REVISÃO PM2025

🚀 Instalar e configurar o WSL

✅ 1.1. Ativar o WSL no Windows

Abra o **PowerShell como administrador** e execute:

```
wsl --install
```

✅ 1.2. Verificar se está usando WSL 2

Execute no PowerShell:

```
wsl --list --verbose
```

Se a coluna "VERSION" mostrar `2`, tá tudo certo. Se não:

```
wsl --set-version Ubuntu-22.04 2
```

✅ 1.3. Acessar o terminal Linux (Ubuntu)

No menu iniciar, procure por **Ubuntu** e abra.

Depois, atualize os pacotes:

```
sudo apt update && sudo apt upgrade -y
```

Instalar o Node.js no Linux (WSL)

A forma **mais segura e recomendada** é com o `nvm`.

📥 1.1 Instalar o NVM:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

Depois, rode:

```
export NVM_DIR="$HOME/.nvm"
source "$NVM_DIR/nvm.sh"
```

🔧 1.2 Instalar o Node mais recente:

```
nvm install node DEPOIS nvm use node
nvm alias default node - VERIFIQUE SE FUNCIONA - node -v && npm -v
```

🐳 Instalar Docker no WSL com `docker.io`

✅ 1.1 Atualizar pacotes e instalar o Docker (com Compose e Buildx)

No terminal do Ubuntu (WSL), execute:

```
sudo apt update
sudo apt install docker.io
sudo apt install docker-compose-v2 docker-buildx -y
```

✅ 1.2. Ativar e configurar permissões

Depois de instalado:

1. **Habilite o serviço do Docker no WSL:**

   ```
   sudo service docker start
   ```

2. **Adicione seu usuário ao grupo `docker` para evitar usar `sudo`:**

   ```
   sudo usermod -aG docker $USER
   ```

3. **Reinicie o terminal** ou execute:

   ```
   newgrp docker DEPOIS TESTE docker --version DEPOIS docker compose version
   ```

📌 Etapa 1: Criar uma página de cadastro com React (frontend)

Objetivo: Página simples com campos como nome, e-mail e senha, e um botão para enviar.

✅ Estrutura de Pastas Inicial

```
projeto-cadastro/
├── frontend/
├── backend/
```

✅ Passo 1.1 – Criar o projeto React

```
npm create vite
```

✅ Passo 1.2 – Criar a página de cadastro `App.tsx`:

```
// frontend/src/App.tsx
import { useEffect, useState } from "react";
import {
  buscarCadastros,
  cadastrar,
  atualizarCadastro,
  excluirCadastro,
  type Cadastro,
} from "./services/cadastroService";

function App() {
  const [cadastros, setCadastros] = useState<Cadastro[]>([]);
  const [form, setForm] = useState({ nome: "", email: "", senha: "" });
  const [editId, setEditId] = useState<number | null>(null);

  const fetchCadastros = async () => {
    try {
      const res = await buscarCadastros();
      setCadastros(res.data);
    } catch (error) {
      console.error("Erro ao buscar cadastros:", error);
```

```
    }
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      if (editId) {
        await atualizarCadastro(editId, form);
        setEditId(null);
      } else {
        await cadastrar(form);
      }
      setForm({ nome: "", email: "", senha: "" });
      fetchCadastros();
    } catch (error) {
      console.error("Erro ao enviar formulário:", error);
    }
  };

  const handleEdit = (cadastro: Cadastro) => {
    setForm({ nome: cadastro.nome, email: cadastro.email, senha: cadastro.
senha });
    setEditId(cadastro.id);
  };

  const handleDelete = async (id: number) => {
    try {
      await excluirCadastro(id);
      fetchCadastros();
    } catch (error) {
      console.error("Erro ao deletar cadastro:", error);
    }
  };

  useEffect(() => {
    fetchCadastros();
  }, []);
```

```
  return (
    <div style={{ padding: 20 }}>
      <h1>Cadastro</h1>
      <form onSubmit={handleSubmit}>
        <input
          placeholder="Nome"
          value={form.nome}
          onChange={(e) ⇒ setForm({ ...form, nome: e.target.value })}
          required
        />
        <input
          placeholder="Email"
          type="email"
          value={form.email}
          onChange={(e) ⇒ setForm({ ...form, email: e.target.value })}
          required
        />
        <input
          placeholder="Senha"
          type="password"
          value={form.senha}
          onChange={(e) ⇒ setForm({ ...form, senha: e.target.value })}
          required
        />
        <button type="submit">{editId ? "Atualizar" : "Cadastrar"}</button>
      </form>

      <hr />

      <h2>Lista</h2>
      <ul>
        {cadastros.map((c) ⇒ (
          <li key={c.id}>
            {c.nome} - {c.email}{" "}
            <button onClick={() ⇒ handleEdit(c)}>Editar</button>
            <button onClick={() ⇒ handleDelete(c.id)}>Excluir</button>
          </li>
        ))}
```

```
      </ul>
    </div>
  );
}

export default App;
```

