**VERONICA NASCIMENTO.**

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

**Introdução**

Esta documentação técnica descreve a implementação de uma API RESTful para um sistema de controle de inventário de uma pequena loja.

A API permite adicionar, atualizar, deletar e listar produtos no inventário. Cada produto possui um nome, descrição, categoria, preço, quantidade em estoque e data de validade.

**Estrutura do Projeto**

A estrutura do projeto é organizada de forma a facilitar a manutenção e o desenvolvimento da API.

Abaixo está a estrutura do projeto, juntamente com uma breve descrição de cada pasta e arquivo:

**Veronica folheados**/

├── config/

│ └── db.js **//** O arquivo db.js conterá a lógica para conectar ao banco de dados MySQL

├── controllers // armazenará os arquivos de controladores da aplicação.

│ └── authController.js // configurar o Registro e Login de Usuários

│ └── clientesController.js // Este arquivo conterá as definições de rotas relacionadas às transações financeiras**.**

├── middlewares/ // verificar se o token JWT está presente e válido

│ └── authMiddleware.js //

├── node modules //

├── routes // define as rotas de transações

│ └── auth.js // Definindo Rotas para Registro e Login

│ └── clientes.js // define rotas para adicionar, atualizar e deletar

* emailServices //

├── .env // armazena variáveis de ambiente

├── .gitignore // são as pastas ocultas

├── package.json // instalar as dependencias

├── package-lock.json

├── README

├── server.js //importar as bibliotecas

**Configuração do Ambiente**

**Variáveis de Ambiente**

O arquivo .env na raiz do projeto armazena as variáveis de ambiente que são utilizadas para configurar a conexão com o banco de dados e a porta em que o servidor será executado:

Arquivo .env

DB\_HOST=localhost

DB\_USER=root

DB\_PASS=\*\*\*\*\*\*\*\*

DB\_NAME=veronica folheados

PORT=3000

**Dependências**

As dependências incluem bibliotecas para criar o servidor, conectar-se ao banco de dados, gerenciar variáveis de ambiente, lidar com CORS e analisar o corpo das requisições HTTP:

**npm init -y**

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

**npm install nodemon --save-dev**

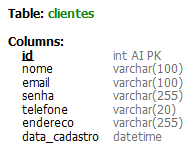
**Banco de Dados**

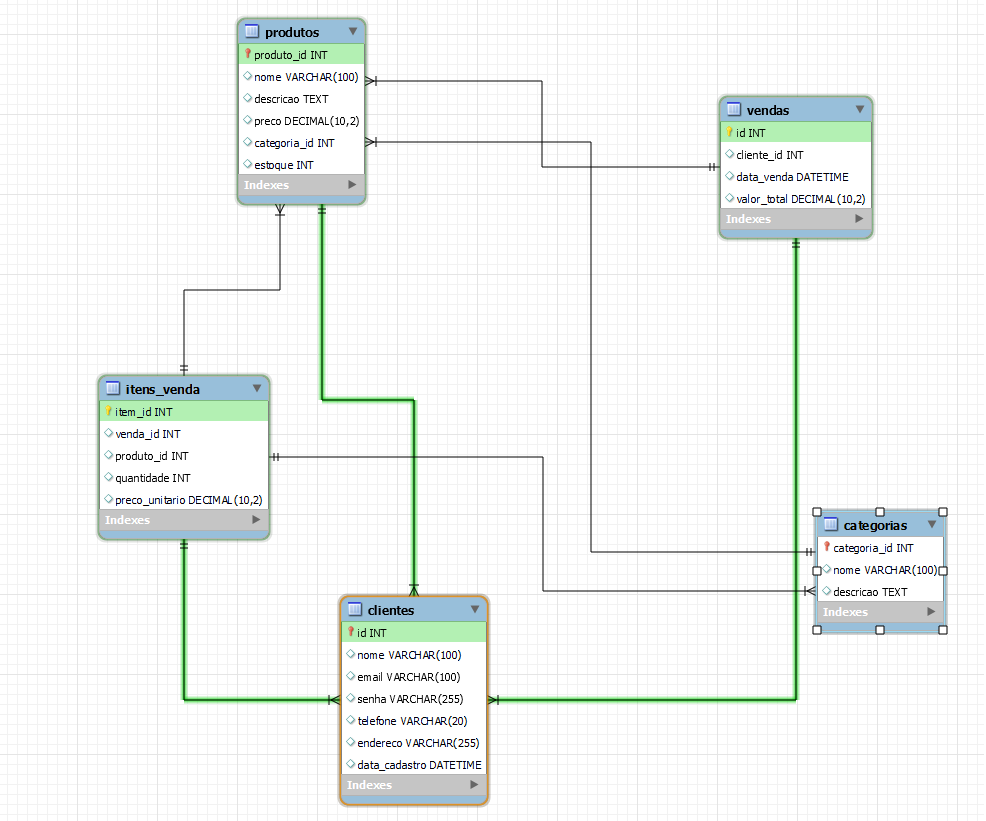
**Estrutura da Tabela**

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

**CREATE DATABASE veronica\_folheados;**

**USE veronica\_folheados;**





**Configuração do Projeto**

**Configuração do Banco de Dados**

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

* Arquivo config/db.js:

const mysql = require('mysql2');

const dotenv = require('dotenv');

dotenv.config();

const db = mysql.createConnection({

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASS,

database: process.env.DB\_NAME

});

db.connect((err) => {

if (err) {

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

return;

}

console.log('Conectado ao banco de dados MySQL');

});

module.exports = db;

**Controladores**

Os controladores contêm a lógica para manipular as requisições e respostas HTTP. Eles interagem com o banco de dados e retornam as respostas apropriadas para o cliente.

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

* Arquivo controllers/clientesController.js:

const db = require('../config/db');

// Função para obter todos os produtos

const getAllclientes = (req, res) => {

db.query('SELECT \* FROM clientes', (err, results) => {

if (err) {

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

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

return;

}

res.json(results);

});

};

**// Função para adicionar uma nova transação**

const addclientes = (req, res) => {

  const {nome, email, senha, telefone, endereco, data\_cadastro} = req.body;

  db.query(

    'INSERT INTO clientes (nome, email, senha, telefone, endereco, data\_cadastro) VALUES (?,?,?,?,?,?)',

    [nome, email, senha, telefone, endereco, data\_cadastro],

    (err, results) => {

      if (err) {

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

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

        return;

      }

      res.status(201).send('cliente adicionado com sucesso');

    }

  );

};

**// Se a cliente não existe, insira-a no banco de dados**

  db.query(

    'INSERT INTO clientes (id,nome, email, senha, telefone, endereco, data\_cadastro) VALUES (?,?,?,?,?,?)',

    [nome, email, senha, telefone, endereco, data\_cadastro],

    (err,results) => {

        if(err) {

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

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

            return;

        }

        res.status(201).send('cliente adicionado com sucesso');

    }

);

**//------------ Verificar se a Transação Existe (PUT, PACTH, DELETE)-------------//**

**//Função para atualizar uma transação existente (substituição completa)**

const updateClientesPut = (req, res) => {

const{id} = req.params;

const {} = req.body;

db.query(

'UPDATE clientes SET nome = ?, senha = ?, telefone = ?, endereco = ?, data\_cadastro = ?, WHERE id = ?',

[nome, email, senha, telefone, endereco, data\_cadastro],

(err,results) => {

if(err) {

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

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

return;

}

**// verifica se nenhuma linha foi afetada pela consulta**

if(results.affectedRows===0){

res.status(404).send('Transação não encontrada');

return;

}

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

}

);

};

**//Função para atualizar uma transação existente (substituição parcial)**

const updateClientesPatch = (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);

db.query(

`UPDATE clientes SET ${query.join(',')} WHERE id = ?`,

values,

(err,results) => {

if(err) {

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

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

return;

}

**// verifica se nenhuma linha foi afetada pela consulta**

if(results.affectedRows===0){

res.status(404).send('Transação não encontrada');

return;

}

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

}

);

};

**//Função para deletar uma transação existente**

const deleteclientes = (req,res) => {

const{id} = req.params;

db.query('DELETE FROM clientes WHERE id = ?',[id],

(err,results) => {

if(err) {

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

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

return;

}

**// verifica se nenhuma linha foi afetada pela consulta**

if(results.affectedRows===0){

res.status(404).send('cliente não encontrada');

return;

}

res.send('cliente deletado com sucesso');

}

);

};

module.exports = {

getAllClientes,

addclientes,

updateClientesPut,

updateClientesPatch,

deleteclientes

};

**Rotas**

As rotas definem os endpoints da API e associam cada rota a uma função no controlador.

