

# **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
2. Backend (A Cozinha) .....	3
Como eles trabalham juntos?.....	4
1ª Etapa: 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.....	8
Configurando o Git na pasta .....	9
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.....	12
Pausa para o Commit.....	13
Criando o arquivo de persistência com o banco .....	13

# 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).

### 2. 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.

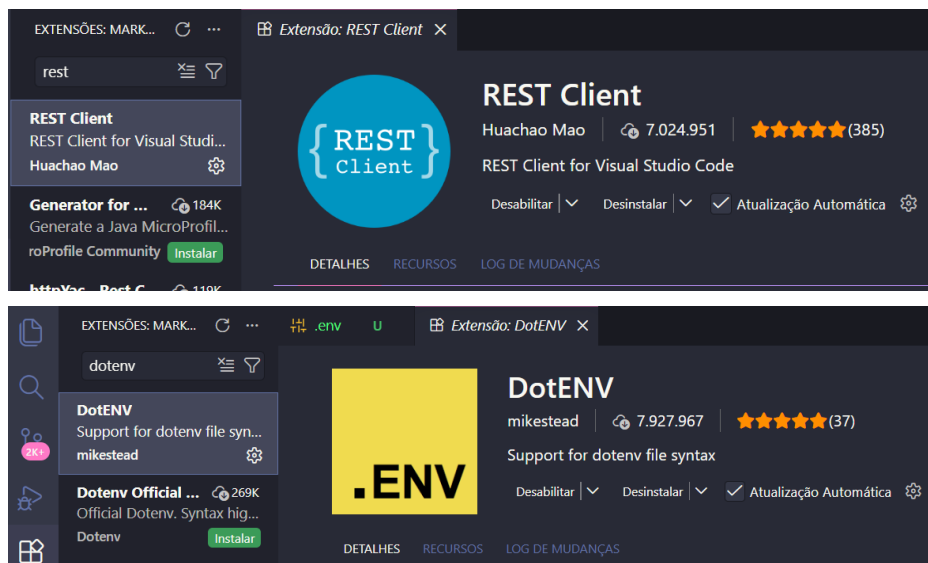
O significado de cada letra é:

- **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.

## 1ª Etapa: Começando com o BackEnd

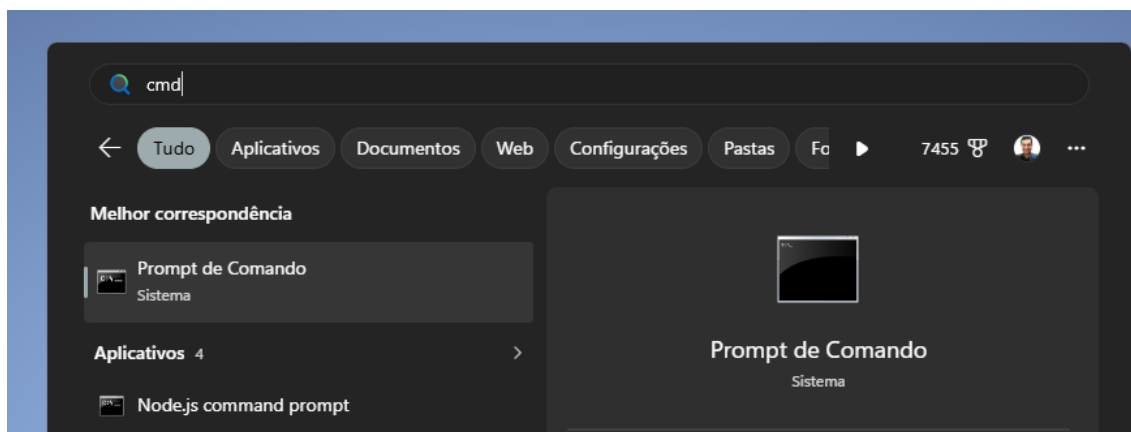
Vamos criar toda a parte de **backend**, logo após faremos a interface no **frontend** e em uma terceira etapa faremos a comunicação entre os dois.

