Web Fullstack na Prática:
Sistema de Cadastro de Vídeos

Sumário

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.

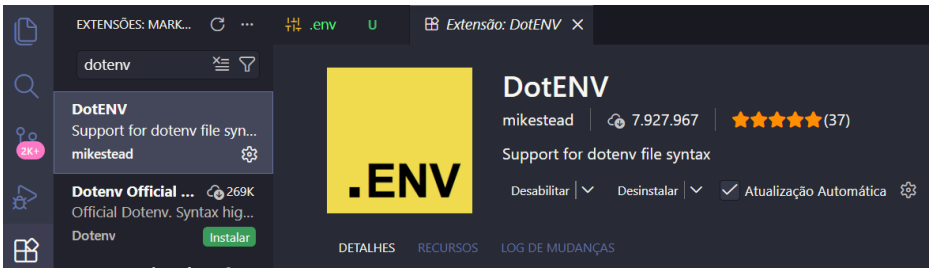
O significado de cada letra é:

- **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

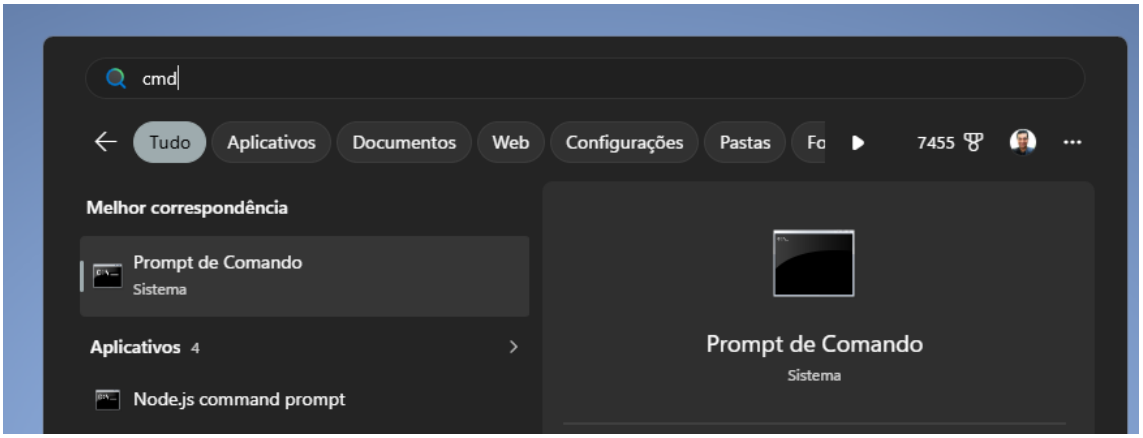
Vamos criar toda a parte de **backend**, logo após faremos a interface no **frontend** e em uma terceira etapa faremos a comunicação entre os dois.

Extensões do VSCode que iremos usar no backend



Organização das pastas do projeto

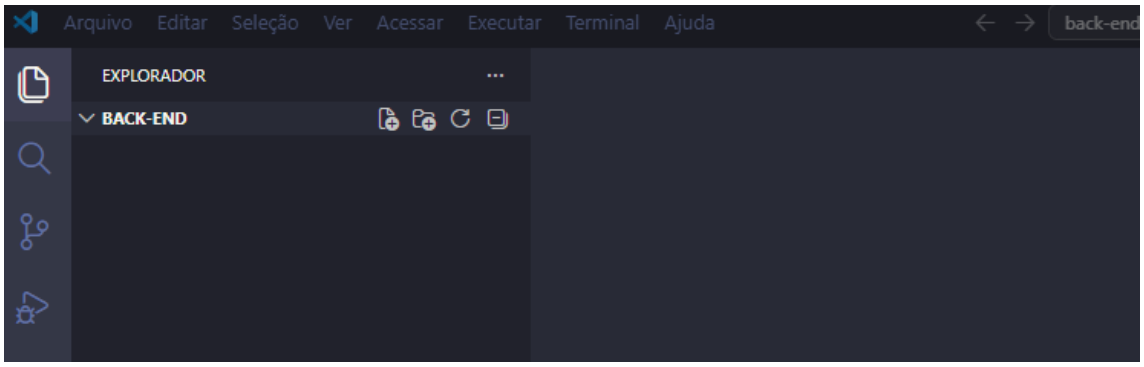
Vamos criar a estrutura de pastas do projeto a começar pela pasta raiz (principal), esta pasta se chamara “gestor-videos” e dentro dela será criado a pasta back-end, vamos usar o terminal **cmd** (Prompt de Comandos).



