Atividade proposta – Construção de API

Nome: Victor Freitas Garces

**Introdução ao desenvolvimento de API**

A empresa “Casa da tecnologia”, que oferece produtos na área da tecnologia, deseja a construção de uma aplicação API REST interligada com banco de dados. Tal estrutura deve ser capaz de adicionar, listar, atualizar e deletar informações e dados dentro do banco de dados da instituição.

**Estrutura do projeto**

A estrutura do projeto tem como objetivo lista como a API está modelada, quais as pastas utilizadas e como se deu a construção da aplicação.

Atividade-proposta-desenvolvimentoAPI

|\_ config

|\_.env

|\_Controllers

|\_controllersRouters

|\_database

|\_inventory\_db

|\_.gitignore

|\_.env

\_\_Serve.js

\_\_Package

\_\_Package-lock

**Configurações de ambiente**

1. Variáveis de ambiente

As variáveis de ambiente são aquelas usadas para configuração do banco de dados, que permite a conexão e a interligação com o mesmo.

DB\_HOST=localhost

DB\_USERNAME=root

DB\_PASSWORD=07112001

DB\_NAME=inventory\_db

PORT=5000

1. Dependências

As dependências se referem a extensões e bibliotecas que foram utilizadas e são essenciais para a construção da aplicação que interliga as ações do usuário com movimentações fazendo uso das requisições HTTP. Desta forma teremos:

npm init -y

npm install cors body-parser dotenv express mysql2

npm install nodemon –save--dev

**Banco de dados**

Para termos uma aplicação integrada com banco de dados, faremos uso do banco de dados MySql, permitindo o gerenciamento de informações e características a respeitos dos produtos e serviços ofertados pela empresa demandante. Na estruturação do banco de dados teremos apenas a tabela de “products”, na seguinte estrutura:

create database inventory\_db;

USE inventory\_db;

-- criando a tabela products

CREATE TABLE products (

id\_product int(5) primary key auto\_increment,

name\_product varchar(255),

description\_product text,

category varchar(100),

price\_product decimal(5,2),

quant\_product int,

data\_validade date

);

-- inserindo produtos na tabela products

INSERT INTO products (name\_product, description\_product, category, price\_product, quant\_product, data\_validade) VALUES

('Notebook', 'Notebook Dell Inspiron 15 3000', 'Eletrônicos', 250.00, 10, '2025-12-31'),

('Smartphone', 'Smartphone Samsung Galaxy S21', 'Eletrônicos', 350.00, 20, '2025-12-31'),

('Geladeira', 'Geladeira Brastemp Frost Free', 'Eletrodomésticos', 450.00, 5, '2028-06-30'),

('Microondas', 'Microondas Electrolux 30L', 'Eletrodomésticos', 800.00, 15, '2027-08-15'),

('Cadeira Gamer', 'Cadeira Gamer DXRacer', 'Móveis', 150.00, 25, '2030-05-20'),

('Mesa de Escritório', 'Mesa de Escritório em L', 'Móveis', 700.00, 8, '2030-05-20'),

('Livro', 'Livro - Clean Code', 'Livros', 120.00, 50, '2050-01-01'),

('Caderno', 'Caderno Universitário 200 folhas', 'Papelaria', 20.00, 100, '2030-12-31'),

('Mouse', 'Mouse Gamer Logitech G502', 'Acessórios de Computador', 250.00, 30, '2029-11-11'),

('Teclado', 'Teclado Mecânico Razer', 'Acessórios de Computador', 500.00, 20, '2029-11-11');

**Configurações do projeto**

* **Configuração do banco de dados**

Na configuração do banco de dados, teremos o arquivo db.js sendo responsável por arquitetar e permitir a interligação entre o banco de dados (inventory\_db) com a API.

