1. **INTRODUÇÃO**

Para a realização do inventário dos produtos da loja “XXX” houve a necessidade da implementação de uma API (Interface de Programação de Aplicação) assim simplificando e acelerando o desenvolvimento do software, possibilitando que os desenvolvedores integrem dados, como o que foi feito nesse projeto.

1. **DESCRIÇÃO DO PROJETO**

Inicialmente utilizamos a ferramenta **Git Hub** que **é** uma plataforma de hospedagem de códigos e arquivos com controle de versão usando o Git, criamos um repositório com o nome “argent\_store” lá foram armazenados os códigos e dados que foram utilizados nesse projeto. Para a edição do código foi utilizado o **Visual Studio Code**.

Foi utilizado o **Node.js** que é um ambiente de execução do JavaScript no servidor, permitindo a execução do código JavaScript fora do navegador. Já o **npm**  é um gerenciador de pacotes do **Node.js** ele nos permite atualizar e remover bibliotecas e ferramentas adicionais, facilitando a gestão de dependências da aplicação.

Para iniciá-lo é necessário um comando **(npm init -y)** no terminal que criará um arquivo package.json com as configurações padrão. O arquivo package.json é usado para gerenciar as dependências e scripts do projeto.

Ao abrir o VS code iniciamos um novo arquivo com o nome **server.js** lá abriremos o terminal para a instalação das bibliotecas (conjunto de subprogramas ou funções, geralmente organizadas em classes, que podem ser usadas para a construção de um software ou aplicativo mobile tornando a utilização de uma linguagem de programação mais fácil) e ferramentas necessárias para o projeto como **cors, body-parser, dotenv, nodemon e mysql2** elas auxiliaram na criação do servidor, analisar o corpo das requisições HTTP (Protocolo de Transferência de Hipertexto), gerenciam as atividades de ambiente, reiniciam automaticamente o servidor quando os arquivos são modificados e conectam e interagem com bancos de dados MySQL.

Frameworks como **express e nestjs** somaram no projeto pois, são uma coleção de bibliotecas que facilitam o desenvolvimento ao oferecer soluções prontas para problemas comuns servindo como um guia, trilhando um caminho lógico e organizado no desenvolvimento de um projeto. Para instalação das dependências utilizamos os seguintes comandos no terminal: **npm install express cors body-parser dotenv e npm install nodemon --save-dev.**

1. **ESTRUTURA DO PROJETO**

Abaixo apresentamos a estrutura do projeto com uma breve descrição dos arquivos.

**argent\_store/**

**├── banco\_de\_dados/** - Pasta para armazenar o banco de dados

**├── config/** - Pasta de configuração de dados

**│ └── db.js** - Arquivo js para conexão de dados

**├── controllers/** - Pasta de controladores

**│ └── authController.js** - Arquivo js que configura o Registros e Logins

**│ └── carrinhoController.js** - Arquivo js controles de requisições do carrinho

**│ └── comprasController.js** - Arquivo js controles de requisições de compras

**│ └── estoqueController.js -** Arquivo js controles de requisições do estoque

**│ └── produtosController.js** - Arquivo js controles de requisições dos produtos

**├── middlewares/** - Pasta de middlewares

**│ └── authMiddleware.js -** Arquivo js para verificar se o token JWJ está presente

**├── public/** - Pasta para armazenar os protótipos

**├── routes/** - Pasta para rotas de API

**│ └── auth.js** - Arquivo para definição das rotas de autenticação

**│ └── carrinho.js** - Arquivo para definição das rotas do carrinho

**│ └── compras.js** - Arquivo para definição das rotas de compras

**│ └── estoque.js** - Arquivo para definição das rotas de estoque

**│ └── produtos.js** - Arquivo para definição das rotas dos produtos

**├── services/** - Pasta para armazenar arquivos de envio de Email

**│ └── emailService.js -** Arquivo js que configura o serviço de envio de Email

**├── .env**  - Arquivo para as variáveis de ambiente

**├── .gitignore -** Arquivo para ocultar pastas

**├── package.json** - Arquivo que guarda as dependências, a versão, o autor

**├── package-lock.json -** Arquivo queregistrar a versão exata que foi instalada

**├── server.js** - Arquivo js do servidor

1. **CONFIGURAÇÃO DE AMBIENTE**

* **Variáveis de ambiente**

No arquivo (.env) no projeto as variáveis tiveram um papel muito importante de armazenar os dados que o usuário e o sistema utilizaram, assim executando corretamente o servidor.