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

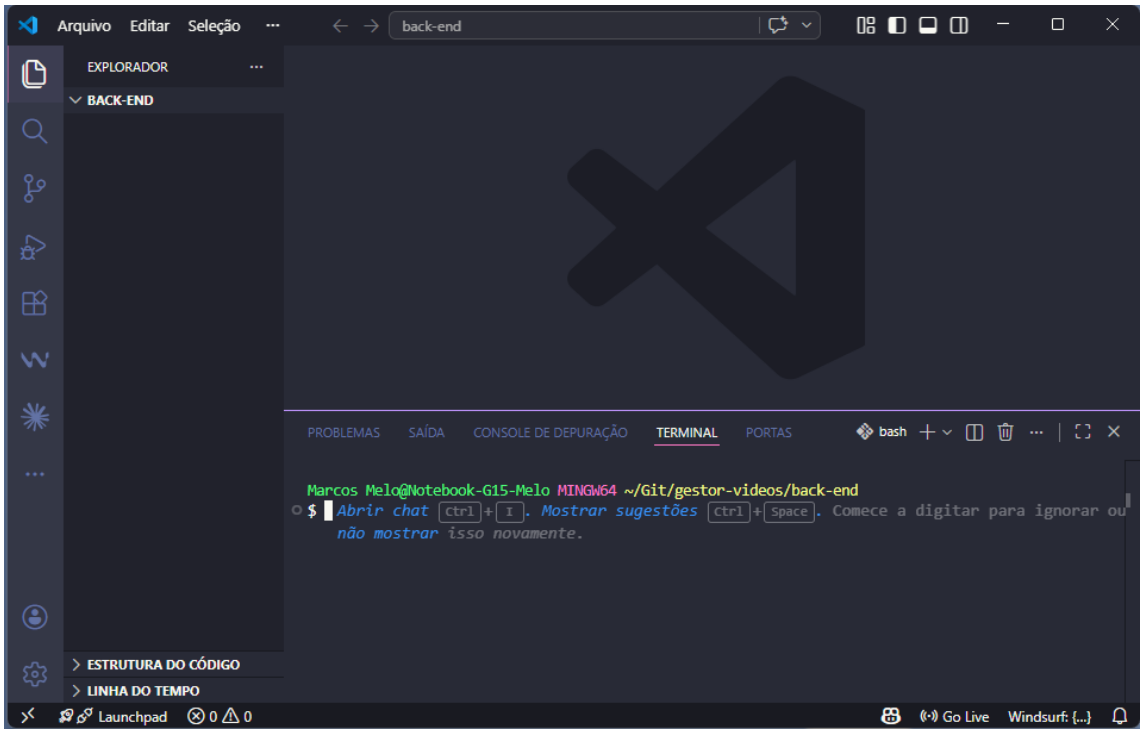
```
md gestor-videos
cd gestor-videos
md back-end
cd back-end
code .
```

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.

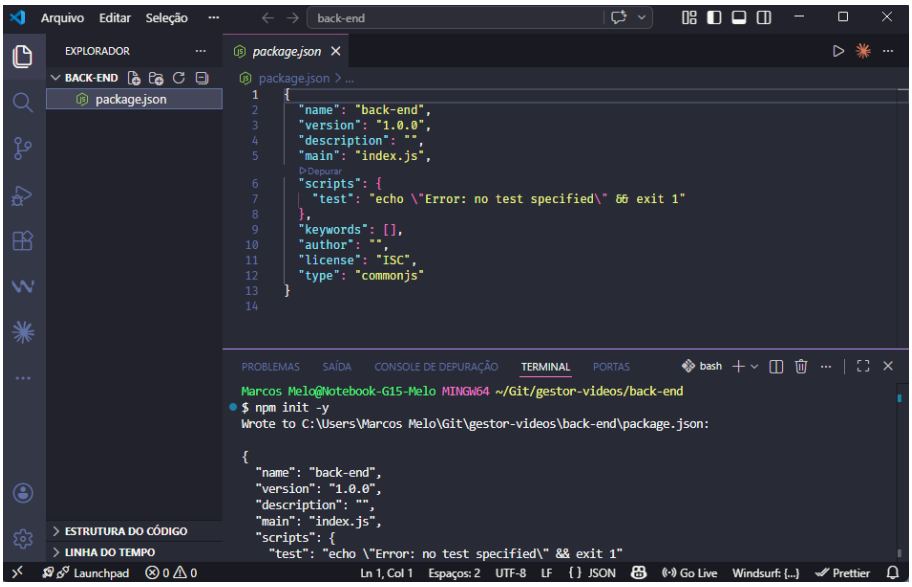
```
git init
```

Iniciando o gerenciador de pacotes do NodeJS na pasta raiz

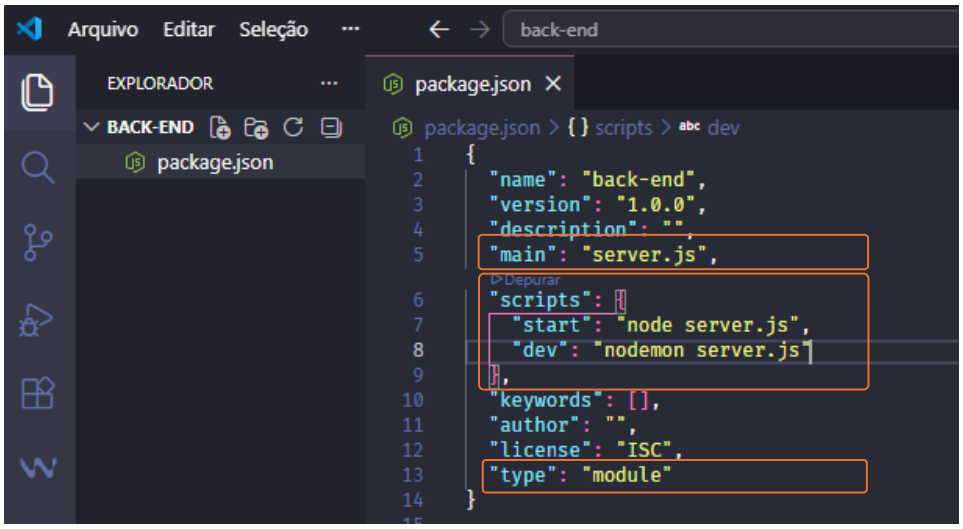
No terminal digite o comando abaixo para inicializar o gerenciador de pacotes NPM dentro da pasta raiz.

```
npm init -y
```