O arquivo routes/products.js contém as definições de rotas para obter, adicionar, atualizar e deletar produtos:

* Arquivo routes/products.js:

const express = require('express'); // Importa o framework Express

const router = express.Router(); // Cria um novo roteador

const productsController = require('../controllers/productsController'); // Importa o controlador de produtos

// Definindo uma rota para obter todos os produtos

router.get('/', productsController.getAllProducts);

// Definindo uma rota para adicionar um novo produto

router.post('/', clientesController.addclientes);

// Definindo uma rota para atualizar um produto existente (substituição completa)

router.put('/:id', clientesController.updateclientesPut);

// Definindo uma rota para atualizar um cliente existente (atualização parcial)

router.patch('/:id', clientesController.updateclientePatch);

// Definindo uma rota para deletar um cliente existente

router.delete('/:id', clientesController.deleteclientes);

// Exportando o roteador

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:

const dotenv = require('dotenv'); **// Importa o pacote dotenv para gerenciar variáveis de ambiente**

**// Carregar as Variáveis de Ambiente**

dotenv.config(); **// Carrega as variáveis definidas no arquivo** .env para process.env

**// Importar as Bibliotecas**

const express = require('express'); **// Importa o framework Express**

const cors = require('cors'); **// Importa o pacote cors para permitir requisições de diferentes origens**

const bodyParser = require('body-parser'); **// Importa o pacote body-parser para analisar o corpo das requisições HTTP**

const db = require('./config/db'); **// Importa a conexão com o banco de dados**

**// Inicializar nova aplicação Express**

const app = express(); // Inicializa uma nova aplicação Express

**// Configurar o CORS e o body-parser**

app.use(cors()); **// Habilita o CORS para todas as rotas**

app.use(bodyParser.json()); **// Configura o body-parser para analisar requisições JSON**

**// Importar as rotas de produtos**

const clientesRoutes = require('./routes/clientes'); **// Importa as rotas de clientes**

**// Usar as rotas de produtos para todas as requisições que começam com /api/clientes**

app.use('/api/clientes', clientesRoutes**); // Configura o servidor para usar as rotas de clientes**

**// Rota inicial para testar o servidor**

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

res.send('Servidor está rodando'); **// Define uma rota inicial para testar o servidor**

});

**// Configurar o servidor para uma porta específica**

const PORT = process.env.PORT || 3000; **// Define a porta a partir da variável de ambiente ou usa a porta 3000 como padrão**

app.listen(PORT, () => {

console.log(`Servidor rodando na porta ${PORT}`); **// Loga uma mensagem informando que o servidor está rodando**

});

* **Arquivo** authController.js

const db = require('../config/db'); **// Importa a configuração do banco de dados**

const bcrypt = require('bcrypt'); **// Importa o bcrypt para criptografar senhas**

const jwt = require('jsonwebtoken'); **// Importa o jsonwebtoken para gerar tokens** JWT

**// Função para registrar um novo usuário**

const registerCLientes = async (req, res) => {

  const {nome, email, senha, telefone, endereco, data\_cadastro  } = req.body**; // Desestrutura os dados do corpo da requisição**

**// Verificar se o usuário já existe no banco de dados**

  try {

    const [existingClientes] = await db.promise().query('SELECT \* FROM clientes WHERE email = ?',

[email]);

    if (existingClientes.length > 0) {

      return res.status(400).send('Usuário já registrado');

    }

**// Criptografar a senha usando bcrypt**

    const hashedsenha = await bcrypt.hash(senha, 10);

**// Inserir o novo usuário no banco de dados**

    await db.promise().query(

      'INSERT INTO clientes (nome, email, senha, telefone, endereco, data\_cadastro) VALUES (?, ?, ?, ?, ?, ?)',

      [nome, email, senha, telefone, endereco, data\_cadastro]

    );

    res.status(201).send('Usuário registrado com sucesso');

  } catch (err) {

    console.error('Erro ao registrar usuário:', err);

    res.status(500).send('Erro ao registrar usuário');

  }

};

**//--------------------------------------------------------LOGIN---------------------------------------------------------------------------//**

**// Função para autenticar um usuário**

const loginClientes = async (req, res) => {

  const { email, senha } = req.body**; // Desestrutura os dados do corpo da requisição**

**// Verificar se o usuário existe no banco de dados**

  try {

    const [senha] = await db.promise().query('SELECT \* FROM senha WHERE email = ?', [email]);

    if (user.length === 0) {

      return res.status(400).send('Credenciais inválidas');

    }

**// Comparar a senha fornecida com a senha criptografada no banco de dados**

    const isMatch = await bcrypt.compare(senha, clientes[0].senha);

    if (!isMatch) {

      return res.status(400).send('Credenciais inválidas');

    }

    // Gerar um token JWT

    const token = jwt.sign({ clientesId: clientes[0].id }, process.env.JWT\_SECRET, { expiresIn: '1h' });

    res.json({ token });

  } catch (err) {

    console.error('Erro ao autenticar usuário:', err);

    res.status(500).send('Erro ao autenticar usuário');

  }

};