***DB\_HOST = localhost***

***DB\_USER = root***

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

***DB\_NAME = argentstore\_db***

***EMAIL\_USER= sistemarec22@gmail.com***

***EMAIL\_PASS = \*\*\*\*\*\*\*\*\*\*\*\****

***JWT\_SECRET = \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*****\*\**

1. **BANCO DE DADOS**

* **Estrutura da tabela**

Utilizamos **o MySQL (**é um sistema de gerenciamento de banco de dados relacional de código aberto baseado em SQL**)** para armazenar o banco de dados **“*argentstore\_db*”** e as tabelas nomeadas como, **“Registro, Produtos, Estoque, Carrinho, Pedidos”** onde todos os dados da loja foram armazenados.

***CREATE DATABASE argentstore\_db;***

***USE argentstore\_db;***

***CREATE TABLE registro (***

***id INT AUTO\_INCREMENT PRIMARY KEY,***

***nome VARCHAR(255),***

***sobrenome VARCHAR(255),***

***email VARCHAR(255) UNIQUE,***

***telefone CHAR(25),***

***data\_nascimento DATE,***

***cep VARCHAR(255),***

***endereco varchar(255),***

***senha varchar(255),***

***reset\_password\_token VARCHAR(255),***

***reset\_password\_expires DATETIME NULL***

***);***

***CREATE TABLE produtos (***

***id INT AUTO\_INCREMENT PRIMARY KEY,***

***descricao VARCHAR(255),***

***categoria VARCHAR(255),***

***preco\_venda DECIMAL(10, 2)***

***);***

***CREATE TABLE estoque (***

***id INT AUTO\_INCREMENT PRIMARY KEY,***

***quant\_estoque INT,***

***data\_entrada varchar(15),***

***fornecedor varchar(50),***

***produtos\_id int,***

***FOREIGN KEY (produtos\_id) REFERENCES produtos(id));***

***CREATE TABLE carrinho (***

***id INT AUTO\_INCREMENT PRIMARY KEY,***

***data\_compra varchar(15),***

***forma\_pagamento varchar(20),***

***quantidade int,***

***total\_compra DECIMAL(10, 2),***

***registro\_id int,***

***FOREIGN KEY (registro\_id) REFERENCES registro(id)***

***);***

***CREATE TABLE pedidos (***

***id INT AUTO\_INCREMENT PRIMARY KEY,***

***data\_pagamento date,***

***status\_pedido varchar(50),***

***carrinho\_id int,***

***FOREIGN KEY (carrinho\_id) REFERENCES carrinho(id),***

***produtos\_id int,***

***FOREIGN KEY (produtos\_id) REFERENCES produtos(id)***

***);***

1. **CONFIGURAÇÃO D BANCO DE DADOS**

Após instalar a biblioteca do msql2 para conectar os dados ao MySQL com o comando no terminal **npm install mysql2** criamos uma pasta “config” que armazenou o arquivo de configuração da conexão ao banco de dados, dentro da pasta criamos o arquivo “db.js” onde conectamos a lógica ao banco de dados MySQL.

***//Esta pasta armazena a logica a ser conectada ao banco de dados***

***const mysql = require('mysql2') //Importa a biblioteca mysql2 anteriormente instalada e cria a conexão com o banco de dados***