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



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



Instalando todas as bibliotecas

Instale a biblioteca nodemon

```
npm install nodemon -D
```

Instale a biblioteca dotenv

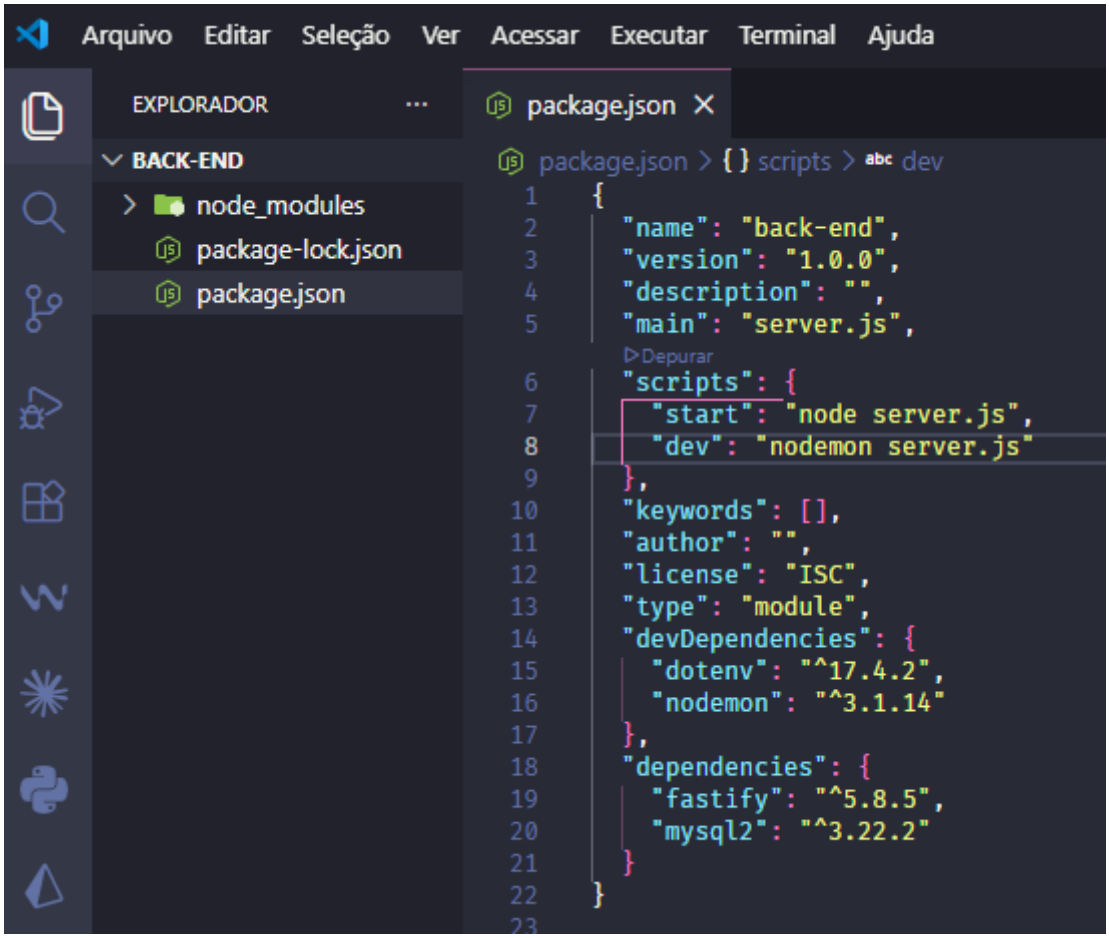
```
npm install dotenv -D
```

Instale a biblioteca do Fastify

```
npm install fastify
```

Instale a biblioteca de conexão com o banco de dados MySQL

```
npm install mysql2
```



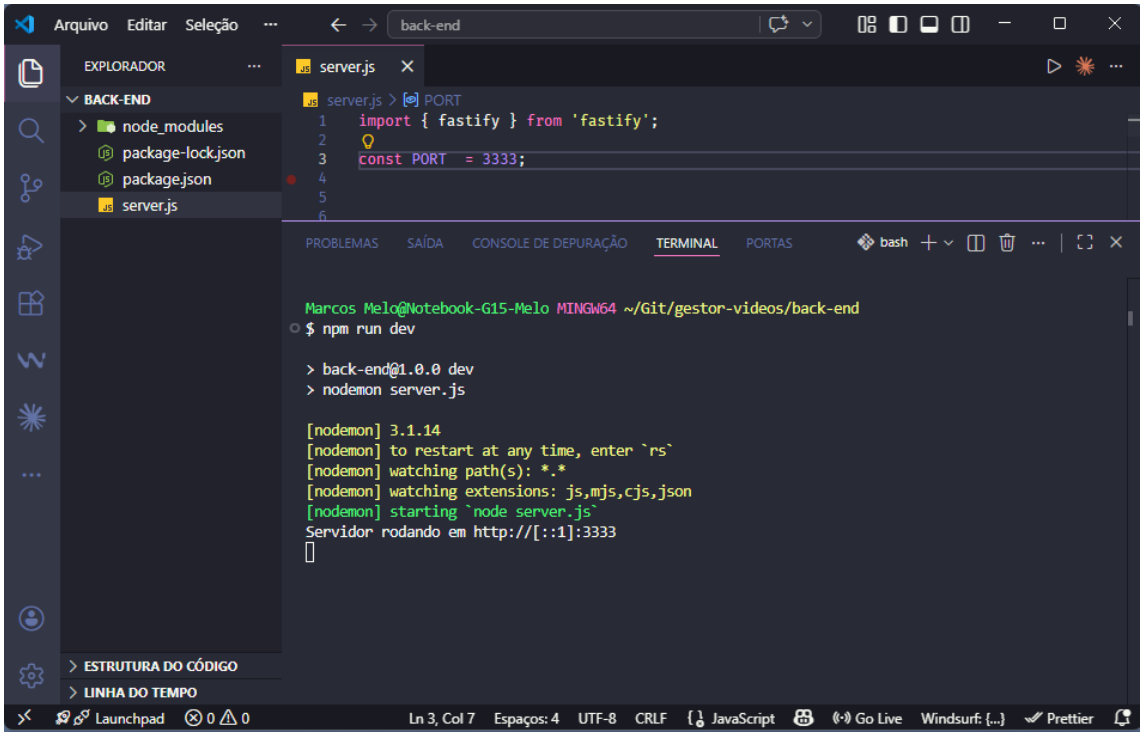
Configurando o servidor com Fastify

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;