✅ Passo 1.2 – Criar a página de cadastro `App.tsx`:

```
// src/services/api.ts
import { api } from "../api";

export type Cadastro = {
  id: number;
  nome: string;
  email: string;
  senha: string;
};

export const buscarCadastros = () => api.get<Cadastro[]>("/cadastros");

export const cadastrar = (dados: Omit<Cadastro, "id">) =>
  api.post("/cadastrar", dados);

export const atualizarCadastro = (id: number, dados: Omit<Cadastro, "id">)
=>
  api.put(`/cadastros/${id}`, dados);

export const excluirCadastro = (id: number) =>
  api.delete(`/cadastros/${id}`);
```

🔧Configurar o serviço de API

instalar o axios:

```
npm install axios
```

```
npm install -D @types/axios
```

📁 src/api.ts

```
import axios from 'axios';

export const api = axios.create({
  baseURL: 'http://localhost:3001',
});
```

✅ Etapa 2 – Backend com Express: Criar o endpoint de cadastro

📁 Estrutura de diretório

```
projeto-cadastro/
├── frontend/
└── backend/
```

✅ Passo 2.1 – Iniciar projeto Node no backend

```
cd ../backend
npm init -y
npm install express cors
npm install cors
npm install --save-dev nodemon
```

✅ Passo 2.2 – Criar arquivos principais

1️⃣ server.js

```
const express = require("express");
const cors = require("cors");

const app = express();
```

```
const port = 3001;
app.use(cors({
  origin: "http://localhost:5173"
}));
app.use(express.json());

let cadastros = [];

app.post("/cadastrar", (req, res) => {
  const { nome, email, senha } = req.body;
  const novoCadastro = { id: Date.now(), nome, email, senha };
  cadastros.push(novoCadastro);
  res.status(201).json({ mensagem: "Cadastro realizado", cadastro: novoCa
dastro });
});

app.get("/cadastros", (req, res) => {
  res.json(cadastros);
});

app.put("/cadastros/:id", (req, res) => {
  const { id } = req.params;
  const { nome, email, senha } = req.body;

  const index = cadastros.findIndex(c => c.id == id);
  if (index === -1) return res.status(404).json({ erro: "Cadastro não encontr
ado" });

  cadastros[index] = { ...cadastros[index], nome, email, senha };
  res.json({ mensagem: "Cadastro atualizado", cadastro: cadastros[index]
});
});

app.delete("/cadastros/:id", (req, res) => {
  const { id } = req.params;
  cadastros = cadastros.filter(c => c.id != id);
  res.json({ mensagem: "Cadastro removido" });
});
```

```
app.listen(port, () => {
  console.log(`Servidor rodando na porta ${port}`);
});
```

```
// controllers/cadastroController.js
const db = require("../data/db");

let cadastros = db.cadastros;

function listarCadastros(req, res) {
  res.json(cadastros);
}

function cadastrar(req, res) {
  const { nome, email, senha } = req.body;
  const novoCadastro = { id: Date.now(), nome, email, senha };
  cadastros.push(novoCadastro);
  res.status(201).json({ mensagem: "Cadastro realizado", cadastro: novoCa
dastro });
}

function atualizar(req, res) {
  const { id } = req.params;
  const { nome, email, senha } = req.body;
  const index = cadastros.findIndex(c => c.id == id);
  if (index === -1) return res.status(404).json({ erro: "Cadastro não encontr
ado" });
  cadastros[index] = { ...cadastros[index], nome, email, senha };
  res.json({ mensagem: "Cadastro atualizado", cadastro: cadastros[index]
});
}

function remover(req, res) {
  const { id } = req.params;
  cadastros = cadastros.filter(c => c.id != id);
  db.cadastros = cadastros;
  res.json({ mensagem: "Cadastro removido" });
```

```
}

module.exports = {
  listarCadastros,
  cadastrar,
  atualizar,
  remover,
};
```

```
// data/db.js
module.exports = {
  cadastros: [], // isso vai virar banco depois
};
```

```
// routes/cadastroRoutes.js
const express = require("express");
const router = express.Router();
const controller = require("../controllers/cadastroController");

router.get("/cadastros", controller.listarCadastros);
router.post("/cadastrar", controller.cadastrar);
router.put("/cadastros/:id", controller.atualizar);
router.delete("/cadastros/:id", controller.remover);

module.exports = router;
```