//INICIANDO CONFIGURAÇÃO DO BANCO DE DADOS

    //Importando bibliotecas

    const mysql2 = require('mysql2');

    //configurando conexão

    const db = mysql2.createConnection({

        host: process.env.DB\_HOST,

        user: process.env.DB\_USERNAME,

        password: process.env.DB\_PASSWORD,

        database: process.env.DB\_NAME

    }

    )

    db.connect ((err) => {

        if(err) {

            console.log('Erro ao conectar', err)

        return};

        console.log(`Conectado com sucesso ao banco de dados ${process.env.DB\_NAME}`)

    });

    //exportando a conexão do banco de dados

    module.exports=db;

* **Controlando as rotas(controllerRouter)**

controllerRouter.js representa a parcela da aplicação responsável por receber e interpretar as requisições HTTP, e devolver as respostas destas requisições, permitindo a integração completa da aplicação. Assim, se torna possível adicionar, alterar, atualizar e deletar dados no banco de dados conectado previamente.

//CONFIGURANDO ROTAS DA API

    //IMPORTANDO CONEXÃO COM BANCO DE DADOS

    const { configDotenv } = require('dotenv');

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

    //configurando rota para puxar todas as movimentações

    const getAllProducts = (req, res) => {

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

            if(err) {

                console.error('Erro ao visualizar produtos', err);

                res.status(500).send('erro ao visualizar produtos');

            return;

            }

            res.json(results);

        })

    };

\*a requisição GET tem como objetivo retornar todos os dados gravados no banco de dados ao usuário, a fim de que este possa visualizá-los. Desta forma, teremos um lógica que envia o comando sql “SELECT\* FROM PRODUCTS” que permite visualizar todas as inserções na tabela products.

//Configurando rota para adicionar produtos na tabela

    const AddProducts = (req,res) => {

        const {name\_product, description\_product, category, price\_product, quant\_product, data\_validade} = req.body;

        db.query('INSERT INTO products(name\_product, description\_product, category, price\_product, quant\_product, data\_validade) VALUES (?,?,?,?,?,?)', [name\_product, description\_product, category, price\_product, quant\_product, data\_validade],

            (err, results) => {

                if(err) {

                    console.error('Erro ao adicionar produto. Tente novamente', err)

                    console.log('ERRO AO ADICIONAR PRODUTO', err)

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

                return;

                }

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

            }

        )};

\*a requisição POST tem como objetivo enviar dados ao banco de dados conectado, a fim de estes sejam lidos e gravados na tabela products. Desta forma, fazemos uso da lógica inserindo o produto com a linguagem sql “INSERT INTO <table> VALUES<valores>”

        /\*CONFIGURANDO ROTA PARA ATUALIZAR COMPLETAMENTE O INVENTÁRIO(PUT)\*/

        const UpdateProductsPut = (req, res) => {

            const {id\_product} = req.params;

            const {name\_product, description\_product, category, price\_product, quant\_product, data\_validade} = req.body;

            db.query('UPDATE products SET name\_product=?, description\_product=?, category=?, price\_product=?, quant\_product=?, data\_validade=? WHERE id\_product=?',

                [name\_product, description\_product, category, price\_product, quant\_product, data\_validade, id\_product],

                (err, results) => {

                if(err) {

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

                    res.status(500).send('Erro! Não foi possível atualizar inventário');

                return;

                }

                res.send('Atualização concluída com sucesso')

                });

        }

\*A requisição PUT tem objetivo de atualizar informações a respeitos de produtos inseridos no banco de dados. Assim, faz uso do comando sql “UPDATE <table> Where <column\_table>....”

/\*CONFIGURANDO ROTA PARA ATUALIZAR PARCIALMENTE(PATCH)\*/

        const UpdateProductsPatch = (req, res) => {

            const {id\_product} = 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\_product);

            db.query(

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

                values,

                (err, results) => {

                    if(err) {

                        console.error('ERRO AO ATUALIZAR O INVENTÁRIO!', err);

                        res.status(500).send('ERRO. Não foi possível atualizar o inventário');

                    return;

                    }

                    res.send('Inventário atualizado com sucesso!');

                }

            )

        };

\*A requisição PATCH tem a intenção de atualizar informações específicas a respeito de algum produto no banco de dados.