***//Variaveis de ambiente***

***console.log('DB\_HOST:', process.env.DB\_HOST);***

***console.log('DB\_USER:', process.env.DB\_USER);***

***console.log('DB\_PASS:', process.env.DB\_PASS);***

***console.log('DB\_NAME:', process.env.DB\_NAME);***

***const db = mysql.createConnection({ // Função cria uma conexão com db de acordo com as configurações fornecidas***

***host:process.env.DB\_HOST, // Endereço do servidor***

***user:process.env.DB\_USER, // Nome do usuario***

***password:process.env.DB\_PASS, // Senha do usuario***

***database:process.env.DB\_NAME // Nome do banco de dados***

***});***

***//Conectando o banco de dados e exportando a conexão***

***db.connect((err) =>{***

***if (err){***

***console.error('Erro ao conectar ao banco de dados', err); // Exibição da mensagem de erro***

***return;***

***}***

***console.log(`Conectado ao db no MySQL ${process.env.DB\_NAME}`);***

***});***

***module.exports= db; // Exporta a conexão para ser usada posteriomente em outros arquivos***

1. **CONTROLADORES**

Controladores são os responsáveis por manipular os dados, requisições e respostas HTPP, cada controlador teve a função especificar a aplicação como adicionar, inserir, atualizar completamente e parcialmente e deletar **(CRUD)** que um acrônimo para create, read, update e delete, operações básicas relacionadas ao banco de dados da aplicação. **Create (POST)** requisição para adicionar novas informações, **Read (GET)** requisição para buscar informações da api, Update **(Put ou Patch)** requisição para atualizar completamente ou parcialmente as informações e Deletar **(Delete)** requisição que deleta as informações que não serão mais utilizadas.

Através dessas operações básicas de manipulação de dados foi possível organizar os dados da aplicação, como o registro, produtos, estoque, pedidos e carrinho dessa maneira a manutenção do código ficou mais prática e compreenssivel .

* Arquivo de Controller de Produtos

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

***// Função para obter todas as transações***

***const getAllProdutos = (require, res) => {***

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

***if (err) {***

***console.error('Erro ao obter todos os produtos do estoque:', err);***

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

***return;***

***}***

***res.json(results);***

***});***

***};***

***//Função para adicionar uma nova transação (Com verificação de Duplicidade)***

***const addProdutos = (req, res) => {***

***const {descricao,categoria, preco\_venda} = req.body;***

***//Verificar se a transação já existe***

***db.query(***

***'SELECT \* FROM produtos WHERE descricao=? AND categoria=? AND preco\_venda=?',***

***[descricao,categoria, preco\_venda],***

***(err, results) => {***

***if (err) {***

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

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

***return;***

***}***

***if (results.length > 0) {***

***//se a transação já existe***

***res.status(400).send('Produto duplicado')***

***}***

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

***db.query(***

***'INSERT INTO produtos (descricao,categoria, preco\_venda) VALUES (?, ?, ?)',***

***[descricao,categoria, preco],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao adicionar novo produto', err);***

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

***return;***

***}***

***res.status(201).send('Novo produto adicionado');***

***}***

***);***

***}***

***);***

***};***

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

***const putProdutos = (req, res) => {***

***const { id } = req.params;***

***const {descricao,categoria, preco\_venda} = req.body;***

***db.query(***

***'UPDATE produtos SET descricao=?, categoria=?, preco\_venda=? WHERE id=?',***

***[descricao,categoria, preco\_venda, id],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao subestituir produto ', err);***

***res.status(500).send('Erro ao substituir produto');***

***return;***

***}***

***res.send('Dados do produto atualizado com sucesso');***

***}***

***);***

***};***

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

***const updateProdutos = (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 produtos SET ${query.join(', ')} WHERE id = ?`, values,***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao atualizar dado do produto:', err);***

***res.status(500).send('Erro ao atualizar dado do produto');***

***return;***

***}***

***res.send(' Dados do produto atualizados com sucesso');***

***}***

***);***

***};***

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

***const deleteProdutos= (req, res) => {***

***const { id } = req.params;***

***db.query('DELETE FROM produtos WHERE id = ?', [id], (err, results) => {***

***if (err) {***

***console.error('Erro ao deletar produto:', err);***

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

***return;***

***}***

***res.send('Produto deletado com sucesso');***

***});***

***};***

***module.exports = {***

***getAllProdutos,***

***addProdutos,***

***putProdutos,***

***updateProdutos,***

***deleteProdutos***

***};***

* Arquivo de Controller de Estoque

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

***// Função para obter todas as transações de estoque***

***const getAllEstoque = (require, res) => {***

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

***if (err) {***

***console.error('Erro ao obter todos os produtos do estoque:', err);***

***res.status(500).send('Erro ao obter produtos de estoque');***

***return;***

***}***

***res.json(results);***

***});***

***};***

***//Função para adicionar uma nova transação (Com verificação de Duplicidade)***

***const addEstoque = (req, res) => {***

***const { quant\_estoque,data\_entrada,fornecedor,produtos\_id } = req.body;***

***//Verificar se a transação já existe***

***db.query(***

***'SELECT \* FROM estoque WHERE quant\_estoque=? AND data\_entrada =? AND fornecedor =? AND produtos\_id =? ',***

***[quant\_estoque ,data\_entrada,fornecedor,produtos\_id ],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao adicionar produto ao estoque', err);***