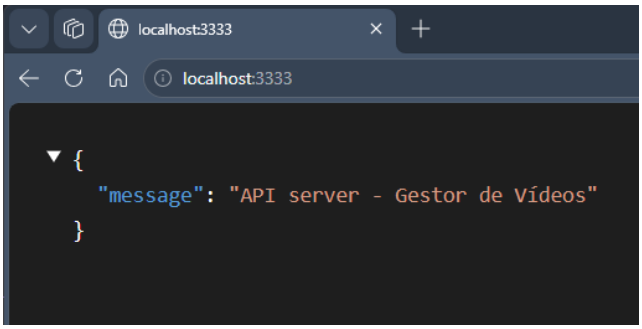
Após digitar o código teste o servidor executando o código a seguir no terminal;

```
npm run dev
```



Acesse no navegador pelos endereços

[::1]:3333 ou http://localhost:3333

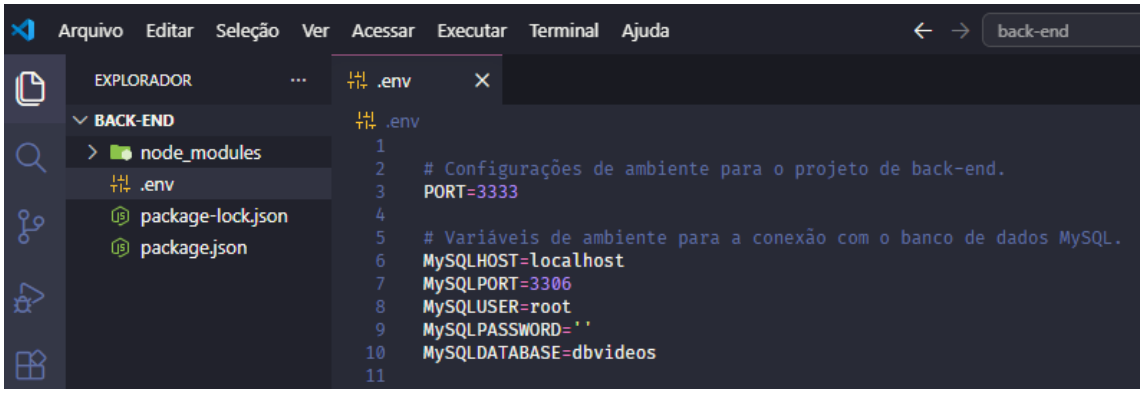


Arquivo ENV

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.

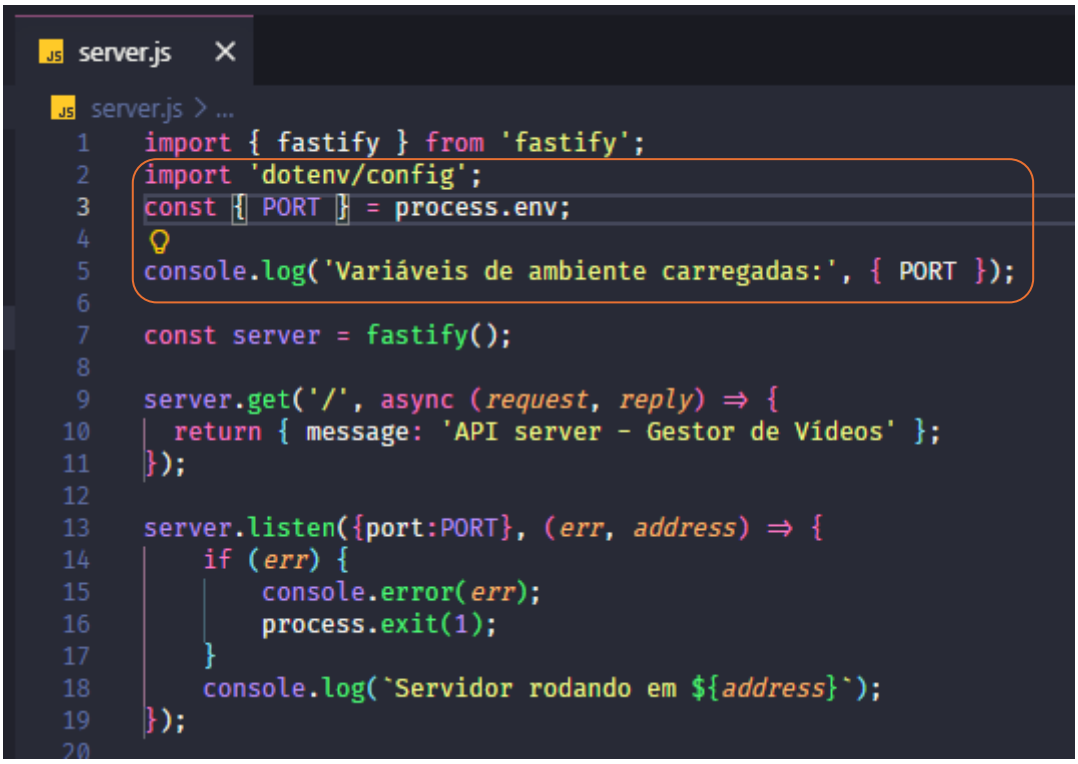
A forma que iremos acessar as informações deste arquivo é através da **Biblioteca dotenv:** É a solução mais comum para versões anteriores do Node.js.

