > .gitignore
1 node_modules
2 .env
```

Commit do projeto no Git e publicando no GitHub

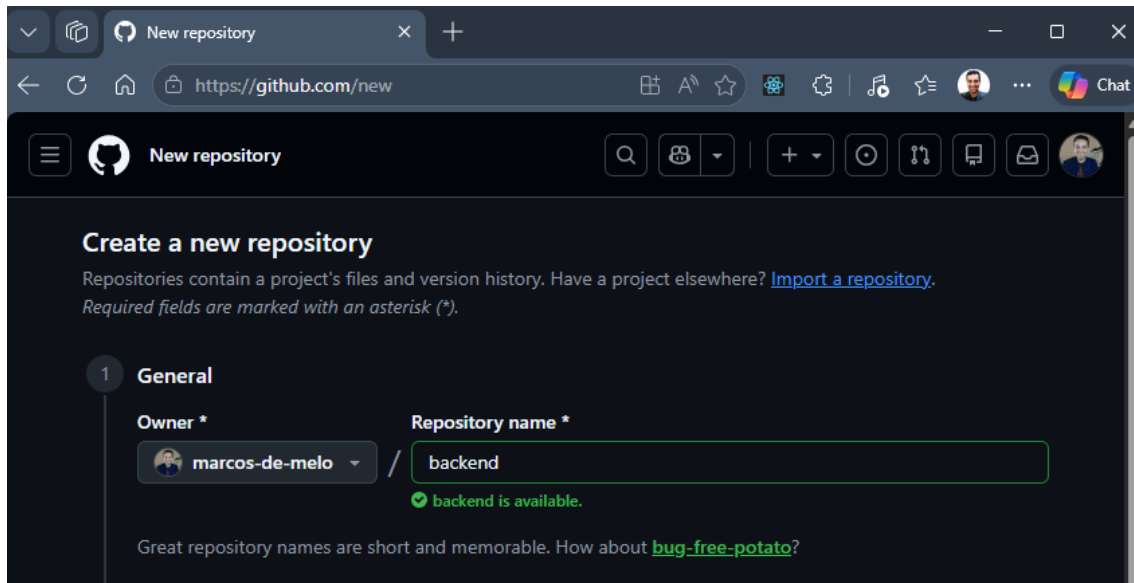
Pausa no projeto para publicar no GitHub, vamos fazer o commit do projeto até o momento e subir o projeto para o GitHub, assim poderemos dar continuidade nas próximas aulas.

Antes de fazer o commit e enviar os arquivos para o GITHUB digite os comandos a seguir caso já não tenha feito para configuração do seu usuário no GIT. Sem esta configuração o GIT não fará o commit.

