**Documentação Técnica da API de Controle de Inventário**

**Introdução**

Foi necessária uma criação de api para estruturar o desenvolvimento de um mini mercado

Ela permite atualizar todos os produtos e tempo real, seja adicionando, removendo e fazendo alterações em seus dados

**Estrutura do Projeto**

### A estrutura foi desenvolvida de forma simples para proporcionar entendimento e execução por qualquer pessoa, ate mesmo as sem entendimento da área

### inventory-api/ \*criação de uma pasta para hospedar o banco de dados junto de suas tabelas

### Host(pasta)->cnct.js \* conexão com o bando de dados do mercado(nome, usuário e senha)

### 

### Controlador(pasta)-> -> controller.js \*pasta onde estão guardados os comandos para edição das tabelas (adicionar, deletar, atualizar)

### Rotas(pasta)-> -> transaction.js \* Pasta na qual armazena as rotas da API juntamente com a estrutura da rota dos produtos

### .env \* Arquivo de variáveis de ambiente

### server.js \* Arquivo principal do servidor

### package.json \* Arquivo de configuração do npm, listando as dependências do projeto

**Configuração do Ambiente**

**Variáveis de Ambiente**

O arquivo .env armazena as informações necessárias para acessar o bando de dados de sua loja

DB\_HOST=localhost

DB\_USER=root

DB\_PASS=031024

DB\_NAME=inventory\_db

PORT=3000

**Dependências**

As dependências incluem tudo aquilo que é necessário para rodar todo o sistema sem causar nenhum erro, obrigatório que seja feito no seu projeto as seguintes instalações

### npm init -y

### npm install express mysql2 dotenv cors body-parser

### npm install nodemon --save-dev

**Banco de Dados**

**Estrutura da Tabela**

Para armazenar os produtos no inventário, precisamos criar uma tabela no banco de dados MySQL.

### CREATE DATABASE inventory\_db;

### USE inventory\_db;

### CREATE TABLE products (

### id INT AUTO\_INCREMENT PRIMARY KEY,

### name VARCHAR(255)

### description TEXT

### category VARCHAR(255)

### price DECIMAL(10, 2)

### stock INT

### expiry\_date

**Configuração do Projeto**

**Configuração do Banco de Dados**

O arquivo host/cnct.js configura a conexão com o banco de dados MySQL utilizando as variáveis de ambiente definidas no arquivo .env:

**Arquivo host/cnct.js**:

const mysql = require('mysql2');

// conecção com o mysql

const cnct = mysql.createConnection({

    host: process.env.DB\_HOST,

    user: process.env.DB\_USER,

    password: process.env.DB\_PASS,

    database: process.env.DB\_NAME,

});

// uso das databases

cnct.connect((err) => {

    if (err) {

        console.error('Erro ao se conectar ao banco de dados:', err);

        return;

    }

    console.log(`Conectado ao banco de dados, bem-vindo: ${process.env.DB\_NAME}`);

});

module.exports = cnct;

**Controladores**

Os controladores contêm a lógica para manipular as requisições e respostas HTTP. Facilitando uma melhor interação e entendimento para o as pessoas na hora de interagirem com o programa

O código do arquivo controlador/controller.js, contém funções para obter, adicionar, atualizar e deletar produtos:

**Arquivo controlador/controller.js:**

const cnct = require('../host/cnct');

// SELECIONAR TODOS

const getAllTransactions = (req, res) => {

    cnct.query('select \* from products', (err, results) => {

        if (err) {

            console.error('Erro ao obter transações', err);

            res.status(500).send('Erro ao obter transações');

            return;

        }

        res.json(results);

    });

};

// ADICIONAR PRODUTOS

const addTransactions = (req, res) => {

    const { nome\_produto, desc\_produto, categoria, preco, QNT\_STQ, data\_validade } = req.body;

    cnct.query(

        'insert into products (nome\_produto, desc\_produto, categoria, preco, QNT\_STQ, data\_validade) VALUES (?, ?, ?, ?, ?, ?)',

        [nome\_produto, desc\_produto, categoria, preco, QNT\_STQ, data\_validade],

        (err, results) => {

            if (err) {

                console.error('Erro ao adicionar transação', err);

                res.status(500).send('Erro ao adicionar transação');

                return;

            }

            res.send('Transação adicionada com sucesso');

        }

    );

};

// UPDATE TOTAL

const updateTransactionsPut = (req, res) => {

    const { id } = req.params;

    const { nome\_produto, desc\_produto, categoria, preco, QNT\_STQ, data\_validade } = req.body;

    cnct.query(

        'update products set nome\_produto = ?, desc\_produto = ?, categoria = ?, preco = ?, QNT\_STQ = ?, data\_validade = ? WHERE id = ?',

        [nome\_produto, desc\_produto, categoria, preco, QNT\_STQ, data\_validade, id],

        (err, results) => {

            if (err) {

                console.error('Erro ao atualizar transação', err);

                res.status(500).send('Erro ao atualizar transação');

                return;

            }

            res.send('Transação atualizada com sucesso');

        }

    );

};

//UPDATE PARCIAL

const updateTransactionPatch = (req, res) => {

    const { id } = req.params;

    const fields = req.body;

    const query = [];

    const values = [];

    for (const [key, value] of Object.entries(fields)) {

        query.push(`${key} = ?`);

        values.push(value);

    }

    values.push(id);

    cnct.query(

        `update products set ${query.join(', ')} where id = ?`,

        values,

        (err, results) => {

            if (err) {

                console.error('Erro ao atualizar transação', err);

                res.status(500).send('Erro ao atualizar transação');

                return;

            }

            res.send('Transação atualizada com sucesso');

        }

    );

};

//DELETAR

const deleteTransaction = (req, res) => {

    const { id } = req.params;

    cnct.query(

        'delete from products where id = ?',

        [id],

        (err, results) => {

            if (err) {

                console.error('Erro ao deletar transação', err);

                res.status(500).send('Erro ao deletar transação');

                return;

            }

            res.send('Transação deletada com sucesso');

        }

    );

};

module.exports = {

    getAllTransactions,

    addTransactions,

    updateTransactionsPut,

    updateTransactionPatch,

    deleteTransaction

};

**Rotas**

As rotas definem o local onde há uma interação entre dois sistemas diferentes (endpoints) cada um deles associa cada rota a uma função no controlador.