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

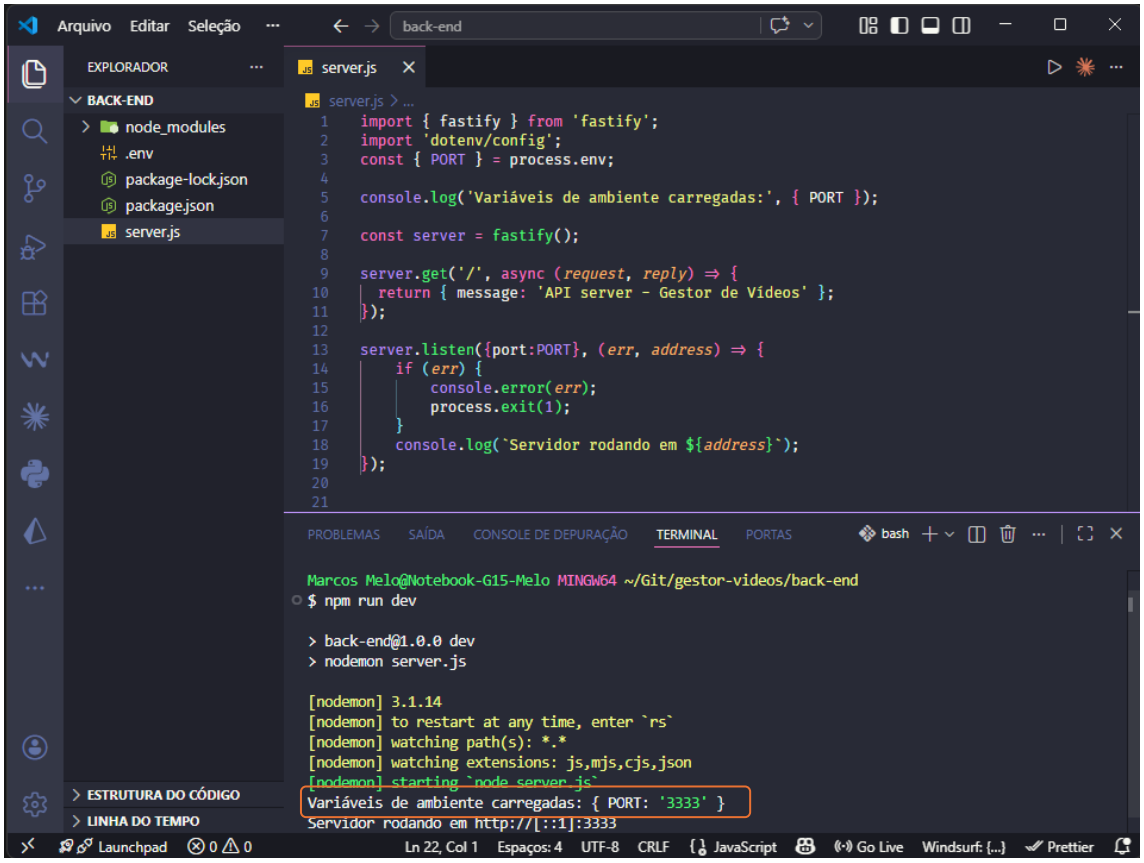
```
npm install dotenv -D
```

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;



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



Configurando o Git na pasta

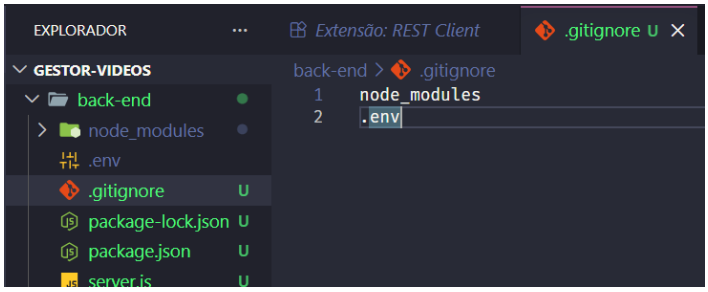
Se há a intensão de colocar este projeto no github, é preciso configurar o GIT na pasta backend para poder fazer os commits do projeto.

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

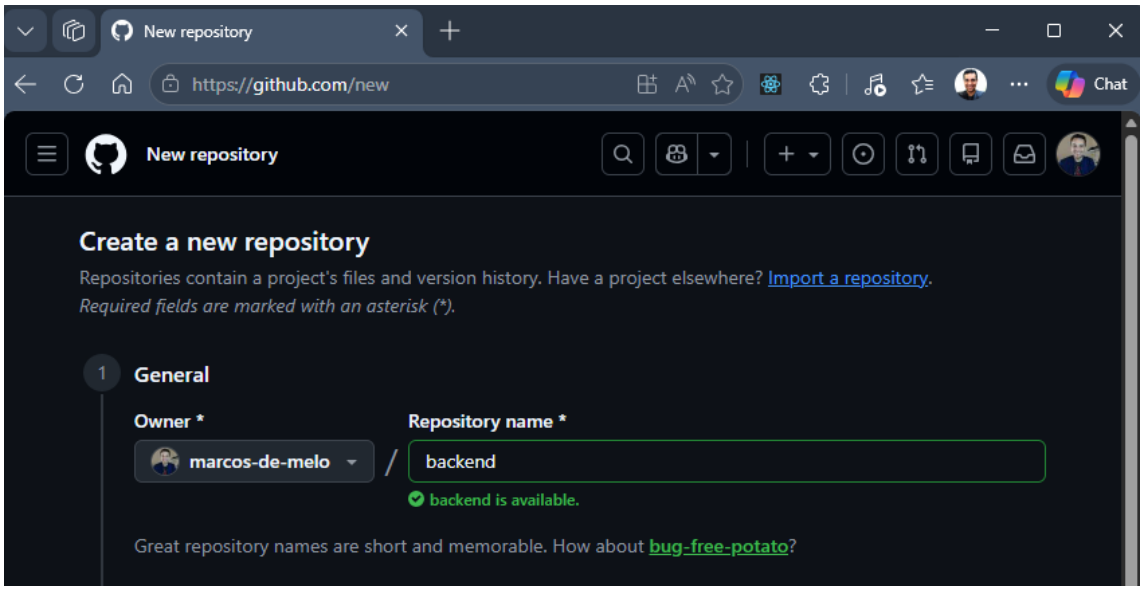
Pausa no projeto para publicar no GitHub, vamos fazer o commit do projeto até o momento e subir o projeto para o GitHub, assim poderemos dar continuidade nas próximas aulas.