```
git config --global user.name "seu nome aqui"
```

```
git config --global user.email "seu e-mail aqui"
```

Acesse a sua conta no GitHub e nos seus repositórios crie o repositório com o mesmo nome da pasta **backend** do projeto.



Após criar o repositório digite os comandos a seguir fornecidos na criação do repositório no terminal para realizar o commit e enviar os arquivos do projeto para o GITHUB.

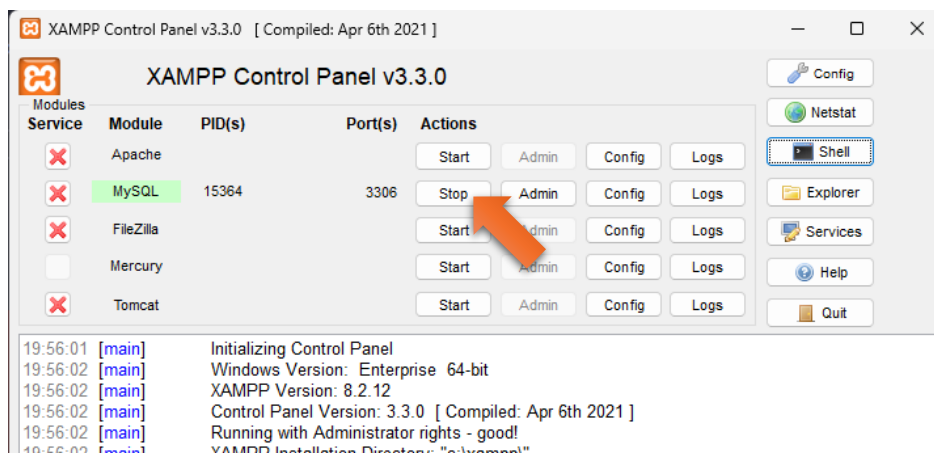
```
git add .
git commit -m "primeiro commit - criação do servidor por fastify"
git branch -M main
git remote add origin https://github.com/sua-conta-aqui/backend.git
git push -u origin main
```

Persistência com o banco de dados

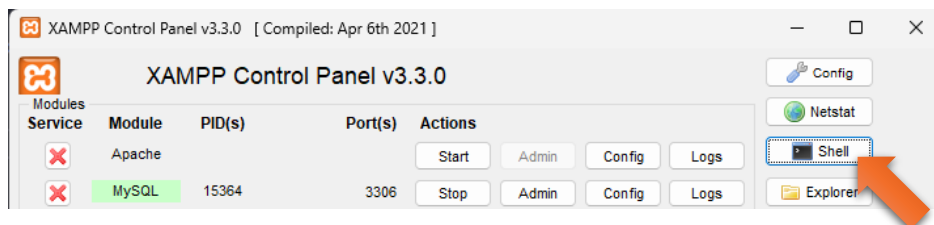
Chegou a hora de criar o banco de dados que será persistido na nossa API do backend, então antes que qualquer procedimento devemos iniciar o servidor do banco de dados que iremos criar acessar e persistir.

O banco de dados relacional que iremos utilizar será o MySQL, o pacote de desenvolvimento web, o XAMPP possui o **MariaDB** que é um Fork do MySQL Server e apesar do nome vamos tratá-lo como se fosse o MySQL.

Abrindo o XAMPP Control pressione o botão Start do MySQL para iniciar o servidor.



Clique no botão Shell para abrir o terminal do XAMPP para acessar o MySQL onde vamos criar o banco de dados que vamos usar.



No terminal que abrir, digite o código a seguir para acessar o servidor do MySQL.

```
C:\> Administrador: XAMPP for Windows

Setting environment for using XAMPP for Windows.
Marcos Melo@NOTEBOOK-G15-ME c:\xampp
# mysql -u root -p
```

Aviso: se o usuário root do seu servidor MySQL possui senha, digite-a, caso contrário só pressione **enter** para acessar.

```
C:\> Administrador: XAMPP for Windows - mysql -u root -p

Setting environment for using XAMPP for Windows.
Marcos Melo@NOTEBOOK-G15-ME c:\xampp
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Na tela seguinte digite o comando mostrado na imagem a seguir para criar o banco de dados para ser usado no nosso projeto.

```
C:\> Administrador: XAMPP for Windows - mysql -u root -p

MariaDB [(none)]> create database dbvideos;
Query OK, 1 row affected (0.002 sec)
```

Se nenhum erro aparecer o banco deve ter sido criado, mas para verificar se foi mesmo criado digite no terminal o comando a seguir;

```
show databases;
```

Criando o arquivo de conexão com o banco de dados

Crie na pasta raiz o arquivo **db.js**, este arquivo será usado para conectar com o servidor de banco de dados e consequentemente com o banco de dados que criamos.

```
1 import dotenv from 'dotenv';
2 import mysql from 'mysql2/promise'; // Importante usar a versão /promise
3
4 dotenv.config();
5
6
7 const { MySQLHOST, MySQLPORT, MySQLUSER, MySQLPASSWORD, MySQLDATABASE } = process.env;
8
9 const pool = mysql.createPool({
10   host: MySQLHOST,
11   port: MySQLPORT || 3306, // Porta padrão do MySQL é 3306 se não for especificada no .env
12   user: MySQLUSER,
13   password: MySQLPASSWORD,
14   database: MySQLDATABASE,
15   waitForConnections: true,
16   connectionLimit: 10,
17   queueLimit: 0,
18   ssl: false // Caso precise de SSL no futuro, mude para { rejectUnauthorized: false }
19 });
20
21
22 export const sql = pool;
23
```