**//---------------------------------------------------------------------------RECUPERAR SENHA-------------------------------------------------------//**

**// Função para solicitar redefinição de senha**

const requestsenhaReset = async (req, res) => {

    const { email } = req.body;

    try {

      const [clientes] = await db.promise().query('SELECT \* FROM clientes WHERE email = ?', [email]);

      if (user.length === 0) {

        return res.status(404).send('Usuário não encontrado');

      }

      const token = crypto.randomBytes(20).toString('hex'); // Gera um token aleatório

      const expireDate = new Date(Date.now() + 3600000); // 1 hora para expiração

      await db.promise().query('UPDATE clientes SET reset\_\_ = ?, reset\_password\_expires = ? WHERE email = ?', [token, expireDate, email]);

      const resetLink = `http://localhost:3000/reset-senha/${token}`; // Link para redefinição de senha

      sendEmail(email, 'Recuperação de Senha', `Por favor, clique no link para redefinir sua senha: ${resetLink}`);

      res.send('E-mail de recuperação de senha enviado');

    } catch (err) {

      console.error('Erro ao solicitar redefinição de senha:', err);

      res.status(500).send('Erro ao solicitar redefinição de senha');

    }

  };

**//-------------------------------------------------------------------FUNCAO PARA REDEFINIR SENHA--------------------------------------------------**

const resetsenha = async (req, res) => {

    const { token, newsenha } = req.body;

    try {

      const [user] = await db.promise().query('SELECT \* FROM clientes WHERE reset\_senha\_token = ? AND reset\_senha\_expires > NOW()', [token]);

      if (user.length === 0) {

        return res.status(400).send('Token inválido ou expirado');

      }

      const hashedsenha = await bcrypt.hash(newsenha, 10); **// Criptografa a nova senha**

      await db.promise().query('UPDATE users SET password = ?, reset\_password\_token = NULL, reset\_password\_expires = NULL WHERE id = ?', [hashedPassword, user[0].id]);

      res.send('Senha redefinida com sucesso');

    } catch (err) {

      console.error('Erro ao redefinir senha:', err);

      res.status(500).send('Erro ao redefinir senha');

    }

  };

module.exports = {

  registerCLientes,

  loginClientes ,

  requestsenhaReset,

  resetsenha

};

* **Arquivo** AuthMiddlewares.js

const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {

const token = req.header('Authorization').replace('Bearer ', ''**); // Obtém o token do cabeçalho da requisição**

if (!token) {

return res.status(401).send('Acesso negado. Nenhum token fornecido.');

}

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET); **a validade d// Verifica o token**

req.user = decoded; **// Adiciona as informações do usuário à requisição**

next(); **// Passa o controle para a próxima função middleware**

} catch (err) {

res.status(400).send('Token inválido.');

}

};

module.exports = authMiddleware**; // Exporta o middleware de autenticação**

* **Arquivo** auth.js

const express = require('express'**); // Importa o framework Express**

const router = express.Router(); **// Cria um novo roteador**

const authController = require('../controllers/authController'**); // Importa o controlador de**

**autenticação**

**// Rota para registro de usuário**

router.post('/register', authController.registerUser);

**// Rota para login de usuário**

router.post('/login', authController.loginUser);

module.exports = router; **// Exporta o roteador**

* Email services

const nodemailer = require('nodemailer'); // Importa o módulo 'nodemailer', que será utilizado para enviar e-mails a partir da aplicação de forma simples e eficiente.

//--------------------------------------- Configuração do serviço de e-mail--------------------------------//

const transporter = nodemailer.createTransport({

 service: 'gmail', // Use o serviço de e-mail de sua escolha

host: 'smtp.gmail.com', // Use o serviço de e-mail de sua escolha

port: 465,

secure: true, //true se utilizar porta 465, false outras portas

auth: {

user: process.env.EMAIL\_USER, // Seu e-mail definido nas variáveis de ambiente

pass: process.env.EMAIL\_PASS // Sua senha de e-mail definida nas variáveis de ambiente

 }

});

//---------------------------------- Função para enviar e-mail---------------------------------------//

const sendEmail = (to, subject, text) => {

  const mailOptions = {

    from: process.env.EMAIL\_USER,

    to,

    subject,

    text

  };

  transporter.sendMail(mailOptions, (error, info) => {

    if (error) {

      return console.log('Erro ao enviar e-mail:', error);

    }

    console.log('E-mail enviado:', info.response);

  });

};

// Exporta a função 'sendEmail' para que ela possa ser utilizada em outros módulos da aplicação, permitindo o envio de e-mails a partir desses módulos.

module.exports = { sendEmail };

**Testes da API**

Para garantir que a API está funcionando corretamente, realizamos testes usando o Insomnia. Vamos testar cada endpoint da API 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**

A resposta deve indicar que a transação foi adicionada com sucesso.