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.

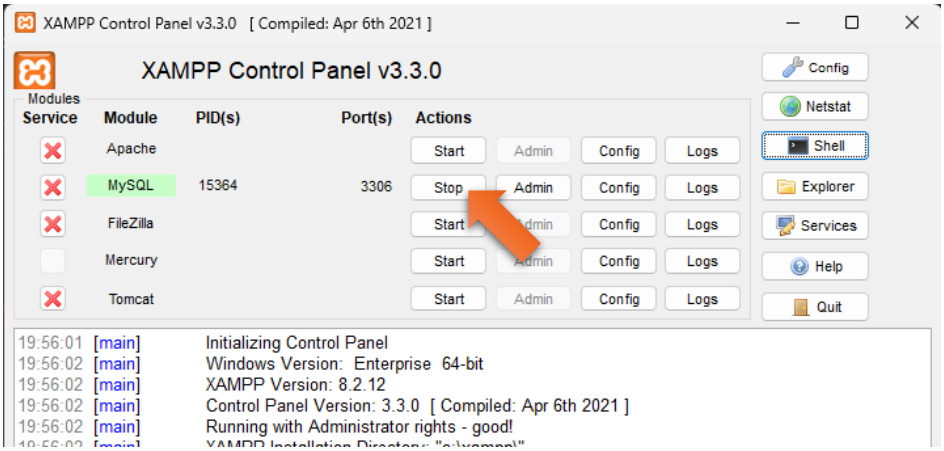
```
git add .
git commit -m "primeiro commit – criação do servidor por fastify"
git branch -M main
git remote add origin https://github.com/sua-conta-aqui/backend.git
git push -u origin main
```

Persistência com o banco de dados

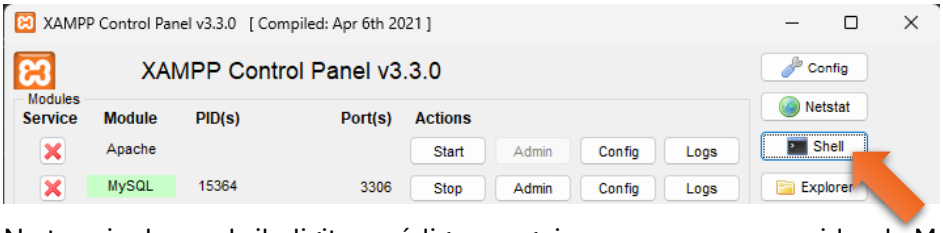
Chegou a hora de criar o banco de dados que será persistido na nossa API do backend, então antes que qualquer procedimento devemos iniciar o servidor do banco de dados que iremos criar acessar e persistir.

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.

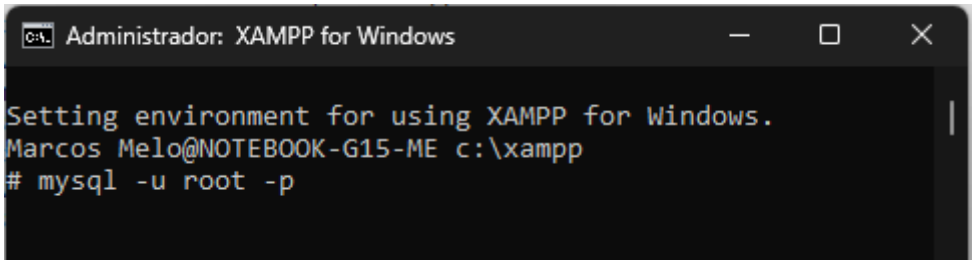
Abrindo o XAMPP Control pressione o botão Start do MySQL para iniciar o servidor.



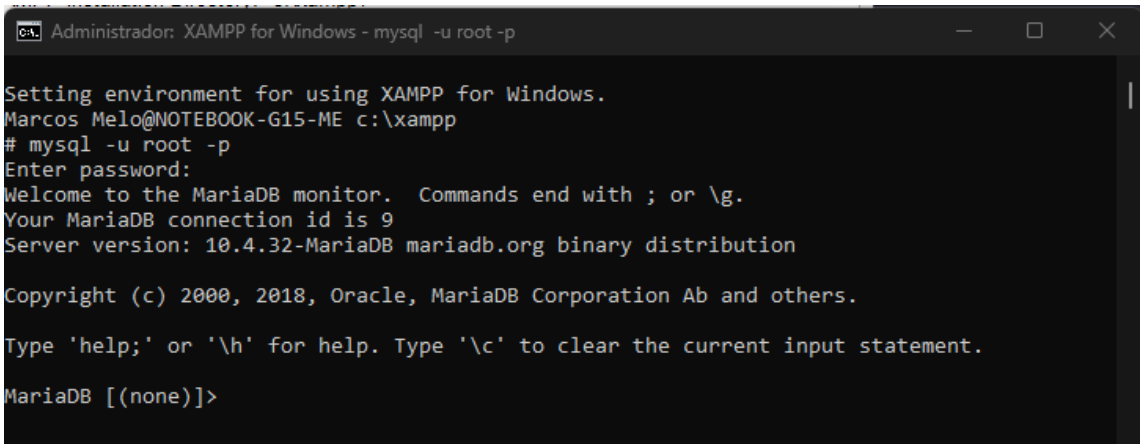
Clique no botão Shell para abrir o terminal do XAMPP para acessar o MySQL onde vamos criar o banco de dados que vamos usar.



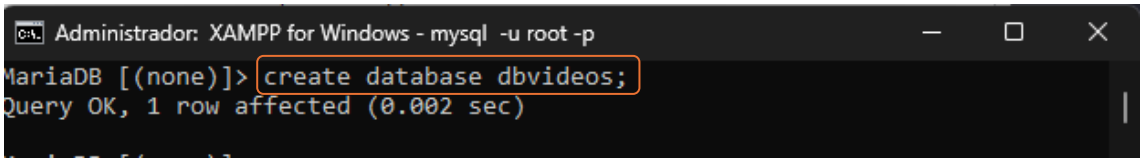
No terminal que abril, digite o código a seguir para acessar o servidor do MySQL.