### Extensões do VSCode que iremos usar no backend



### Organização das pastas do projeto

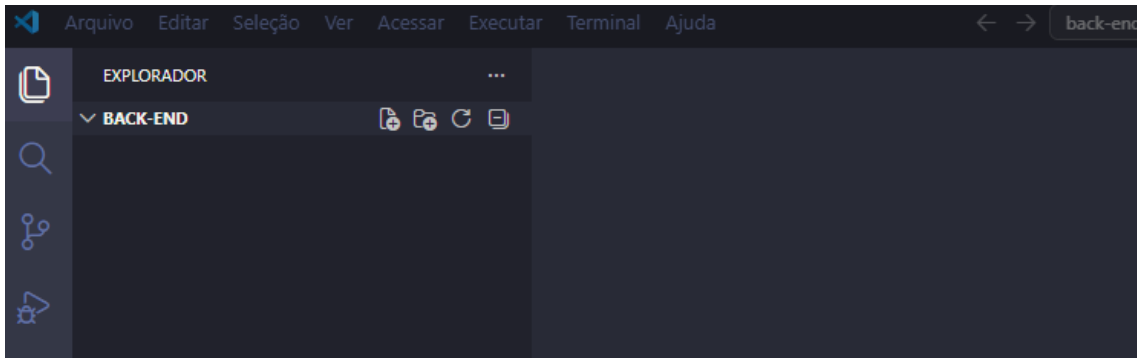
Vamos criar a estrutura de pastas do projeto a começar pela pasta raiz (principal), esta pasta se chamara "gestor-videos" e dentro dela será criado a pasta back-end, vamos usar o terminal **cmd** ( Prompt de Comandos).



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

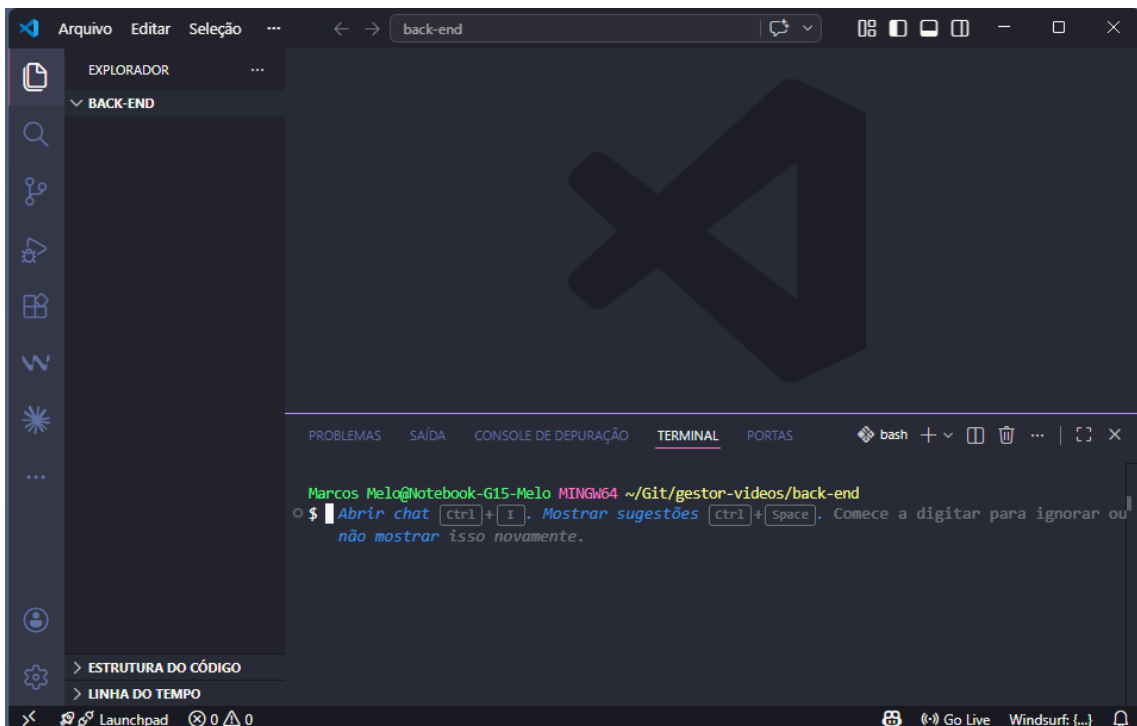
```
md gestor-videos
cd gestor-videos
md back-end
cd back-end
code .
```