2️⃣ `package.json` – adicionar script

Adiciona isso na seção `"scripts"` do `package.json`:

```
"scripts": {
  "dev": "nodemon server.js"
}
```

✅ Passo 2.3 – Rodar o servidor

```
npm run dev - SAIDA ESPERADA: Servidor rodando em http://localhost:300
1
```

✅ Passo 2.4 – Testar com frontend

Volta no navegador, preenche os campos e clica em **Cadastrar**.

Se aparecer um `alert` com `{ mensagem: "Cadastro recebido com sucesso!" }`, tá tudo 🔥.

---

1. **GET `/cadastros` — Listar todos os cadastros -** URL: `http://localhost:3001/cadastros`

1. POST `/cadastrar` — Criar um cadastro - URL: `http://localhost:3001/cadastrar`
   - Body (JSON):

```
"nome": "Muri",
"email": "muri@example.com",
"senha": "123456"
```

Não esqueça de testar no frontend !!

Subindo container

---

1. Criar um `docker-compose.yml` na raiz do projeto (uma pasta pai que tenha backend e frontend)

Exemplo básico:

```
version: '3.8'

services:
  mongodb:
    image: mongo:6.0
    container_name: mongodb
    restart: always
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db

  postgres:
```

```
  image: postgres:15
  container_name: postgres
  restart: always
  environment:
    POSTGRES_USER: myuser
    POSTGRES_PASSWORD: mypassword
    POSTGRES_DB: mydb
  ports:
    - "5432:5432"
  volumes:
    - pgdata:/var/lib/postgresql/data

backend:
  build:
    context: ./backend
  container_name: backend
  restart: always
  ports:
    - "3001:3001"
  depends_on:
    - mongodb
    - postgres
  environment:
    - MONGO_URI=mongodb://mongodb:27017/mydb
    - POSTGRES_USER=myuser
    - POSTGRES_PASSWORD=mypassword
    - POSTGRES_DB=mydb
    - POSTGRES_HOST=postgres
  volumes:
    - ./backend:/usr/src/app
  command: npm run dev

frontend:
  build:
    context: ./frontend
  container_name: frontend
  restart: always
  ports:
```

```
    - "5173:5173"
  depends_on:
    - backend
  volumes:
    - ./frontend:/usr/src/app
  command: npm run dev

volumes:
  mongo_data:
  pgdata:
```

2. Dockerfile para backend (`backend/Dockerfile`)

```
FROM node:18

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3001

CMD ["npm", "run", "dev"]
```

3. Dockerfile para frontend (`frontend/Dockerfile`)

```
FROM node:18

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install
```

```
COPY . .

EXPOSE 5173

CMD ["npm", "run", "dev"]
```