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



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

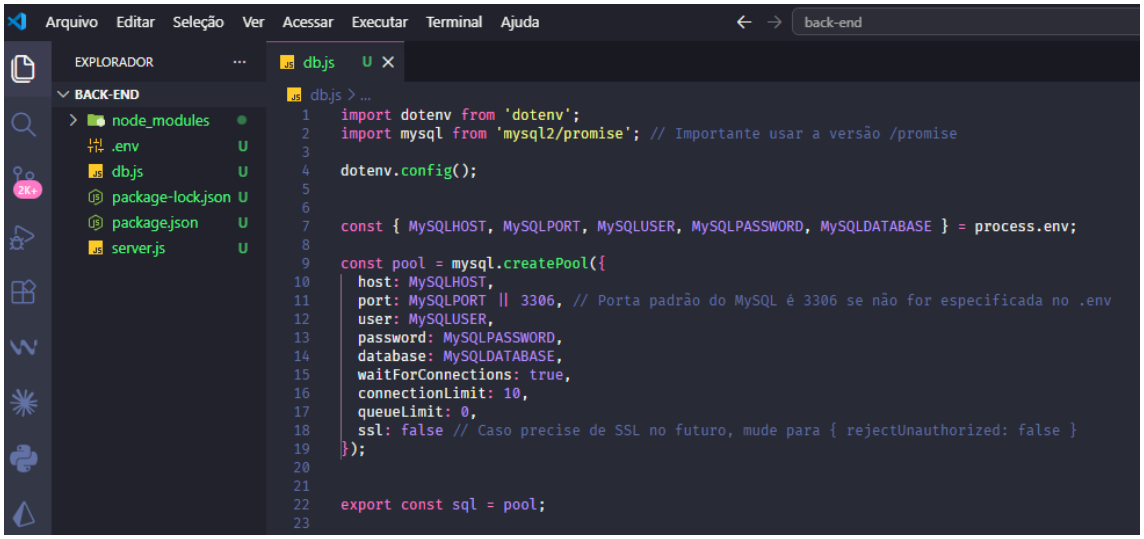


Se nenhum erro aparecer o banco deve ter sido criado, mas para verificar se foi mesmo criado digite no terminal o comando a seguir;

```
show databases;
```

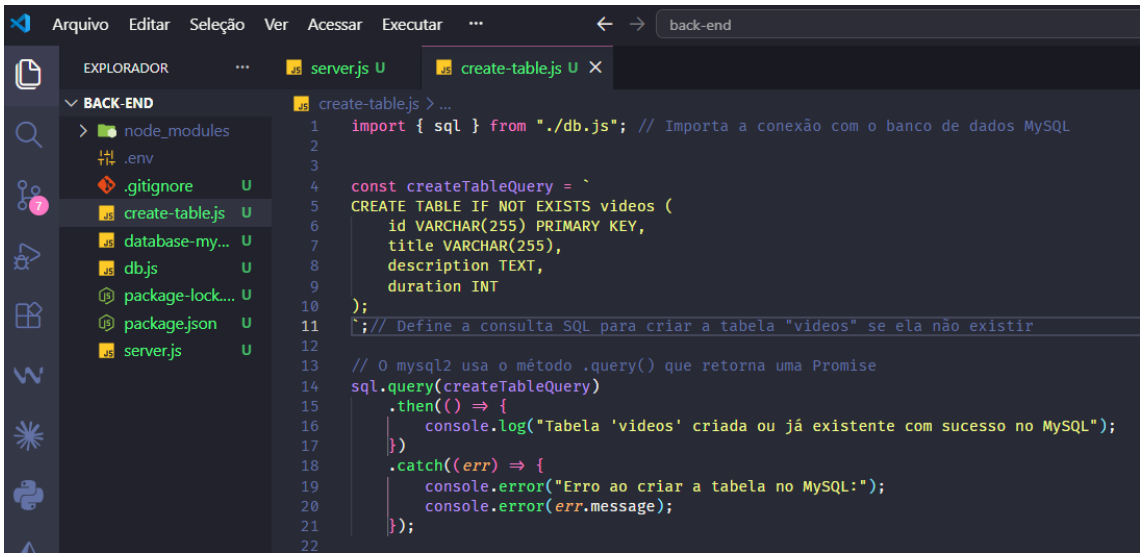
Criando o arquivo de conexão com o banco de dados

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.