**Requisição:**

* Método: POST
* URL: http://localhost:3000/veronicafolheados/clientes
* Corpo:

{

"nome": "nome do cliente",

"email": "email do cliente",

"senha": "\*\*\*\*\*\*\*\*",

"telefone":numero da cliente,

"endereco":emdereco do cliente,

"data\_cadastro": "data de cadastro"

}

**Resposta de Sucesso:**

{

"message": "cliente adicionado com sucesso"

}

**Resposta de Produto Duplicado:**

{

"message": "Produto duplicado"

}

1. **Atualizar um Produto Existente (Substituição Completa)**

A resposta deve indicar que a transação foi atualizada com sucesso.

**Requisição:**

* Método: PUT
* URL: http://localhost:3000/veronicafolheados/clientes/1
* Corpo:

{

"nome": "nome do cliente",

"email": "email do cliente",

"senha": "\*\*\*\*\*\*\*\*",

"telefone":numero da cliente,

"endereco":emdereco do cliente,

"data\_cadastro": "data de cadastro"

}

**Resposta de Sucesso:**

{

"message": "cliente atualizado com sucesso"

}

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

{

"message": "cliente não encontrado"

}

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

A resposta deve indicar que a transação foi atualizada com sucesso.

**Requisição:**

* Método: PATCH
* URL: http://localhost:3000/veronicafolheados/clientes/1
* Corpo:

{

"numero": numero atualizado,

"email": email atualizado

}

**Resposta de Sucesso:**

{

"message": "cliente atualizado com sucesso"

}

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

{

"message": "cliente não encontrado"

}

**4. Deletar um Produto Existente**

**Requisição:**

**A** resposta deve indicar que a transação foi atualizada com sucesso

* Método: DELETE
* URL: http://localhost:3000/veronicafolheados/clientes/1

**Resposta de Sucesso:**

{

"message": "cliente deletado com sucesso"

}

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

{

"message": "clientes não encontrado"

}

**5. Listar Todos os Produtos**

A resposta deve conter uma lista de transações no formato JSON, retornando todas as transações armazenadas no banco de dados.

**Requisição:**

* Método: GET
* URL: http://localhost:3000/veronicafolheados/clientes

**Resposta de Sucesso:**

[

{

"nome": "nome do cliente",

"email": "email do cliente",

"senha": "\*\*\*\*\*\*\*\*",

"telefone":numero da cliente,

"endereco":emdereco do cliente,

"data\_cadastro": "data de cadastro"

}

{

"nome": "nome do cliente",

"email": "email do cliente",

"senha": "\*\*\*\*\*\*\*\*",

"telefone":numero da cliente,

"endereco":emdereco do cliente,

"data\_cadastro": "data de cadastro"

}

]

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

{

"message": "Erro ao obter clientes"

}

**Resolução de Problemas Comuns**

**Erro 404 (Not Found)**

* **Causa**: O recurso solicitado não foi encontrado.
* **Solução**: Verifique se o ID do produto existe no banco de dados e se a URL da requisição está correta.

**Erro 500 (Internal Server Error)**

* **Causa**: Ocorreu um erro no servidor ao processar a requisição.
* **Solução**: Verifique os logs do servidor para identificar a causa do erro. Certifique-se de que a conexão com o banco de dados está configurada corretamente e que a query SQL está correta.

**Considerações Finais**

Após testar todos os endpoints utilizando uma ferramenta como o Insomnia ou Postman, visando garantir que a API esteja funcionando corretamente, foram aplicados os conceitos de tratamento de erros e verificação de duplicidade para manter a integridade dos dados no banco de dados.

**O que é a API**

API significa "Application Programming Interface" (Interface de Programação de Aplicações). É um conjunto de definições e protocolos que permite que diferentes softwares se comuniquem entre si. As APIs facilitam a interação entre diferentes sistemas, permitindo que um programa use funcionalidades ou dados de outro sem precisar saber como esse outro programa foi implementado. Elas são muito usadas em desenvolvimento de software para integrar serviços, acessar dados e expandir funcionalidades.

**O que é o token ?**

Nas APIs, um **token** é um pequeno pedaço de informação utilizado para autenticação e autorização de usuários ou sistemas. Ele é emitido por um servidor de autenticação após um processo de login bem-sucedido e é usado para identificar quem está fazendo a solicitação. Em vez de enviar informações de login (como usuário e senha) a cada vez que uma API é acessada, o cliente pode usar o token para se autenticar em futuras requisições.