**Documentação Técnica da Api (Inventário)**

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.

Os dados que foram integrados nesse projeto de inventário por meio da API são:

* **Id**
* **Nome do produto (Name)**
* **Descrição do produto (Description)**
* **Categoria do produto (Category)**
* **Preço do produto (Price)**
* **Quantidade em estoque (Stock)**
* **Data de validade (Expiry date)**

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 “apiInventory” 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 , permindo que você execute código JavaScript fora do navegador. Já o **npm**  é um gerenciador de pacotes do **Noje.js** ele nos permite atualizar e remover bibliotecas e ferramentas adicionais, facilitando a gestão de dependências da aplicação.

Para inicia-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 facial) e fermentas necessárias para o projeto como **cors, boby-parser, doteenv, nodemon e mysql2** elas auxiliaram na criação do servidor, analisar o corpo das requisições HTPP, 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 a 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.

banco de dados - Pasta para armazenar o bando de dados

inventory\_db\_products.sql - Banco de dados

config - Pasta de configuração de dados

dbi.js - Arquivo js para conexão de dados

controllers - Pasta de controladores

productsController.js - Arquivo js controles de requisições dos products

routes - Pasta para rotas de api

products.js - Arquivo js para definição das rotas

.env - Arquivo para as variáveis de ambiente

Package.json - Arquivo que guarda as dependências, a versão, o autor

Server.js - Arquivo js do servidor

1. **CONFIGURAÇÃO DE AMBIENTE**

* **Variáveis de ambiente** (Arquivo .env)

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 = 230114*

*DB\_NAME = inventory\_db*

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 **“inventory\_db”** e tabela **”products”** onde os dados dos produtos do inventário foram armazenados.

*create database inventory\_db;*

*use inventory\_db;*

*create table products (*

*id INT AUTO\_INCREMENT PRIMARY KEY,*

*name VARCHAR(255),*

*description TEXT(255),*

*category VARCHAR(255),*

*price DECIMAL(10,2),*

*stock INT,*

*expiry\_date DATE*

*);*

**6. CONFIGURAÇÃO DO BANCO DE DADOS** (Pasta config e Arquivo dbi.js)

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 “dbi.js” onde conectamos a lógica ao banco de dados MySQL.

***// Importa o pacoteMysql2 para conectar ao banco de dados***

const mysql = require('mysql2');

*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);*

***// Função para criar conexão com o banco de dados***

*const dbi = mysql.createConnection({*

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

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

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

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

*});*

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

*if (err){*

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

*return;*

*}*

*console.log(`Conectando ao banco de dados MySQL ${process.env.DB\_NAME}`);*

*});*

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

**7. CONTROLADORES (**Pasta Controllers e Arquivo productscontoller)

Controladores são os responsáveis por manipular os dados, requisições e respostas HTPP, cada controlador teve a função especifica da aplicação como adicionar, inserir, atualizar completamente e parcialmente, deletar e listar os produtos do inventario, facilitando a organização e a manutenção do código.

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

***//Função para obter todos os produtos (get)***

const getAllProducts = (req,res) =>{

dbi.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);

});

};

***//Função para adicionar um novo produto (post)***

const addProducts = (req, res) => {

const { name, description, category, price, stock, expiry\_date } = req.body;

dbi.query(

'INSERT INTO products (name, description, category, price, stock, expiry\_date) VALUES (?, ?, ?, ?, ?, ? )',

[name, description, category, price, stock, expiry\_date],

(err, results) => {

if (err) {

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

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

return;

}

res.status(201).send('Transação adicionada com sucesso');

}

);

};

***//Função para atualizar completamente um produto (put)***

const updateProductsPut = (req,res) => {

const {id} = req.params;

const { name, description, category, price, stock, expiry\_date } = req.body;

dbi.query(

'UPDATE products SET name = ?, description = ?, category = ?, price = ?, stock = ?, expiry\_date = ? WHERE id = ?',

[id, name, description, category, price, stock, expiry\_date],

(err, results) =>{

if (err) {

console.error('Não foi possivel atualizar a atransação', err);

res.status(500).send('Erro! A transação não foi atualizada');

return;

}

res.send('Atualização concluida com sucesso!')

}

);

};

***//Função para atualizar parcialmente um produto (patch)***

const updateProductsPatch = (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);

dbi.query(

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

values,

(err,results) =>{

if(err){

console.error('Não foi possivel atualizar a transação', err);

res.status(500).send('Erro! Não foi possivel atualizar a transação');

return;

}

res.send('Atualização concluida com sucesso!');

}

);

};

***//Função para deletar um produto (delete)***

const deleteProducts = (req, res) => {

const {id} = req.params;

dbi.query('DELETE FROM products WHERE id = ?', [id], (err,results) =>{

if (err) {

console.error('Erro! Não foi possivel deletar a atransação', err);

res.status(500).send('Erro! Não foi possivel deletar a atransação');

return;

}

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

});

}

***//Exporta todas as funções para que possam ser ultilizadas em outros arquivos***

module.exports = {

getAllProducts,

addProducts,

updateProductsPut,

updateProductsPatch,

deleteProducts

};

**8. ROTAS** (Pasta routes e Arquivo products.js)

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 o arquivo de definição de rotas da aplicação, já o arquivo “products.js” conteve as definiçoes das rotas relacionadas aos produtos, ultilizando métodos como o get, post, put, patch e delete.