***res.status(500).send('Erro ao adicionar produto ao estoque');***

***return;***

***}***

***if (results.length > 0) {***

***//se a transação já existe***

***res.status(400).send('Produto duplicado')***

***}***

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

***db.query(***

***'INSERT INTO estoque (quant\_estoque,data\_entrada,fornecedor,produtos\_id ) VALUES (?, ?, ?, ?)',***

***[quant\_estoque,data\_entrada,fornecedor,produtos\_id ],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao adicionar novo produto no estoque', err);***

***res.status(500).send('Erro ao adicionar novo produto no estoque');***

***return;***

***}***

***res.status(201).send('Novo produto adicionado ao estoque');***

***}***

***);***

***}***

***);***

***};***

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

***const putEstoque = (req, res) => {***

***const { id } = req.params;***

***const { quant\_estoque,data\_entrada,fornecedor,produtos\_id } = req.body;***

***db.query(***

***'UPDATE estoque SET quant\_estoque = ?, data\_entrada =?, fornecedor = ?, produtos\_id = ? WHERE id = ?',***

***[quant\_estoque,data\_entrada,fornecedor,produtos\_id, id],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao subestituir o produto do estoque', err);***

***res.status(500).send('Erro ao substituir o produto do estoque');***

***return;***

***}***

***res.send('Dados do produto atualizado com sucesso');***

***}***

***);***

***};***

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

***const updateEstoque = (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 estoque SET ${query.join(', ')} WHERE id = ?`, values,***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao atualizar dado do produto em estoque:', err);***

***res.status(500).send('Erro ao atualizar dado do produto em estoque');***

***return;***

***}***

***res.send(' Dados do produto atualizados com sucesso');***

***}***

***);***

***};***

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

***const deleteEstoque= (req, res) => {***

***const { id } = req.params;***

***db.query('DELETE FROM estoque WHERE id = ?', [id], (err, results) => {***

***if (err) {***

***console.error('Erro ao deletar produto do estoque:', err);***

***res.status(500).send('Erro ao deletar produto do estoque');***

***return;***

***}***

***res.send('Produto do estoque deletado com sucesso');***

***});***

***};***

***module.exports = {***

***getAllEstoque,***

***addEstoque,***

***putEstoque,***

***updateEstoque,***

***deleteEstoque***

***};***

* Arquivo de Controller de Pedidos

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

***// Função para obter todas as transações***

***const getAllPedidos = (require, res) => {***

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

***if (err) {***

***console.error('Erro ao obter todas as pedidos:', err);***

***res.status(500).send('Erro ao obter todas as pedidos');***

***return;***

***}***

***res.json(results);***

***});***

***};***

***//Função para adicionar uma nova transação (Com verificação de Duplicidade)***

***const addPedidos = (req, res) => {***

***const {data\_pagamento,status\_pedido ,carrinho\_id,produtos\_id} = req.body;***

***//Verificar se a transação já existe***

***db.query(***

***'SELECT \* FROM pedidos WHERE data\_pagamento=? AND status\_pedido=? AND carrinho\_id=? AND produtos\_id= ?',***

***[data\_pagamento,status\_pedido ,carrinho\_id,produtos\_id],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao adicionar novo pedido', err);***

***res.status(500).send('Erro ao adicionar novo pedido');***

***return;***

***}***

***if (results.length > 0) {***

***//se a transação já existe***

***res.status(400).send('PEDIDO DUPLICADO')***

***}***

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

***db.query(***

***'INSERT INTO pedidos (data\_pagamento,status\_pedido ,carrinho\_id,produtos\_id) VALUES (?, ?, ?, ? )',***

***[data\_pagamento,status\_pedido ,carrinho\_id,produtos\_id],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao adicionar novo pedido', err);***