/\*CONFIGURANDO ROTA PARA DELETAR O PRODUTO DO INVENTÁRIO\*/

        const DeleteProducts = (req, res) => {

            const {id\_product} = req.params;

                db.query ('DELETE FROM products WHERE id\_product = ?', [id\_product],

                    (err, results) => {

                    if(err) {

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

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

                    return;

                    }

                    res.status(201).send('TRANSAÇÃO DELETADA COM SUCESSO');

                    }

                );

            };

    //exportando rota

    module.exports = {

        getAllProducts,

        AddProducts,

        UpdateProductsPut,

        UpdateProductsPatch,

        DeleteProducts

    };

* **Criando as rotas**

Neste arquivo temos a criação das rotas que corresponde a cada requisição HTTP e a resposta que tal irá gera ao usuário conectado.

//ARQUIVO PARA AS ROTAS DA API

    //importando configurações dos controladores das rotas

    const express = require('express');

    const router = express.Router();

    const controllerRouter = require('../Controllers/controllerRouter');

    //Criando rota que permite visualizar o inventário

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

    /\*Criando rota para adicionar produtos ao inventário\*/

    router.post('/', controllerRouter.AddProducts);

    /\*CRIANDO ROTA PARA ATUALIZAR PRODUTOS(PUT)\*/

    router.put('/:id\_product', controllerRouter.UpdateProductsPut);

    /\*CRIANDO ROTA PARA ATUALIZAR PARCIALMENTE(PATCH)\*/

    router.patch('/:id\_product', controllerRouter.UpdateProductsPatch)

    /\*criando rota para deletar produto no inventário(delete)\*/

    router.delete('/:id\_product', controllerRouter.DeleteProducts);

    /\* exportando rotas \*/

    module.exports = router;

* **Servidor(server.js)**

A configuração do servidor é fundamental para o funcionamento da aplicação. Neste caso, usamos a biblioteca ‘express’, ferramenta do node que permite a criação e o funcionamento de um servidor local.

//INICIANDO SERVIDOR PARA API

const dotenv = require('dotenv');

dotenv.config();

//Chamando bibliotecas

const express = require('express');

const cors = require('cors');

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

//Conectando com banco de dados

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

//Configurando express

const inventory\_router = require('./Routers/router');

const app = express();

app.use(cors());

app.use(bodyParser.json());

//Permitindo o uso das rotas

app.use('/api/inventory', inventory\_router,)

//Configurações para rodar servidor

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

    res.send('Servidor está online');

});

const PORT\_SERVER = process.env.PORT\_SERVER || 5000;

app.listen(PORT\_SERVER, () => {

    console.log(`SERVIDOR ONLINE. CONECTADO ${PORT\_SERVER}`);

});

**Teste da API**

Para testar se a aplicação está rodando com eficiência e eficácia, realizamos alguns testes, fazendo uso do software Insomnia, que permite a ligação das requisições HTTP com o servidor criado.

1. Visualizar produtos no inventário

Nesta etapa, temos uma requisição HTTP para visualizar todos os produtos que se encontram no inventário. Assim teremos

Método: GET

URL: <http://localhost:5000/api/inventory>

Corpo:

[

{

"id\_product": 31,

"name\_product": "Geladeira",

"description\_product": "Notebook Dell Inspiron 15 3000",

"category": "Eletrônicos",

"price\_product": "2500.00",

"quant\_product": 10,

"data\_validade": "2025-12-31T03:00:00.000Z"

},

{

"id\_product": 32,

"name\_product": "Smartphone",

"description\_product": "Smartphone Samsung Galaxy S21",

"category": "Eletrônicos",

"price\_product": "350.00",

"quant\_product": 20,

"data\_validade": "2025-12-31T03:00:00.000Z"

},

{

"id\_product": 33,

"name\_product": "Geladeira",

"description\_product": "Geladeira Brastemp Frost Free",

"category": "Eletrodomésticos",

"price\_product": "450.00",

"quant\_product": 5,

"data\_validade": "2028-06-30T03:00:00.000Z"

},

{

"id\_product": 34,

"name\_product": "Microondas",

"description\_product": "Microondas Electrolux 30L",

"category": "Eletrodomésticos",

"price\_product": "800.00",

"quant\_product": 15,

"data\_validade": "2027-08-15T03:00:00.000Z"

},

{

"id\_product": 35,

"name\_product": "Cadeira Gamer",

"description\_product": "Cadeira Gamer DXRacer",

"category": "Móveis",

"price\_product": "150.00",

"quant\_product": 25,

"data\_validade": "2030-05-20T03:00:00.000Z"