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

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

const productsController = require ('../controllers/productsController'); ***// Importa o controlador das transações***

***//Rota para obter todos os produtos***

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

***//Rota para inserir novos produtos***

router.post('/',productsController.addProducts);

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

router.put('/:id',productsController.updateProductsPut);

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

router.patch('/:id',productsController.updateProductsPatch);

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

router.delete('/:id',productsController.deleteProducts);

***//Exportando o roteador***

module.exports = router;

**9. SERVIDOR**

Um servidor no Node.js é um programa executado em um servidor e 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 especifica, nesse caso na porta 3000 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 ".

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 3000”.

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

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

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

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

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

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

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

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

res.send('Servidor está funcionando');

});

***// Define a porta a par r da variável de ambiente ou usa a porta 3000 como padrão***

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

app.listen(PORT, ()=>{

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

});

**10. 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.

**Requisição (Get): Requisição para exibir todos os produtos.**

* Método: GET
* URL: http://localhost:3000/api/products

**Resposta de Sucesso (200):**

*[*

*{*

*"id": 1,*

*"name": "Shampoo de Flor de Lima",*

*"description": "Shampoo",*

*"category": "Cabelo",*

*"price": "47.90",*

*"stock": 8,*

*"expiry\_date": "2024-12-31T03:00:00.000Z"*

*},*

*{*

*"id": 2,*

*"name": "DESODORANTE - HEROS",*

*"description": "DESODORANTE - HEROS",*

*"category": "Corpo",*

*"price": "41.90",*

*"stock": 8,*

*"expiry\_date": "2025-02-24T03:00:00.000Z"*

*},*

*{*

*"id": 3,*

*"name": "Creme de Barbear Black",*

*"description": "Creme de Barbear",*

*"category": "Barba",*

*"price": "22.90",*

*"stock": 6,*

*"expiry\_date": "2025-03-18T03:00:00.000Z"*

*},*

*{*

*"id": 4,*

*"name": "Perfume para Interiores de Lavanda",*

*"description": " Perfume para Interiores",*

*"category": "Casa",*

*"price": "136.90",*

*"stock": 4,*

*"expiry\_date": "2025-04-13T03:00:00.000Z"*

*},*

*{*

*"id": 5,*

*"name": "Óleo de Copaíba",*

*"description": " Óleos Essenciais e Vegetais",*

*"category": "Corpo",*

*"price": "38.90",*

*"stock": 2,*

*"expiry\_date": "2025-05-10T03:00:00.000Z"*

*},*

*{*

*"id": 6,*

*"name": "Kit de Viagem Flor de Lima ",*

*"description": "Kit capilar",*

*"category": "Cabelo",*

*"price": "83.90",*

*"stock": 9,*

*"expiry\_date": "2025-06-11T03:00:00.000Z"*

*},*

*{*

*"id": 7,*

*"name": "Kit Glamour B-Tox",*

*"description": "Kit Capilar ",*

*"category": "Cabelo",*

*"price": "109.90",*

*"stock": 7,*

*"expiry\_date": "2025-07-17T03:00:00.000Z"*

*},*

*{*

*"id": 8,*

*"name": "Água de Beleza Rosa Mosqueta",*

*"description": "Água fascial",*

*"category": "Rosto",*

*"price": "45.90",*

*"stock": 10,*

*"expiry\_date": "2025-08-25T03:00:00.000Z"*

*}*

*]*

**Requisição e resultados (Post): Requisição para inserir um novo produto.**

* Método: Post
* URL: http://localhost:3000/api/products

**Resposta de Sucesso(201):**

{

"message": " Transação adicionada com sucesso"

}

**Requisição e resultados (Put): Requisição para atualização completa de um produto.**

* Método: Put
* URL http://localhost:3000/api/products/1
* Corpo (JSON):

{

"name": "Shampoo de Flor de Lima",

"description": "Shampoo Sólido",

"category": "Cabelo",

"price": 47.90,

"stock": "8",

"expiry\_date": "20250131"

}

**Resposta de Sucesso(200):**

{

"message": " Atualização concluida com sucesso!"

}

**Requisição e resultados (Patch): Requisição para atualização parcial de um produto.**

* Método: Patch
* URL: http://localhost:3000/api/products/1
* Corpo:

{

"name": "Shampoo de Flor de Lima",

"description": "Shampoo",

"category": "Cabelo",

"price": 47.90,

"stock": "3",

"expiry\_date": "20241231"

}

***Houve a atualização da quantidade em estoque.***

**Resposta de Sucesso(200):**

{

"message": " Atualização concluída com sucesso!"

}

**Requisição e resultados (Delete): Requisição para deletar um produto da lista.**

* Método: Delete
* URL: http://localhost:3000/api/products/8

**Resposta de Sucesso:**

{

"message": " Produto deletado com sucesso"

}

**Resposta de erro obtidas:**

**Erro 500 (Internal server Error):** Erro de sintaxe no arquivo “productscontrolers.js” na requisição patch.

TypeError: values.push is not a function  
    at updateProductsPatch (C:\Users\NaryDiniz T.I\Documents\-apiInventory\controllers\productsController.js:56:16)  
    at Layer.handle [as handle\_request] (C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\layer.js:95:5)  
    at next (C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\route.js:149:13)  
    at Route.dispatch (C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\route.js:119:3)  
    at Layer.handle [as handle\_request] (C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\layer.js:95:5)  
    at C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\index.js:284:15  
    at param (C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\index.js:365:14)  
    at param (C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\index.js:376:14)  
    at Function.process\_params (C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\index.js:421:3)  
    at next (C:\Users\NaryDiniz T.I\Documents\-apiInventory\node\_modules\express\lib\router\index.js:280:10)

**11. Considerações finais**

Depois de testar e corrigir erros no servidor com o uso do Insomnia, podemos perceber que as respostas das requisições da aplicação estão respondendo com sucesso as funções de listar, inserir, atualizar e deletar produtos do inventário da loja.