***res.status(500).send('Erro ao adicionar novo pedido');***

***return;***

***}***

***res.status(201).send('Novo pedido adicionado com sucesso!');***

***}***

***);***

***}***

***);***

***};***

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

***const putPedidos = (req, res) => {***

***const { id } = req.params;***

***const {data\_pagamento,status\_pedido ,carrinho\_id,produtos\_id} = req.body;***

***db.query(***

***'UPDATE pedidos SET data\_pagamento=?, status\_pedido=? ,carrinho\_id=?, produtos\_id=? WHERE id=?',***

***[data\_pagamento,status\_pedido ,carrinho\_id,produtos\_id,id],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao subestituir pedido ', err);***

***res.status(500).send('Erro ao substituir pedido'+ err);***

***return;***

***}***

***res.send('Dados do pedido atualizados com sucesso!');***

***}***

***);***

***};***

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

***const updatePedidos = (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 pedidos SET ${query.join(', ')} WHERE id = ?`, values,***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao atualizar pedido:', err);***

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

***return;***

***}***

***res.send('Dados do pedido atualizados com sucesso!');***

***}***

***);***

***};***

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

***const deletePedidos = (req, res) => {***

***const { id } = req.params;***

***db.query('DELETE FROM pedidos WHERE id = ?', [id], (err, results) => {***

***if (err) {***

***console.error('Erro ao deletar compra:', err);***

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

***return;***

***}***

***res.send('Compra deletada com sucesso');***

***});***

***};***

***module.exports = {***

***getAllPedidos,***

***addPedidos,***

***putPedidos,***

***updatePedidos,***

***deletePedidos***

***}***;

* Arquivo de Controller de Carrinho

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

***// Função para obter todas as transações***

***const getAllVenda = (require, res) => {***

***db.query("SELECT \* FROM carrinho", (err, results) => {***

***if (err) {***

***console.error("Erro ao obter todos os produtos do carrinho:", err);***

***res.status(500).send("Erro ao obter produtos");***

***return;***

***}***

***res.json(results);***

***});***

***};***

***//Função para adicionar produto ao carrinho***

***const addCarrinho = (req, res) => {***

***const {data\_compra,forma\_pagamento,quantidade,registro\_id } = req.body;***

***db.query(***

***"INSERT INTO carrinho (data\_compra,forma\_pagamento,quantidade,total\_compra,registro\_id ) VALUES (?, ?, ?, ?, ?)",***

***[data\_compra,forma\_pagamento,quantidade,registro\_id],***

***(err, results) => {***

***if (err) {***

***console.error("Erro ao adicionar produto ao carrinho:", err);***

***res.status(500).send("Erro ao adicionar produto ao carrinho");***

***return;***

***}***

***res.status(201).send("Produto adicionado ao carrinho");***

***}***

***);***

***};***

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

***const putCarrinho = (req, res) => {***

***const { id } = req.params;***

***const { data\_compra,forma\_pagamento,quantidade,registro\_id} = req.body;***

***db.query(***

***'UPDATE carrinho SET data\_compra=?,forma\_pagamento=?,quantidade=?,registro\_id=? WHERE id=?',***

***[data\_compra,forma\_pagamento,quantidade,registro\_id,id],***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao subestituir o produto no carrinho', err);***

***res.status(500).send('Erro ao substituir o produto no carrinho'+ err);***

***return;***

***}***

***res.send('Dados do produto atualizado com sucesso');***

***}***

***);***

***};***

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

***const updateCarrinho = (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 carrinho SET ${query.join(', ')} WHERE id = ?`, values,***

***(err, results) => {***

***if (err) {***

***console.error('Erro ao atualizar dado do produto no carrinho:', err);***

***res.status(500).send('Erro ao atualizar dado do produto no carrinho');***

***return;***

***}***

***res.send(' Dados do produto atualizados com sucesso');***

***}***

***);***

***};***

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

***const deleteCarrinho= (req, res) => {***

***const { id } = req.params;***

***db.query('DELETE FROM carrinho WHERE id = ?', [id], (err, results) => {***

***if (err) {***

***console.error('Erro ao deletar produto do carrinho:', err);***

***res.status(500).send('Erro ao deletar produto do carrinho');***

***return;***

***}***

***res.send('Produto do carrinho deletado com sucesso');***

***});***

***};***

***module.exports = {***

***getAllVenda,***

***addCarrinho,***

***putCarrinho,***

***updateCarrinho,***

***deleteCarrinho***

***};***

* **Autenticação da API**

Para garantir a segurança e integridade dos dados dos clientes e usuários da loja houve a necessidade da implementação da autenticação que é o processo de verificar a identidade de um usuário. Assim garantindo que usuários autorizados possam ter acesso aos recursos protegidos da aplicação. Para que isso fosse executado se fez necessária a instalação de dependências para a autenticação, foram eles:

**JSON Web Token (JWT)** é um padrão aberto que permite a transmissão de informações de forma segura entre partes, por meio de um objeto JSON. Os **JWTs** são usados para autenticação e autorização, e são comuns em aplicativos web e mobile e Bcrypt é um método de criptografia do tipo hash para senhas. Foi utilizado o seguinte comando no terminal da **aplicação npm install jsonwebtoken bcrypt.**

Para começar a configurar as rotas e controladores é de suma importância atualizar as variáveis de ambiente no arquivo .env com autenticação que nesse caso foi a implementação da chave secreta (JWT\_SECRET), que utilizada para assinar os tokens JWT. A chave secreta foi obtida através do comando que gerou uma string aleatória: **node -e “console.log(require(‘crypto).radomBytes(64).toString('hex'))"**

***const crypto = require('crypto')***

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

***const bcrypt = require("bcrypt"); // Importa o bcrypt para criptografar senhas dos usuarios***

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

***const sendEmail = require("../services/emailService").sendEmail;***

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

***const registerUser = async (req, res) => {***

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

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

***try {***

***const [existingUser] = await db.promise().query("SELECT \* FROM registro WHERE email = ?",***

***[email]);***

***if (existingUser.length > 0) {***

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

***}***

***// Criptografar a senha usando bcrypt***

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

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

***await db.promise().query(***

***"INSERT INTO registro ( nome,sobrenome,email,telefone,data\_nascimento, cep, endereco,senha) VALUES (?, ?, ?, ?, ?, ? ,?, ?)",***

***[ nome,sobrenome,email,telefone,data\_nascimento, cep, endereco,senha,hashedPassword ]***

***);***

***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");***

***}***

***};***

***// Função para autenticar um usuário***

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

***if (user.length === 0) {***

***return res.status(400).send("Credenciais inválidas (email inválido)");***

***}***

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

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

***if (!isMatch) {***

***return res.status(400).send(" Credenciais inválidas (senha inválida)");***

***}***

***// Gerar um token JWT***

***const token = jwt.sign({ userId: user[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");***

***}***

***};***

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

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

***const { email } = req.body;***

***try {***

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

***if (registro.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 registro SET reset\_password\_token = ?, reset\_password\_expires = ? WHERE email = ?",***

***[token, expireDate, email]***

***);***

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

***sendEmail(***

***email,***

***"Recuperação de Senha - Sistema da Argent Store",***

***`Por favor, clique no link para redefinir sua senha e continuar comprando: ${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");***

***}***

***};***

***// Função para redefinir a senha***

***const resetarSenha = async (req, res) => {***

***const { token, newPassword } = req.body;***

***try {***

***const [registro] = await db.promise().query("SELECT \* FROM registro WHERE reset\_password\_token = ? AND reset\_password\_expires > NOW()",***

***[token]***

***);***

***if (registro.length === 0) {***

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

***}***

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

***await db***

***.promise().query("UPDATE registro SET senha = ?, 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 = {***

***registerUser,***

***loginRegistro,***

***requestsenhaReset,***

***resetarSenha***

**};**

1. **ROTAS**

As rotas e endpoints são elementos fundamentais em uma API, pois definem os caminhos e métodos disponíveis para acessar recursos. Os endpoints são os URIs definidos nas rotas para os quais as requisições podem ser enviadas.

A pasta “routes” guardou os arquivos de definição de rotas da aplicação, já os arquivos dentro dessa pasta conteve as definições das rotas relacionadas ao registro, produtos, estoque, pedidos e carrinho além das rotas de autenticação utilizando métodos como o get, post, put, patch e delete.