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 varias 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.

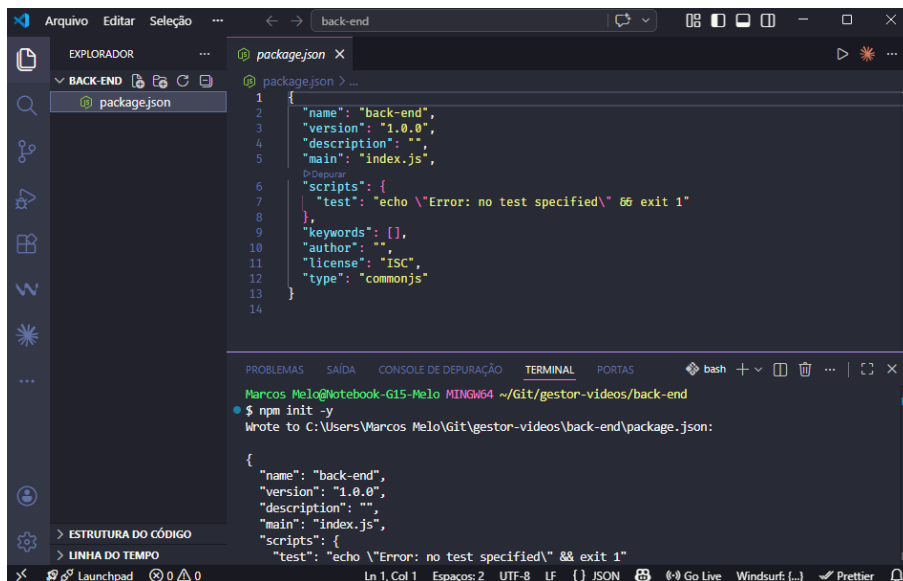
```
git init
```

## 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.



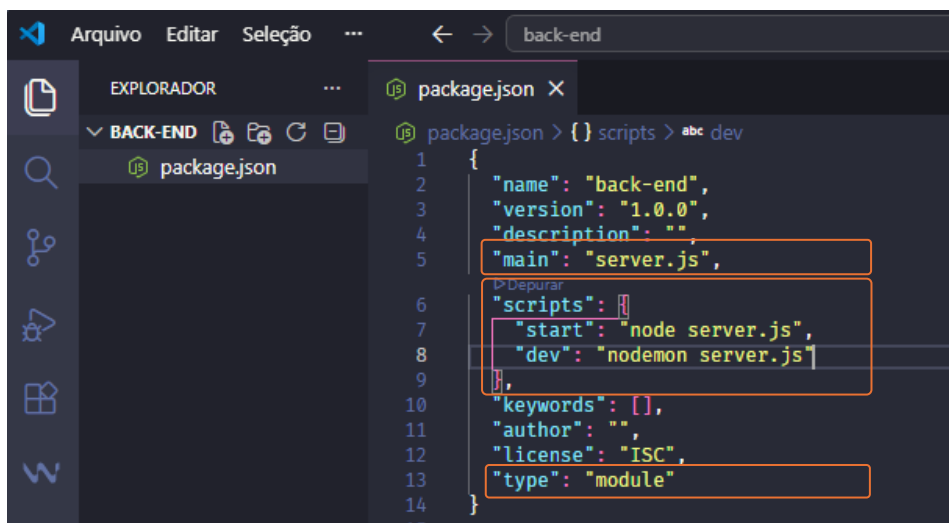
The screenshot shows the Visual Studio Code editor with a file explorer on the left showing the 'package.json' file. The main editor displays the content of 'package.json' with the following JSON structure:

```
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 execution and the resulting JSON:

```
 Marcos Melo@notebook-G15-Melo MINGW64 ~/Git/gestor-videos/back-end
• $ npm init -y
Write to C:\Users\Marcos Melo\Git\gestor-videos\back-end\package.json:
{
  "name": "back-end",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  }
}
```

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 JSON structure is modified 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' is now 'server.js', 'scripts' now includes 'start' and 'dev' scripts, and 'type' is now '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 the project structure: BACK-END > node\_modules, package-lock.json, and package.json. The main editor displays the package.json file with the following content:

```
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**.

The screenshot shows the Visual Studio Code interface with the Explorer on the left showing the project structure: BACK-END > node\_modules, package-lock.json, package.json, and server.js. The main editor displays the server.js file with the following content:

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

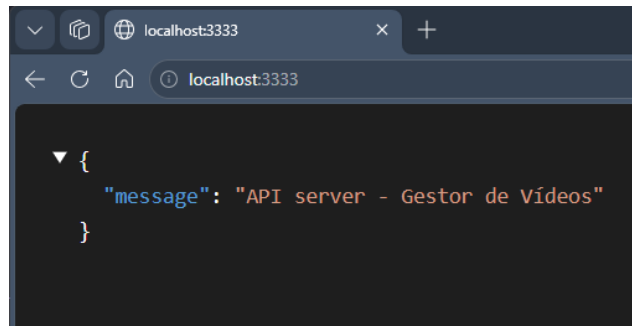
The terminal at the bottom shows the command `npm run dev` being executed, resulting in the following output:

```
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>

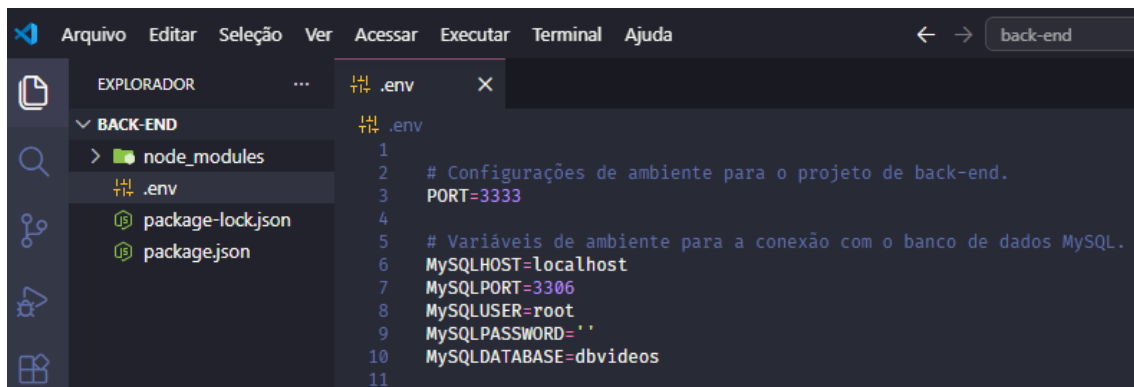


## Arquivo ENV

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.



## 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`.

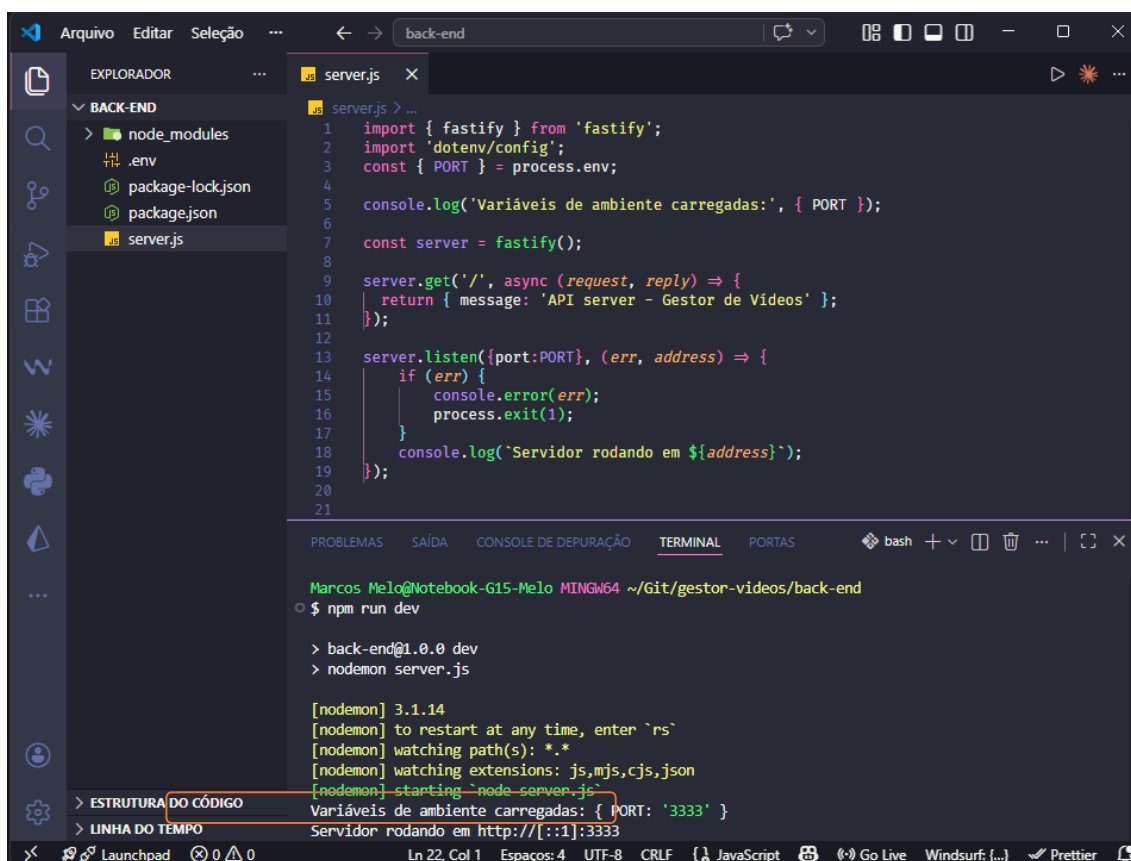
```
npm install dotenv -D
```

**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
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
```

