

ChatGPTChatGPT



mudar nome do ultimo commit

Para alterar o nome (mensagem) do último commit no Git, use o seguinte comando:

```
bash

git commit --amend -m "Nova mensagem do commit"
```

Se o commit já foi enviado para um repositório remoto, você precisará forçar a atualização:

```
bash

git push --force
```

⚠ **Cuidado** ao usar `--force`, pois isso pode sobrescrever o histórico compartilhado com outras pessoas. Se outras pessoas já puxaram esse commit, pode ser melhor evitar forçar a atualização.



```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000
  }
}) refresh force
```

Se você deseja forçar o refresh no Vite sempre que fizer uma alteração no código, pode tentar algumas abordagens:

1 Ativar Full Reload no Vite

Adicione a opção `server.watch.usePolling` no seu `vite.config.js` para garantir que o Vite detecte todas as alterações:

```
js

import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    watch: {
      usePolling: true
    }
  }
})
```

```
server: {
  port: 3000,
  watch: {
    usePolling: true
  }
}
```

Isso ajuda principalmente em sistemas de arquivos que não detectam mudanças automaticamente (como WSL ou Docker).

2 Forçar o Refresh com um Plugin

Se quiser forçar um refresh total sempre que houver uma mudança, pode usar o ``vite-plugin-full-reload``:

```
bash

npm install vite-plugin-full-reload --save-dev
```

E no ``vite.config.js``:

```
js

import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import FullReload from 'vite-plugin-full-reload'

export default defineConfig({
  plugins: [react(), FullReload(['**/*'])], // O '**/*' força reload em qualquer arquivo alterado
  server: {
    port: 3000
  }
})
```

Isso faz com que o Vite recarregue completamente a página quando qualquer arquivo for alterado.

3 Ativar Hot Module Replacement (HMR) Manualmente

Se o HMR (Hot Module Replacement) não estiver funcionando corretamente, tente adicionar isso no ``vite.config.js``:

```
js

server: {
  port: 3000,
  hmr: {
    overlay: false // Desativa o overlay de erro, mas mantém o HMR
  }
}
```

Isso pode ajudar se o HMR estiver quebrando devido a erros na página.

Se ainda estiver com problemas, me avise! 🚀



Mude tudo para import ES6 e separe em arquivos

Aqui está a estrutura separada em arquivos usando importação ES6:

Estrutura do projeto

```
arduino

/project-root
├── vite.config.js
├── /config
│   ├── plugins.js
│   └── server.js
```

plugins.js (Configuração dos plugins)

```
js

import react from '@vitejs/plugin-react';
export const plugins = [react()];
```

server.js (Configuração do servidor)

```
js

export const server = {
  port: 3000,
  watch: {
    usePolling: true
  }
};
```

vite.config.js (Configuração principal do Vite)

```
js

import { defineConfig } from 'vite';
import { plugins } from './config/plugins.js';
import { server } from './config/server.js';

export default defineConfig({
  plugins,
  server
});
```

Agora o código está modularizado, usando ES6 `import/export`. 🚀



```
// src/server.js
const express = require('express')
```

```
const mongoose = require('mongoose')
const cors = require('cors')
const dotenv = require('dotenv')

dotenv.config()

const app = express()

app.use(cors())
app.use(express.json())

mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log('Conectado ao MongoDB'))
.catch(err => console.error('Erro ao conectar ao MongoDB:', err))

// Rotas
const productRoutes = require('./routes/products')
app.use('/api/products', productRoutes)

const PORT = process.env.PORT || 3000
app.listen(PORT, () => {
  console.log(`Servidor rodando na porta ${PORT}`)
})

// src/models/Product.js
const mongoose = require('mongoose')

const productSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  price: {
    type: Number,
    required: true
  },
  image: {
    type: String,
    required: true
  },
  stock: {
    type: Number,
    required: true,
    min: 0
  }
})
```

```

},
active: {
  type: Boolean,
  default: true
}
}, {
  timestamps: true
})

module.exports = mongoose.model('Product', productSchema)

// src/routes/products.js
const express = require('express')
const router = express.Router()
const Product = require('../models/Product')

// GET - Listar produtos com paginação
router.get('/', async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1
    const perPage = 9
    const total = await Product.countDocuments({ active: true })

    const products = await Product.find({ active: true })
      .skip((page - 1) * perPage)
      .limit(perPage)
      .sort('-createdAt')

    res.json({
      products,
      total,
      perPage,
      currentPage: page,
      totalPages: Math.ceil(total / perPage)
    })
  } catch (error) {
    res.status(500).json({ message: error.message })
  }
})

// POST - Criar produto
router.post('/', async (req, res) => {
  try {
    const product = new Product(req.body)
    await product.save()
    res.status(201).json(product)
  } catch (error) {
    res.status(400).json({ message: error.message })
  }
})