Criando a tabela vídeos

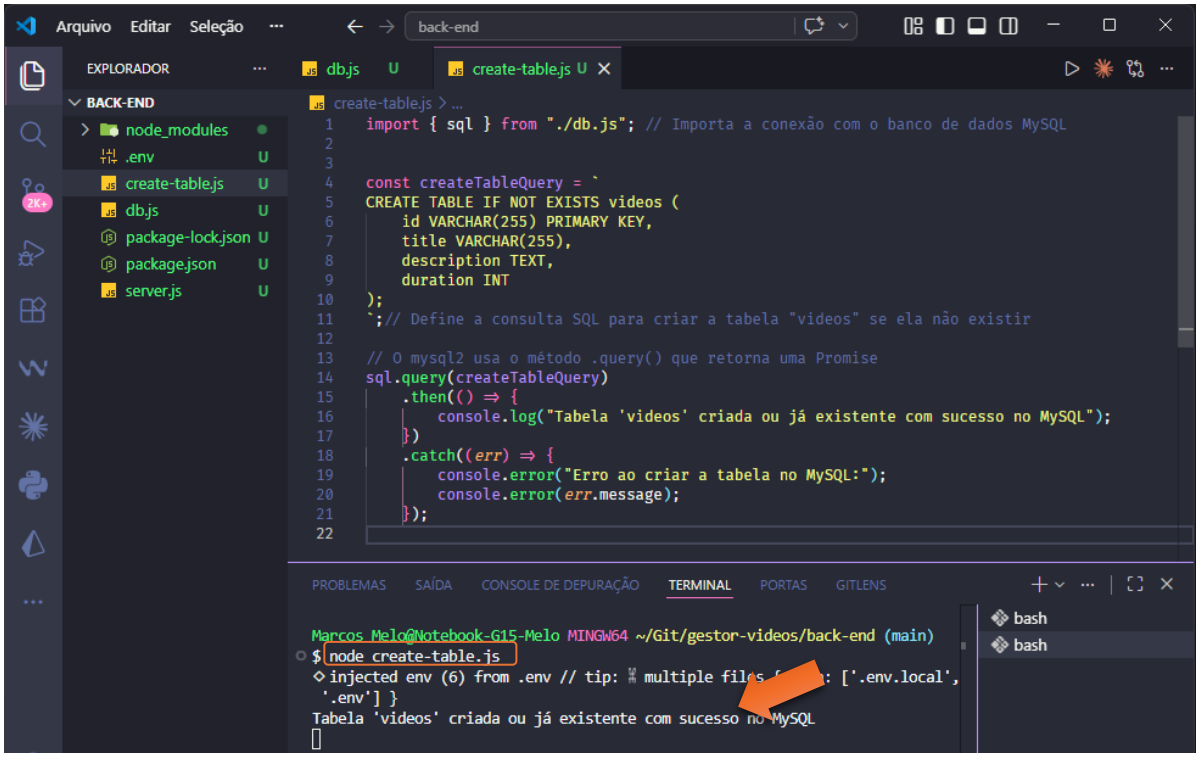
O arquivo que será criado agora é só para criar no banco de dados a tabela vídeos e será usado somente uma vez, poderíamos ter criado a tabela já anteriormente quando criamos o banco, mas deixamos para criar agora, para demonstrar como acessar o banco de dados através do arquivo de conexão **db.js**.



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.



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

back-end > database-mysql.js > ...

```
import { randomUUID } from "node:crypto";
import { sql } from "./db.js";

Windsurf: Refactor | Explain
export class DatabaseMYSQL {

// Listagem de vídeos, com opção de busca por título usando o operador LIKE
    Windsurf: Refactor | Explain |
    async list(search) {
        let videos;

        if (search) {
            // No mysql2, usamos o caractere "?" como placeholder para evitar SQL Injection
            // O resultado vem como [videos, fields], por isso usamos a desestruturação [videos]
            [videos] = await sql.execute(
                'SELECT * FROM videos WHERE title LIKE ?',
                [`%${search}%`]
            );
        } else {
            [videos] = await sql.execute('SELECT * FROM videos');
        }

        return videos;
    }

// Criação de um novo vídeo, gerando um ID único usando randomUUID
    Windsurf: Refactor | Explain |
    async create(video) {
        const videoId = randomUUID();
        const { title, description, duration } = video;

        // No mysql2, passamos os valores em um array como segundo argumento
        await sql.execute(
            'INSERT INTO videos (id, title, description, duration) VALUES (?, ?, ?, ?)',
            [videoId, title, description, duration]
        );
    }

// Atualização de um vídeo específico usando o ID
    Windsurf: Refactor | Explain |
    async update(id, video) {
        const { title, description, duration } = video;
        await sql.execute(
            'UPDATE videos SET title = ?, description = ?, duration = ? WHERE id = ?',
            [title, description, duration, id]
        );
    }

// Exclusão de um vídeo específico usando o ID
    Windsurf: Refactor | Explain |
    async delete(id) {
        await sql.execute('DELETE FROM videos WHERE id = ?', [id]);
    }
}
```

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 U

server.js > server.put("/videos/:id") callback

```
import { fastify } from 'fastify';
import { DatabaseMYSQL } from './database-mysql.js';
import 'dotenv/config';
const { PORT } = process.env;

console.log('Variáveis de ambiente carregadas:', { PORT });

const server = fastify();

server.get('/', async (request, reply) => {
  return { message: 'API server - Gestor de Vídeos' };
});



// Criando uma instância da classe DatabaseMYSQL para
// interagir com o banco de dados
const database = new DatabaseMYSQL();


// Rota para criar um novo vídeo, recebendo os dados no corpo
// da requisição e usando o método create do database
server.post("/videos", async (request, reply) => {
    const { title, description, duration } = request.body;
    await database.create({
        title,
        description,
        duration
    });
    console.log(await database.list());
    return reply.status(201).send();
})

// Rota para listar os vídeos, com opção de busca por título
// usando query string e o método list do database
server.get("/videos", async (request) => {
    const search = request.query.search;
    console.log(search);
    const videos = await database.list(search);
    return videos
})

// Rota para atualizar um vídeo específico, recebendo o
// ID na URL e os dados no corpo da requisição, usando o
// método update do database
server.put("/videos/:id", async (request,reply) => {

    const videoId = request.params.id;
    const { title, description, duration } = request.body;

    const video = await database.update(videoId, {
        title,
        description,
        duration,
    });
```

```
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 **rotes.http,** neste arquivo vamos colocar todas as requisições http das rotas que criamos.

rotes.http U

rotes.http > ...

```
Send Request
POST http://localhost:3333/videos
Content-Type: application/json

{
    "title": "Video teste",
    "description": "Este é o primeiro video de Node.js com MySQL",
    "duration": 900
}
###

Send Request
GET http://localhost:3333/videos
# GET http://localhost:3333/videos

###

Send Request
PUT http://localhost:3333/videos/id-do-video-a-ser-atualizado
Content-Type: application/json

{
    "title": "Auterei este video",
    "description": "descrição atualizada",
    "duration": 345
}

###

Send Request
DELETE  http://localhost:3333/videos/id-do-video-a-ser-deletado

```