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



```
Arquivo Editar Seleção ... back-end
EXPLORADOR
BACK-END
node_modules
.env
package-lock.json
package.json
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
PROBLEMAS SAÍDA CONSOLE DE DEPURACÃO TERMINAL PORTAS
bash
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`
Variáveis de ambiente carregadas: { PORT: '3333' }
Servidor rodando em http://[::1]:3333
Ln 22, Col 1 Espaços: 4 UTF-8 CRLF { } JavaScript Go Live Windsurf: ... Prettier
```

## Configurando o Git na pasta

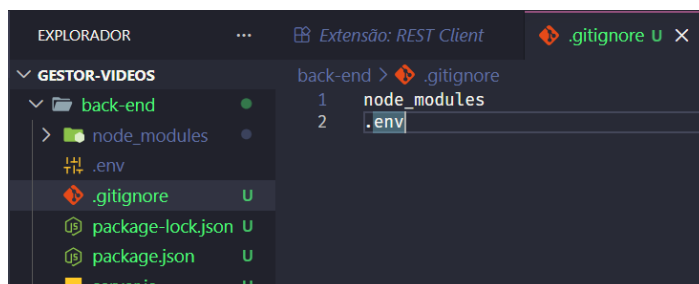
Se há a intenção de colocar este projeto no github, é preciso configurar o GIT na pasta backend para poder fazer os commits do projeto.