Criando a tabela vídeos

O arquivo que será criado agora é só para criar no banco de dados a tabela vídeos e será usado somente uma vez, poderíamos ter criado a tabela já anteriormente quando criamos o banco, mas deixamos para criar agora, para demonstrar como acessar o banco de dados através do arquivo de conexão **db.js**.

```
1 import { sql } from "./db.js"; // Importa a conexão com o banco de dados MySQL
2
3
4 const createTableQuery = `
5 CREATE TABLE IF NOT EXISTS videos (
6   id VARCHAR(255) PRIMARY KEY,
7   title VARCHAR(255),
8   description TEXT,
9   duration INT
10 );
11 `; // Define a consulta SQL para criar a tabela "videos" se ela não existir
12
13 // O mysql2 usa o método .query() que retorna uma Promise
14 sql.query(createTableQuery)
15   .then(() => {
16     console.log("Tabela 'videos' criada ou já existente com sucesso no MySQL");
17   })
18   .catch((err) => {
19     console.error("Erro ao criar a tabela no MySQL:");
20     console.error(err.message);
21   });
22
```

Para executar o arquivo create-table.js digite o comando a seguir no terminal;

```
node create-table.js
```

Se tudo deu certo, no terminal aparecerá a mensagem no log do terminal de tabela criada como mostra abaixo na imagem.

```
1 import { sql } from './db.js'; // Importa a conexão com o banco de dados MySQL
2
3
4 const createTableQuery = `
5 CREATE TABLE IF NOT EXISTS videos (
6   id VARCHAR(255) PRIMARY KEY,
7   title VARCHAR(255),
8   description TEXT,
9   duration INT
10 );
11 `; // Define a consulta SQL para criar a tabela "videos" se ela não existir
12
13 // O mysql2 usa o método .query() que retorna uma Promise
14 sql.query(createTableQuery)
15   .then(() => {
16     console.log("Tabela 'videos' criada ou já existente com sucesso no MySQL");
17   })
18   .catch((err) => {
19     console.error("Erro ao criar a tabela no MySQL:");
20     console.error(err.message);
21   });
22
```

```
Marcos_Melo@Notebook-G15-Melo MINGW64 ~/Git/gestor-videos/back-end (main)
$ node create-table.js
injected env (6) from .env // tip: # multiple files with .env: ['.env.local',
'.env'] }
Tabela 'videos' criada ou já existente com sucesso no MySQL
```

Pausa para o Commit

Vamos fazer o **commit** e **push** para o **GITHUB** de tudo o que foi feito até agora.

```
git add .
git commit -m "Conexão com o banco de dados e criação da tabela videos"
git push
```

Criando o arquivo de persistência com o banco

O arquivo que vamos criar agora será o responsável por fazer o CRUD (Create, Read, Update, Delete) no banco dados.

O código do arquivo será em formato de classe contendo os métodos que irão fazer a persistência no banco.

```
database-mysql.js U X
back-end > database-mysql.js > ...
1 import { randomUUID } from "node:crypto";
2 import { sql } from "./db.js";
3
4 export class DatabaseMySQL {
5
6 // Listagem de vídeos, com opção de busca por título usando o operador LIKE
7 async list(search) {
8     let videos;
9
10    if (search) {
11        // No mysql2, usamos o caractere "?" como placeholder para evitar SQL Injection
12        // O resultado vem como [videos, fields], por isso usamos a desestruturação [videos]
13        [videos] = await sql.execute(
14            'SELECT * FROM videos WHERE title LIKE ?',
15            [`%${search}%`]
16        );
17    } else {
18        [videos] = await sql.execute('SELECT * FROM videos');
19    }
20
21    return videos;
22 }
23
24 // Criação de um novo vídeo, gerando um ID único usando randomUUID
25 async create(video) {
26     const videoId = randomUUID();
27     const { title, description, duration } = video;
28
29     // No mysql2, passamos os valores em um array como segundo argumento
30     await sql.execute(
31         'INSERT INTO videos (id, title, description, duration) VALUES (?, ?, ?, ?)',
32         [videoId, title, description, duration]
33     );
34 }
35
36 // Atualização de um vídeo específico usando o ID
37 async update(id, video) {
38     const { title, description, duration } = video;
39     await sql.execute(
40         'UPDATE videos SET title = ?, description = ?, duration = ? WHERE id = ?',
41         [title, description, duration, id]
42     );
43 }
44
45 // Exclusão de um vídeo específico usando o ID
46 async delete(id) {
47     await sql.execute('DELETE FROM videos WHERE id = ?', [id]);
48 }
49 }
```