O arquivo routes/products.js contém as definições de rotas para executar todas as funções atribuídas para cada rota designada:

**Arquivo rotas/transaction.js**:

const express = require('express');

const router =  express.Router();

const controller = require ('../controlador/controller');

//seleciona todos os produtos

router.get('/',controller.getAllTransactions);

//adiciona um novo produto

router.post('/',controller.addTransactions);

//atualiza um produto de forma total

router.put('/:id',controller.updateTransactionsPut);

//atualiza um produto de forma parcial

router.patch('/:id',controller.updateTransactionPatch);

//deleta um produto por id

router.delete('/:id',controller.deleteTransaction);

module.exports = router;

**Servidor**

O arquivo server.js configura e inicializa o servidor “Express”. Ele também define as rotas para a API e inicia o servidor na porta especificada nas variáveis de ambiente:

**Arquivo server.js**:

// biblioteca

const dotenv = require('dotenv');

dotenv.config();

const express = require('express');

const cors = require('cors');

const bodyParser = require('body-parser');

// rotas,databases

const cnct = require('./host/cnct');

const TransRotas = require('./rotas/transaction');

const app = express();

// Configurações Express

app.use(cors());

app.use(bodyParser.json());

// Rotas

app.use('/api/transactions', TransRotas);

app.get('/', (req, res) => {

    res.send('Servidor ativo com sucesso');

});

// Definir porta padrão

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

    console.log(`Servidor rodando na porta ${PORT}`);

});

**Testes da API**

Para garantir que a API está funcionando corretamente, deve ser realizados testes usando o “Insomnia”. Vamos testar cada endpoint e verificar como a API lida com diferentes tipos de requisições, incluindo casos de sucesso e casos de erro.

**Teste com Insomnia**

**1. Adicionar um Novo Produto**

### Método: POST

### URL: http://localhost:3000/api/transaction

### Corpo: JSON

"id": 14,

"nome\_produto": "creme-Cracker",

"desc\_produto": "Biscoito de água e sal",

"categoria": "comida",

"preco": "4.99",

"QNT\_STQ": 40,

"data\_validade": "2025-02-01T03:00:00.000Z"

**Resposta de Sucesso:**

"message": "produto adicionado com sucesso"

**2. Atualizar um Produto Existente (Substituição total)**

### Método: PUT

### URL: http://localhost:3000/api/transiction/15

### Corpo:JSON

"nome\_produto": "Bolacha",

"desc\_produto": "Biscoito de água",

"categoria": "Alimento",

"preco": "6.00",

"QNT\_STQ": 35,

"data\_validade":"2025-02-28"

**Resposta de Sucesso:**

"message": "produto atualizado com sucesso"

**Resposta de Produto Não Encontrado (404):**

"message": "Produto não encontrado"

**3. Atualizar um Produto Existente (Atualização Parcial)**

**Requisição:**

Método: PATCH

URL: http://localhost:3000/api/products/1

Corpo:JSON

"id": 15,

"nome\_produto": "sabão liquido",

"desc\_produto": "usar na sua lava roupa",

"categoria": "produto de limpeza",

"preco": "21.99"

**Resposta de Sucesso:**

"message": "Produto atualizado com sucesso"

**Resposta de Produto Não Encontrado (404):**

"message": "Produto não encontrado"

**4. Deletar um Produto Existente**

**Requisição:**

Método: DELETE

URL: http://localhost:3000/api/products/15

**Resposta de Sucesso:**

"message": "Produto deletado com sucesso"

**Resposta de Produto Não Encontrado (404):**

"message": "Produto não encontrado"

**5. Listar Todos os Produtos**

**Requisição:**

Método: GET

URL: http://localhost:3000/api/products

**Resposta de Sucesso:**

"id": 1,

"nome\_produto": "biscoito",

"desc\_produto": "doce de qualidade",

"categoria": "comida",

"preco": "2.50",

"QNT\_STQ": 42,

"data\_validade": "2033-12-21T03:00:00.000Z"

"id": 2,

"nome\_produto": "Bolacha",

"desc\_produto": "Biscoito de água",

"categoria": "Alimento",

"preco": "6.00",

"QNT\_STQ": 35,

"data\_validade": "2025-02-28T03:00:00.000Z"

**Resposta de Erro Interno do Servidor (500):**

"message": "Erro no código fonte"

**Resolução de Problemas Comuns**

**Erro 404 (Not Found)**

### quando o usuário acessa uma página de um site e encontra a seguinte mensagem: “Error 404 Not Found”, isso significa que a página buscada não foi encontrada dentro do domínio.

### Existem vários motivos para isso ocorrer, entre eles podemos destacar o erro de digitação do usuário, outro motivo bem comum é a remoção de uma página do site, ou a modificação na URL sem a realização do redirecionamento.

**Erro 500 (Internal Server Error)**

Os motivos mais comuns para o erro 500 são arquivos corrompidos.

Uma das melhores coisas para fazer para solucionar esse erro é direcionar sua atenção para aquilo que estava fazendo antes do problema ocorrer