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.



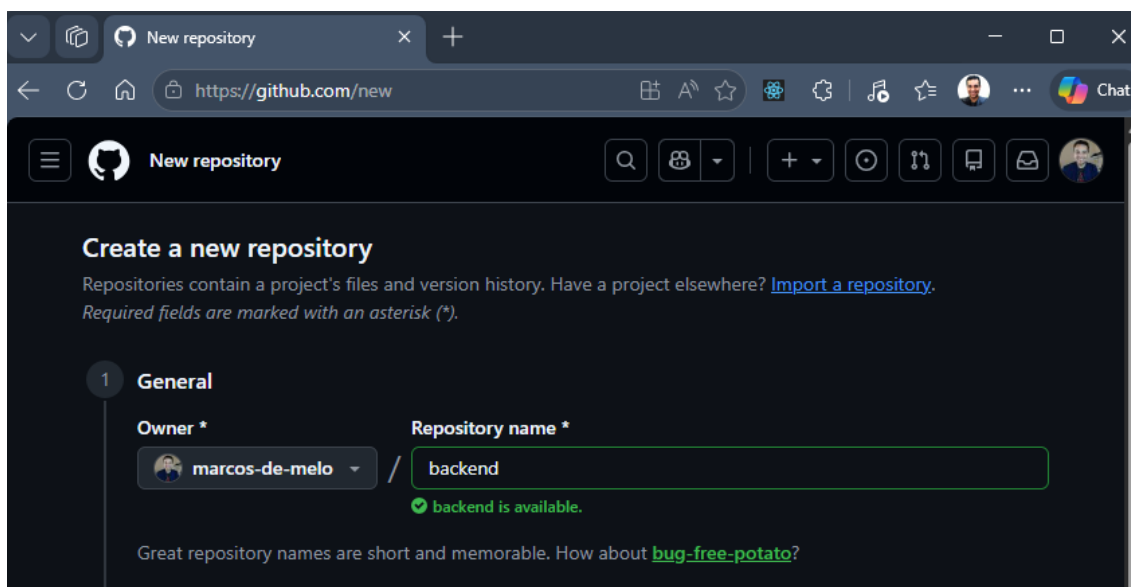
## Commit do projeto no Git e publicando no GitHub

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.

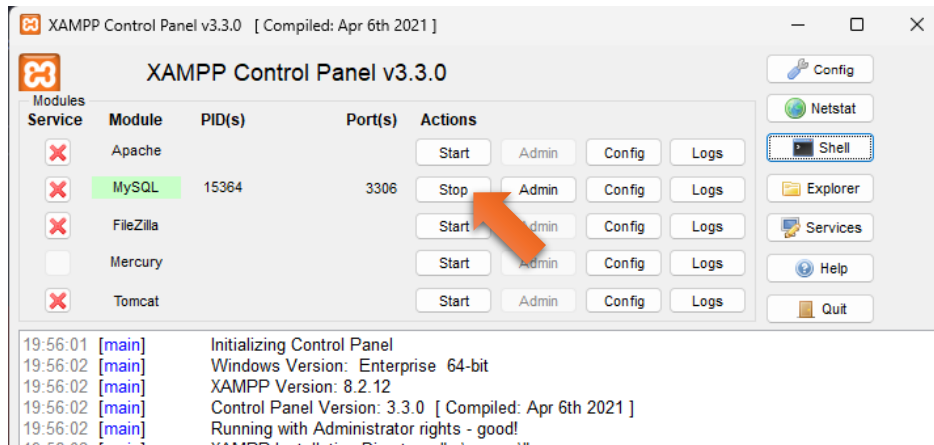
```
git add .  
git commit -m "primeiro commit - criação do servidor por fastify"  
git branch -M main  
git remote add origin https://github.com/sua-conta-aqui/backend.git  
git push -u origin main
```