```

```
// PUT - Atualizar produto
router.put('/:id', async (req, res) => {
  try {
    const product = await Product.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true }
    )
    if (!product) {
      return res.status(404).json({ message: 'Produto não encontrado' })
    }
    res.json(product)
  } catch (error) {
    res.status(400).json({ message: error.message })
  }
})

// DELETE - Remover produto
router.delete('/:id', async (req, res) => {
  try {
    const product = await Product.findByIdAndDelete(req.params.id)
    if (!product) {
      return res.status(404).json({ message: 'Produto não encontrado' })
    }
    res.json({ message: 'Produto removido com sucesso' })
  } catch (error) {
    res.status(500).json({ message: error.message })
  }
})

module.exports = router
```

Aqui está seu código atualizado para usar **ES6 modules** (`import/export`):

Estrutura do projeto

```
bash

/project-root
├── /src
│   ├── server.js
│   ├── /config
│   │   └── db.js
│   ├── /models
│   │   └── Product.js
│   ├── /routes
│   │   └── products.js
│   ├── /controllers
│   │   └── productController.js
├── .env
└── package.json
```

src/config/db.js (Conexão com o MongoDB)

```
js

import mongoose from 'mongoose';

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true
    });
    console.log('Conectado ao MongoDB');
  } catch (err) {
    console.error('Erro ao conectar ao MongoDB:', err);
    process.exit(1);
  }
};

export default connectDB;
```

src/models/Product.js (Modelo do Produto)

```
js

import mongoose from 'mongoose';

const productSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    description: { type: String, required: true },
    price: { type: Number, required: true },
    image: { type: String, required: true },
    stock: { type: Number, required: true, min: 0 },
    active: { type: Boolean, default: true }
  },
  { timestamps: true }
);

const Product = mongoose.model('Product', productSchema);
export default Product;
```

src/controllers/productController.js (Lógica dos produtos)

```
js

import Product from '../models/Product.js';

// GET - Listar produtos com paginação
export const getProducts = async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1;
    const perPage = 9;
    const total = await Product.countDocuments({ active: true });

    const products = await Product.find({ active: true })
      .skip((page - 1) * perPage)
      .limit(perPage)
      .sort('-createdAt');

    res.json({
      products,
      total,
      perPage,
      currentPage: page,
      totalPages: Math.ceil(total / perPage)
    });
  }
};
```

```

    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  };

// POST - Criar produto
export const createProduct = async (req, res) => {
  try {
    const product = new Product(req.body);
    await product.save();
    res.status(201).json(product);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};

// PUT - Atualizar produto
export const updateProduct = async (req, res) => {
  try {
    const product = await Product.findByIdAndUpdate(req.params.id, req.body, {
      new: true
    });
    if (!product) return res.status(404).json({ message: 'Produto não encontrado' });

    res.json(product);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};

// DELETE - Remover produto
export const deleteProduct = async (req, res) => {
  try {
    const product = await Product.findByIdAndDelete(req.params.id);
    if (!product) return res.status(404).json({ message: 'Produto não encontrado' });

    res.json({ message: 'Produto removido com sucesso' });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

```

src/routes/products.js (Rotas do Produto)

```

js

import express from 'express';
import { getProducts, createProduct, updateProduct, deleteProduct } from '../controllers/productCon

const router = express.Router();

router.get('/', getProducts);
router.post('/', createProduct);
router.put('/:id', updateProduct);
router.delete('/:id', deleteProduct);

export default router;

```

src/server.js (Servidor Principal)

```

js

import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import connectDB from './config/db.js';

```



```
import productRoutes from './routes/products.js';

dotenv.config();
const app = express();

// Conectar ao MongoDB
connectDB();

app.use(cors());
app.use(express.json());

// Rotas
app.use('/api/products', productRoutes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor rodando na porta ${PORT}`);
});
```

.env (Arquivo de configuração)

```
ini

MONGODB_URI=mongodb+srv://seu-usuario:senha@seu-cluster.mongodb.net/seu-banco
PORT=3000
```

O que foi melhorado?

- ✓ **Código modularizado** → Separação em controllers, rotas e config
- ✓ **Uso de ES6 (import/export)** → Código mais moderno e compatível
- ✓ **Melhor organização** → Mais fácil de manter e escalar

Agora seu backend está mais organizado e pronto para crescer!  