},

{

"id\_product": 36,

"name\_product": "Mesa de Escritório",

"description\_product": "Mesa de Escritório em L",

"category": "Móveis",

"price\_product": "700.00",

"quant\_product": 8,

"data\_validade": "2030-05-20T03:00:00.000Z"

},

{

"id\_product": 37,

"name\_product": "Livro",

"description\_product": "Livro - Clean Code",

"category": "Livros",

"price\_product": "120.00",

"quant\_product": 50,

"data\_validade": "2050-01-01T03:00:00.000Z"

},

{

"id\_product": 38,

"name\_product": "Caderno",

"description\_product": "Caderno Universitário 200 folhas",

"category": "Papelaria",

"price\_product": "20.00",

"quant\_product": 100,

"data\_validade": "2030-12-31T03:00:00.000Z"

},

{

"id\_product": 39,

"name\_product": "Mouse",

"description\_product": "Mouse Gamer Logitech G502",

"category": "Acessórios de Computador",

"price\_product": "250.00",

"quant\_product": 30,

"data\_validade": "2029-11-11T03:00:00.000Z"

},

{

"id\_product": 40,

"name\_product": "Teclado",

"description\_product": "Teclado Mecânico Razer",

"category": "Acessórios de Computador",

"price\_product": "500.00",

"quant\_product": 20,

"data\_validade": "2029-11-11T03:00:00.000Z"

},

{

"id\_product": 41,

"name\_product": "Tablet",

"description\_product": "Tablet Apple iPad Air",

"category": "Eletrônicos",

"price\_product": "3200.00",

"quant\_product": 12,

"data\_validade": "2026-12-31T03:00:00.000Z"

}

]

1. Adicionar produto ao inventário

A adição de produtos é fundamental em uma aplicação, em que neste caso usamos a requisição HTTP POST, gravando no banco de dados novos produtos.

Método: POST

URL: <http://localhost:5000/api/inventory>

Corpo:

Resposta correta (200) {

Produto adicionado com sucesso!

};

Resposta incorreta (404) {

Não foi possível adicionar produto! Tente novamente.

};

1. Atualizar todas as informações do produto no inventário

Nesta requisição, o objetivo é atualizar informações a respeito de um produto, fazendo uso do método PUT, em que temos a atualização completa de todas as informações de determinado produto.

Método: PUT

URL: <http://localhost:5000/api/inventory/5>

Corpo:

{

"name\_product": "caderno tilibra",

"description\_product": "caderno tilibra 10 matérias" ,

"category": "papelaria",

"price\_product": 0,

"quant\_product": 18,

"data\_validade": "2024-08-20"

};

Resposta correta (200): {

Inventário atualizado com sucesso

};

Resposta incorreta (404): {

ERRO ao atualizar inventário.

};

1. Atualizar informações específicas do produto no inventário

Nesta fazemos uso do método PATCH, utilizado para atualizar informações específicas do produto.

Método: PATCH

URL: <http://localhost:5000/api/inventory/5>

Corpo:

{

"name\_product": "Geladeira",

"price\_product": 2500.00

};

Resposta correta (200): {  
Produto atualizado com sucesso!

};

Resposta incorreta (404): {

ERRO AO ATUALIZAR PRODUTO NO INVENTÁRIO!

};

1. Deletar produto do inventário

Para remover (ou deletar) um produto do inventário fazemos uso do método DELETE.

Método: DELETE

URL: <http://localhost:5000/api/inventory/8>

Corpo:

Resposta correta (201): {

TRANSAÇÃO DELETADA COM SUCESSO!

};

Resposta incorreta (404): {

Erro! Não foi possível deletar produto do inventário!

};

\*Possíveis erros que podem vir a ocorrer:

404 – Erro que ocorre quando a aplicação não consegue retornar a resposta da requisição. Por exemplo, no caso de uma atualização completa (PUT), pode vim ocorrer caso o dado(produto) não seja encontrado para ocorra a resposta desejada.

500 – Erro que corresponde a inconsistências encontradas no servidor, algum obstáculo na conexão ou configuração das rotas, impossibilitando que a requisição seja executada corretamente e o servidor dando erro