## Persistência com o banco de dados

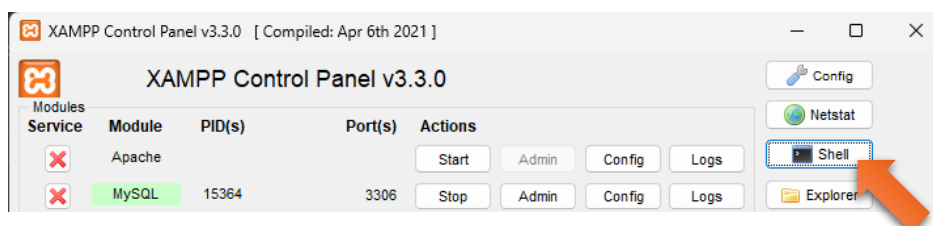
Chegou a hora de criar o banco de dados que será persistido na nossa API do backend, então antes que qualquer procedimento devemos iniciar o servidor do banco de dados que iremos criar acessar e persistir.

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 abriu, digite o código a seguir para acessar o servidor do MySQL.

```
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.

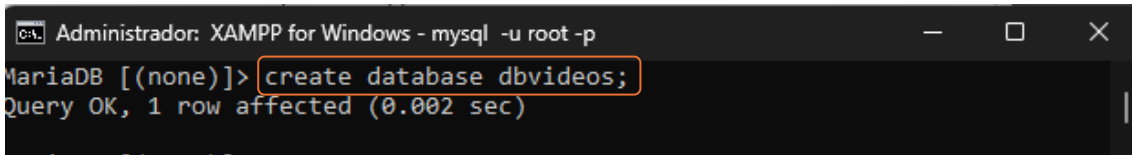
```
Administrador: XAMPP for Windows - mysql -u root -p
Setting environment for using XAMPP for Windows.
Marcos Melo@NOTEBOOK-G15-ME c:\xampp
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.4.32-MariaDB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



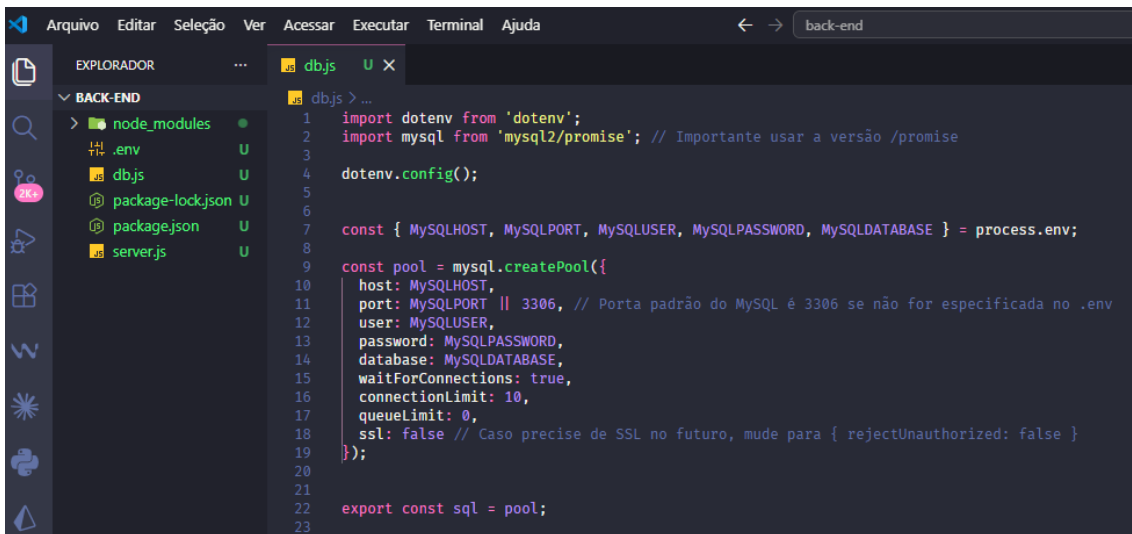
```
Administrador: XAMPP for Windows - mysql -u root -p
MariaDB [(none)]> create database dbvideos;
Query OK, 1 row affected (0.002 sec)
```