5. Rodar o docker-compose

No terminal (dentro da pasta raiz do projeto, onde está o `docker-compose.yml`), rode:

```
docker-compose up --build
```

📦 Etapa 1 – Instalar e configurar o `pg`

1. Instalar dependência:

```
cd backend
npm install pg
```

⚙️ Etapa 2 – Criar um módulo de conexão com PostgreSQL

Cria um arquivo `db.js` na raiz do `backend`:

```
const { Pool } = require('pg');

const pool = new Pool({
  user: process.env.POSTGRES_USER || 'myuser',
  host: process.env.POSTGRES_HOST || 'localhost',
  database: process.env.POSTGRES_DB || 'mydb',
  password: process.env.POSTGRES_PASSWORD || 'mypassword',
  port: 5432,
```

```
});

module.exports = pool;
```

> O docker-compose.yml já fornece essas variáveis: myuser, mypassword, mydb, host postgres.

🧪 Etapa 3 – Criar a tabela no banco

Você pode usar o **PGAdmin** ou rodar esse comando manualmente (via terminal com `psql`, ou conectando via container):

```
CREATE TABLE IF NOT EXISTS cadastros (
  id SERIAL PRIMARY KEY,
  nome TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  senha TEXT NOT NULL
);
```

Se quiser rodar direto via terminal dentro do container PostgreSQL:

```
docker exec -it postgres psql -U myuser -d mydb
```

E lá dentro do prompt do `psql`, cole o `CREATE TABLE`.

🔁 Etapa 4 – Atualizar suas rotas no backend

🧠 Atualização do `cadastroController.js` (sem `app` aqui!)

```
js
CopiarEditar
// controllers/cadastroController.js
const pool = require('../data/db');
```

```
async function listarCadastros(req, res) {
  try {
    const result = await pool.query('SELECT * FROM cadastros ORDER BY id
DESC');
    res.json(result.rows);
  } catch (error) {
    console.error('Erro ao listar cadastros:', error);
    res.status(500).json({ erro: "Erro ao listar cadastros" });
  }
}

async function cadastrar(req, res) {
  try {
    const { nome, email, senha } = req.body;
    const result = await pool.query(
      'INSERT INTO cadastros (nome, email, senha) VALUES ($1, $2, $3) RET
URNING *',
      [nome, email, senha]
    );
    res.status(201).json({ mensagem: "Cadastro realizado", cadastro: result.r
ows[0] });
  } catch (error) {
    console.error('Erro ao cadastrar:', error);
    res.status(500).json({ erro: "Erro ao cadastrar" });
  }
}

async function atualizar(req, res) {
  try {
    const { id } = req.params;
    const { nome, email, senha } = req.body;

    const result = await pool.query(
      'UPDATE cadastros SET nome=$1, email=$2, senha=$3 WHERE id=$4
RETURNING *',
      [nome, email, senha, id]
    );
```

```
    if (result.rowCount === 0) {
      return res.status(404).json({ erro: "Cadastro não encontrado" });
    }

    res.json({ mensagem: "Cadastro atualizado", cadastro: result.rows[0] });
  } catch (error) {
    console.error('Erro ao atualizar cadastro:', error);
    res.status(500).json({ erro: "Erro ao atualizar" });
  }
}

async function remover(req, res) {
  try {
    const { id } = req.params;
    const result = await pool.query('DELETE FROM cadastros WHERE id=$1',
[id]);

    if (result.rowCount === 0) {
      return res.status(404).json({ erro: "Cadastro não encontrado" });
    }

    res.json({ mensagem: "Cadastro removido" });
  } catch (error) {
    console.error('Erro ao deletar cadastro:', error);
    res.status(500).json({ erro: "Erro ao deletar" });
  }
}

module.exports = {
  listarCadastros,
  cadastrar,
  atualizar,
  remover,
};
```

📁 Novo arquivo: `routes/cadastroRoutes.js`

```
js
CopiarEditar
const express = require('express');
const router = express.Router();
const cadastroController = require('../controllers/cadastroController');

router.get('/cadastros', cadastroController.listarCadastros);
router.post('/cadastrar', cadastroController.cadastrar);
router.put('/cadastros/:id', cadastroController.atualizar);
router.delete('/cadastros/:id', cadastroController.remover);

module.exports = router;
```