Criando as Rotas

Em Node.js, rotas representam os caminhos que um aplicativo usa para responder às requisições do cliente (como um navegador ou aplicativo de celular). Elas utilizam os métodos HTTP para definir qual ação será executada no servidor.

Acrescente ao arquivo server.js as linhas de código das rotas na imagem a seguir que estão destacadas.

```
server.js > server.put("/videos/:id") callback
1  import { fastify } from 'fastify';
2  import { DatabaseMySQL } from './database-mysql.js';
3  import 'dotenv/config';
4  const { PORT } = process.env;
5
6  console.log('Variáveis de ambiente carregadas:', { PORT });
7
8  const server = fastify();
9
10 server.get('/', async (request, reply) => {
11   return { message: 'API server - Gestor de Vídeos' };
12 });
13
14
15
16 // Criando uma instância da classe DatabaseMySQL para
17 // interagir com o banco de dados
18 const database = new DatabaseMySQL();
19
20
21 // Rota para criar um novo vídeo, recebendo os dados no corpo
22 // da requisição e usando o método create do database
23 server.post("/videos", async (request, reply) => {
24   const { title, description, duration } = request.body;
25   await database.create({
26     title,
27     description,
28     duration
29   });
30   console.log(await database.list());
31   return reply.status(201).send();
32 })
33
34 // Rota para listar os vídeos, com opção de busca por título
35 // usando query string e o método list do database
36 server.get("/videos", async (request) => {
37   const search = request.query.search;
38   console.log(search);
39   const videos = await database.list(search);
40   return videos
41 })
42
43 // Rota para atualizar um vídeo específico, recebendo o
44 // ID na URL e os dados no corpo da requisição, usando o
45 // método update do database
46 server.put("/videos/:id", async (request, reply) => {
47
48   const videoId = request.params.id;
49   const { title, description, duration } = request.body;
50
51   const video = await database.update(videoId, {
52     title,
53     description,
54     duration,
55   });
56
```

```

56
57     return reply.status(204).send();
58 }
59
60 // Rota para excluir um vídeo específico, recebendo o ID na
61 // URL e usando o método delete do database
62 server.delete("/videos/:id", async (request, reply) => {
63     const videoId = request.params.id;
64     await database.delete(videoId);
65     return reply.status(204).send();
66 })
67
68 server.listen({port:PORT}, (err, address) => {
69     if (err) {
70         console.error(err);
71         process.exit(1);
72     }
73     console.log(`Servidor rodando em ${address}`);
74 });
75

```

Arquivo de Rotas

As rotas criadas anteriormente foram feitas para serem acessadas por requisições http pelo navegador, porém ainda não temos o navegador, por isso vamos usar uma extensão do VSCODE para simular as requisições http e então testar as rotas.

Certifique-se de ter instalado a extensão REST CLIENT no VSCODE.



Com esta extensão instalada vamos criar um arquivo chamado **routes.http**, neste arquivo vamos colocar todas as requisições http das rotas que criamos.

rotes.http U X

rotes.http > ...

```
Send Request
1 POST http://localhost:3333/videos
2 Content-Type: application/json
3
4 {
5   "title": "Video teste",
6   "description": "Este é o primeiro video de Node.js com MySQL",
7   "duration": 900
8 }
9 ###
10
Send Request
11 GET http://localhost:3333/videos
12 # GET http://localhost:3333/videos
13
14 ###
15
Send Request
16 PUT http://localhost:3333/videos/id-do-video-a-ser-atualizado
17 Content-Type: application/json
18
19 {
20   "title": "Auterei este video",
21   "description": "descrição atualizada",
22   "duration": 345
23 }
24
25 ###
26
Send Request
27 DELETE http://localhost:3333/videos/id-do-video-a-ser-deletado
28
```