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

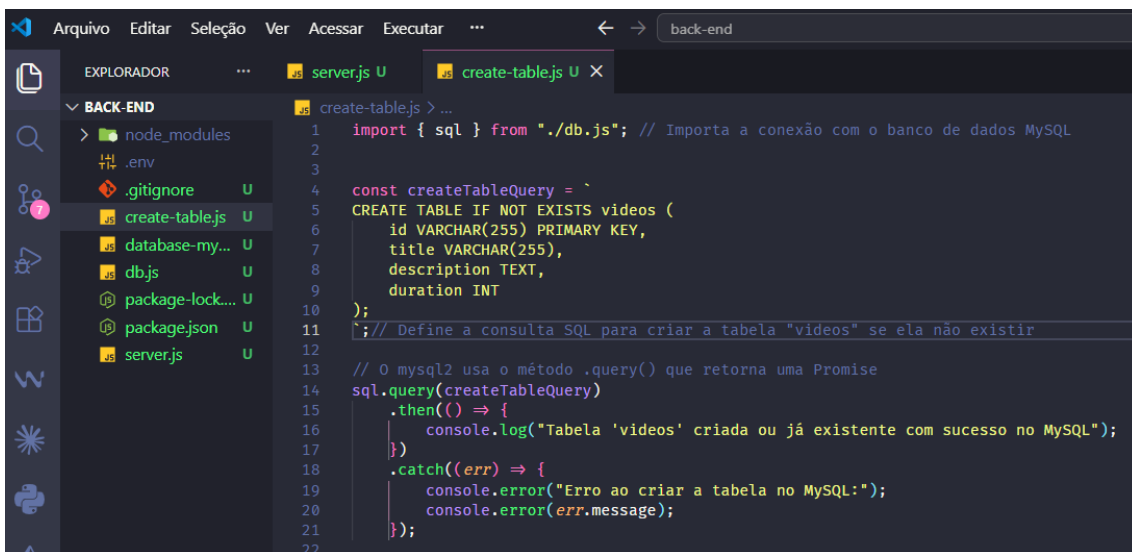
Crie na pasta raiz o arquivo **db.js**, este arquivo será usado para conectar com o servidor de banco de dados e consequentemente com o banco de dados que criamos.



```
Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda
back-end
EXPLORADOR
BACK-END
node_modules
.env
db.js
package-lock.json
package.json
server.js
db.js
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
10 const pool = mysql.createPool({
11   host: MySQLHOST,
12   port: MySQLPORT || 3306, // Porta padrão do MySQL é 3306 se não for especificada no .env
13   user: MySQLUSER,
14   password: MySQLPASSWORD,
15   database: MySQLDATABASE,
16   waitForConnections: true,
17   connectionLimit: 10,
18   queueLimit: 0,
19   ssl: false // Caso precise de SSL no futuro, mude para { rejectUnauthorized: false }
20 });
21
22 export const sql = pool;
```

## 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 banco **db.js**.



```
Arquivo Editar Seleção Ver Acessar Executar ...
back-end
EXPLORADOR
BACK-END
node_modules
.env
.gitignore
create-table.js
database-my...
db.js
package-lock...
package.json
server.js
create-table.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
```

```
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 [ '.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

```
1 import { randomUUID } from "node:crypto";
2 import { sql } from './db.js';
3
4 export class DatabaseMySQL {
5
6   // Listagem de videos, com opção de busca por titulo 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 }
```

```
23
24 // Criação de um novo vídeo, gerando um ID único usando randomUUID
    Windsurf: Refactor | Explain | X
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
    Windsurf: Refactor | Explain | X
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
    Windsurf: Refactor | Explain | X
46 ✓ async delete(id) {
47     await sql.execute('DELETE FROM videos WHERE id = ?', [id]);
48 }
49 }
```

Continua na próxima aula...