📄 Atualiza o `server.js`

```
js
CopiarEditar
const express = require("express");
const cors = require("cors");
require("dotenv").config();

const app = express();
const port = 3001;

const cadastroRoutes = require("./routes/cadastroRoutes");

app.use(cors({ origin: "http://localhost:5173" }));
app.use(express.json());

app.use(cadastroRoutes); // rotas separadas

app.listen(port, () => {
  console.log(`Servidor rodando na porta ${port}`);
});
```

1. Criar a tabela `cadastros`

Você precisa criar essa tabela no banco, com a estrutura certa.

Opção A: Criar manualmente via terminal do container PostgreSQL

1. Entre no container do postgres:

```
bash
CopiarEditar
docker exec -it postgres psql -U myuser -d mydb
```

1. Dentro do prompt do `psql`, cole o seguinte comando SQL:

```
sql
CopiarEditar
CREATE TABLE IF NOT EXISTS cadastros (
  id SERIAL PRIMARY KEY,
  nome TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  senha TEXT NOT NULL
);
```

1. Digite `\q` para sair do psql.

✅ Verificação final

1. Sobe os containers:

```
docker-compose up --build
```

1. Testa as rotas no Thunder Client/Postman:

- POST `/cadastrar`
- GET `/cadastros`
- PUT `/cadastros/:id`
- DELETE `/cadastros/:id`

1. Verifica no PGAdmin se os dados estão sendo inseridos corretamente

Super rápido, Muri! 🚀

Vou te passar um passo a passo enxuto para conectar o Mongo no backend, criar um model básico e fazer o cadastro funcionar — tudo em minutos.

MongoDB no backend com Mongoose — Passo a passo rápido

1. Instala o mongoose no backend

```
cd backend
npm install mongoose
```

2. Cria a conexão com Mongo (`data/mongo.js`)

```
const mongoose = require('mongoose');

const mongoURI = process.env.MONGO_URI || 'mongodb://localhost:2701
7/mydb';

async function connectMongo() {
  try {
    await mongoose.connect(mongoURI);
    console.log('MongoDB conectado!');
  } catch (err) {
    console.error('Erro ao conectar no MongoDB:', err);
  }
}

module.exports = connectMongo;
```

3. Cria o model (`models/Cadastro.js`)

```
const mongoose = require('mongoose');

const cadastroSchema = new mongoose.Schema({
  nome: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  senha: { type: String, required: true }
});

const Cadastro = mongoose.model('Cadastro', cadastroSchema);

module.exports = Cadastro;
```

4. Atualiza o controller para usar o model Mongo

Exemplo para o cadastrar:

```
const Cadastro = require('../models/Cadastro');

async function cadastrar(req, res) {
  try {
    const { nome, email, senha } = req.body;
    const novoCadastro = new Cadastro({ nome, email, senha });
    await novoCadastro.save();
    res.status(201).json({ mensagem: 'Cadastro realizado', cadastro: novoCa
dastro });
  } catch (error) {
    console.error('Erro ao cadastrar:', error);
    res.status(500).json({ erro: 'Erro ao cadastrar' });
  }
}
```

Você adapta o resto do controller (listar, atualizar, remover) de forma similar.

5. No `server.js` importa e conecta o Mongo no início

```
const connectMongo = require('./data/mongo');
```

```
connectMongo();
```

6. Configura variáveis de ambiente para o Mongo no `.env`

```
MONGO_URI=mongodb://mongo:27017/mydb
```

Usando o nome do container `mongo` se estiver no Docker (exemplo `docker-compose`).

7. Testa a conexão

Suba o container do mongo, o backend e teste as rotas.

Se quiser, te ajudo a montar o `docker-compose.yml` para o mongo também. Quer seguir? Tá tranquilo? Quer que faça rápido um boilerplate só com mongo pra você?

Tô aqui pra te dar a força, Muri! 💪😄