* Arquivo de rotas de Autenticação de Registro

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.loginRegistro);***

***// Rota para solicitar redefinição de senha***

***router.post('/request-password-reset', authController.requestsenhaReset);***

***// Rota para redefinir a senha***

***router.post('/reset-password', authController.resetarSenha);***

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

* Arquivo de rotas de Produtos

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

***const router = express.Router(); // Cria um novo roteador que será usado para definie as rotas***

***const proddutosController = require('../controllers/produtosController.js'); // Importa o controlador de transações (estoqueController)***

***const authMiddleware = require('../middleware/authMiddleware.js') // Importa o middleware de autenticação***

***// Definindo uma rota para obter todas as transações***

***router.get('/', authMiddleware, proddutosController.getAllProdutos);***

***// Definindo uma rota para adicionar uma nova transação***

***router.post('/', authMiddleware, proddutosController.addProdutos);***

***//Rota para a atualização completa do produto***

***router.put('/:id', authMiddleware,proddutosController.putProdutos);***

***//Rota para a atualização parcial do produto***

***router.patch('/:id', authMiddleware, proddutosController.updateProdutos);***

***//Rota para deletar um produto***

***router.delete('/:id', authMiddleware, proddutosController.deleteProdutos);***

***// Exportando o roteador***

***module.exports = router***

* Arquivo de rotas de Estoque

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

***const router = express.Router(); // Cria um novo roteador que será usado para definie as rotas***

***const estoqueController = require('../controllers/estoqueController.js'); // Importa o controlador de transações (estoqueController)***

***const authMiddleware = require('../middleware/authMiddleware.js') // Importa o middleware de autenticação***

***// Definindo uma rota para obter todas as transações***

***router.get('/', authMiddleware, estoqueController.getAllEstoque);***

***// Definindo uma rota para adicionar uma nova transação***

***router.post('/', authMiddleware, estoqueController.addEstoque);***

***//Rota para a atualização completa do produto***

***router.put('/:id', authMiddleware, estoqueController.putEstoque);***

***//Rota para a atualização parcial do produto***

***router.patch('/:id', authMiddleware, estoqueController.updateEstoque);***

***//Rota para deletar um produto***

***router.delete('/:id', authMiddleware, estoqueController.deleteEstoque);***

***// Exportando o roteador***

***module.exports = router***

* Arquivo de rotas de Pedidos

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

***const router = express.Router(); // Cria um novo roteador que será usado para definie as rotas***

***const pedidosController = require('../controllers/pedidosController.js'); // Importa o controlador de transações (estoqueController)***

***const authMiddleware = require('../middleware/authMiddleware.js') // Importa o middleware de autenticação***

***// Definindo uma rota para obter todas as transações***

***router.get('/',authMiddleware, pedidosController.getAllPedidos);***

***// Definindo uma rota para adicionar uma nova transação***

***router.post('/',authMiddleware, pedidosController.addPedidos);***

***//Rota para a atualização completa do produto***

***router.put('/:id', authMiddleware, pedidosController.putPedidos);***

***//Rota para a atualização parcial do produto***

***router.patch('/:id', authMiddleware, pedidosController.updatePedidos);***

***//Rota para deletar um produto***

***router.delete('/:id', authMiddleware, pedidosController.deletePedidos);***

***// Exportando o roteador***

***module.exports = router***

* Arquivo de rotas de Carrinho

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

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

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

***const authMiddleware = require('../middleware/authMiddleware.js') // Importa o middleware de autenticação***

***// Definindo uma rota para ver se há uma venda***

***router.get('/', authMiddleware, carrinhoController.getAllVenda);***

***// Definindo uma rota para adicionar uma nova venda***

***router.post('/', authMiddleware, carrinhoController.addCarrinho);***

***//Rota para a atualização completa do produto***

***router.put('/:id', authMiddleware, carrinhoController.putCarrinho);***

***//Rota para a atualização parcial do produto***

***router.patch('/:id', authMiddleware, carrinhoController.updateCarrinho);***

***//Rota para deletar um produto***

***router.delete('/:id', authMiddleware, carrinhoController.deleteCarrinho);***

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

1. **MIDDLEWARES**

Para proteger as rotas utilizamos middleware de autenticação são uma camada de software que atua como intermediária entre um cliente e um servidor, facilitando o processo de autenticação de usuários em aplicações web. Essa abordagem é essencial para garantir que apenas usuários autorizados tenham acesso a recursos e dados da aplicação. Criamos um arquivo assim JavaSript de middleware para verificar se o token JWT está presente e válido. Adicionamos assim os o middlewares de autenticação **(authMiddleware)** a todas as rotas de presentes, garantindo que apenas usuários autenticados possam acessar essas rotas.

***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); // Verifica a validade do 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***

1. **SERVICES**

Implementar uma recuperação de senha é essencial para a complexidade de uma aplicação. Para que isso fosse possível foi adicionado as colunas de **reset\_password\_token e reset\_password\_expires** na tabela registro e atualizar o arquivo **.env** para para armazenar informações sensíveis, como credenciais de e-mail. O comando **npm install nodemailer** foi executado pois houve a necessidade de implementar uma biblioteca para enviar os emails de recuperação para só assim configurar os servises.

***const nodemailer = require('nodemailer');***

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

***const transporter = nodemailer.createTransport({***

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

***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);***

***});***

***};***

***module.exports = { sendEmail };***

1. **SERVIDOR**

Um servidor no Node.js é um programa executado em um servidor é capaz de aceitar solicitações HTTP e enviar respostas além de realizar tarefas de processamento de dados, configurando e inicializando o express e definir rotas para a aplicação iniciar o servidor na porta específica, nesse caso na porta 5000 nas variáveis de ambiente.

Para configurar o servidor express importamos as bibliotecas, express, dotenv, cors e bobyParser que são ultilizadas para criar e gerenciar servidores web e APIs, gerenciar variáveis de ambiente, como credenciais de banco de dados. O cors permite que o servidor aceite requisições de diferentes origens, o pacote bobyParser utilizado para analisar o corpo das requisições HTTP, permitindo que o servidor manipule dados enviados pelo cliente.

Para fazer testar o servidor definimos uma rota inicial com o get para a URL raiz (/). Quando um cliente faz uma requisição GET para a URL raiz, o servidor responde a requisição com a mensagem " Servidor está funcionando corretamente".

Configuramos a porta 3000 como padrão que o servidor vai executar a partir da variável de ambiente. No “app.listen” estamos configurando o servidor para escutar na porta especificada. Quando o servidor começa a escutar na porta, exibimos uma mensagem no console indicando que o servidor está rodando. Ao executar o comando **npm start** no terminal recebemos a mensagem do servidor “O servidor está rodando na porta 5000”

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

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

***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***

***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 transações e autenticação***

***const produtosRoutes = require('./routes/produtos'); // Importa as rotas de produtos***

***const authRoutes = require('./routes/auth'); // Importa as rotas de autenticação***

***const carrinhoRoutes = require('./routes/carrinho'); // Importa as rotas do carrinho***

***const estoqueRoutes = require('./routes/estoque'); // Importa as rotas de estoque***

***const pedidosRoutes = require ('./routes/pedidos'); //Importa as rotas de compras***

***// Usar as rotas de transações e autenticação para as requisições***

***app.use('/api/produtos', produtosRoutes); // Usar as rotas de transações para todas as requisições que começam com /api/transactions***

***app.use('/api/auth', authRoutes); // Configura o servidor para usar as rotas de autenticação***

***app.use('/api/carrinho', carrinhoRoutes); // Configura o servidor para usar as rotas do carrinho***

***app.use('/api/estoque', estoqueRoutes); // Configura o servidor para usar as rotas do estoque***

***app.use('/api/pedidos',pedidosRoutes); // Configura o servidor para usar as rotas das compras***

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

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

***res.send('Servidor está funcionando corretamente');***

***});***

***// Define a porta a par r da variável de ambiente ou usa a porta 5000 como padrão***

***const PORT = process.env.Port || 5000;***

***app.listen(PORT, ()=>{***

***console.log(`Servidor esta rodando na PORTA ${PORT}`);***

***});***

1. **TESTES**

Para verificar se as requisições da API obtiveram a resposta esperada houve a necessidade de testes e eles foram feitos através do aplicativo **insomnia** onde as respostas de sucesso e erro foram exibidas.

1. **CONSIDERAÇÕES FINAIS**