**ESTRUTUTRA DO PROJETO PDV**

|desafio-backend-final-dds-t16/

**|PDV/**

**|SRC/**

**|config/**

**|app.js**

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

*const routes =* ***require****('../routs/routes')*

*const app =* ***express****();*

*app.****use****(express.****json****());*

*app.****use****('/', routes)*

*module.exports = app;*

**|database.js**

*const { Pool } =* ***require****('pg');*

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

*dotenv.****config****();*

*const pool = new* ***Pool****({*

*user: process.env.DB\_USER,*

*host: process.env.DB\_HOST,*

*database: process.env.DB\_NAME,*

*password: process.env.DB\_PASS,*

*port: process.env.DB\_PORT,*

*});*

*pool.****on****('connect', () => {*

*console.****log****('Base de dados conectada.');*

*});*

*pool.****on****('error', (err) => {*

*console.****error****('Erro na base de dados:', err);*

*});*

*module.exports = pool;*

**|enviroment.js**

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

*dotenv.****config****();*

*module.exports = {*

*port: process.env.PORT || 3000,*

*jwtSecret: process.env.JWT\_SECRET,*

*};*

**|controllers/**

**|categoryController.js**

*const CategoryService =* ***require****('../services/categoryService');*

*class CategoryController {*

*async* ***listCategories****(req, res) {*

*try {*

*const categories = await CategoryService.****listCategories****();*

*res.****status****(200).****json****(categories);*

*} catch (error) {*

*res.****status****(500).****json****({ error: 'Erro ao listar categorias' });*

*}*

*}*

*async* ***createCategory****(req, res) {*

*try {*

*const category = await CategoryService.****createCategory****(req.body);*

*res.****status****(201).****json****(category);*

*} catch (error) {*

*res.****status****(400).****json****({ error: 'Erro ao criar categoria' });*

*}*

*}*

*}*

*module.exports = new* ***CategoryController****();*

**|pedidoController.js**

*const PedidoService =* ***require****('../services/pedidoService');*

*class PedidoController {*

*async* ***listPedidos****(req, res) {*

*try {*

*const pedidos = await PedidoService.****listPedidos****();*

*res.****json****(pedidos);*

*} catch (error) {*

*res.****status****(500).****json****({ error: 'Erro ao listar pedidos.' });*

*}*

*}*

*async* ***createPedido****(req, res) {*

*try {*

*const pedido = await PedidoService.****createPedido****(req.body);*

*res.****json****(pedido);*

*} catch (error) {*

*res.****status****(400).****json****({ error: 'Erro ao criar o pedido.' });*

*}*

*}*

*async* ***buscarPedidoPorId****(req, res) {*

*try {*

*const pedido = await PedidoService.****buscarPedidoPorId****(req.params.id);*

*if (!pedido) {*

*return res.****status****(404).****json****({ error: 'Pedido não encontrado.' });*

*}*

*res.****json****(pedido);*

*} catch (error) {*

*res.****status****(500).****json****({ error: 'Falha ao buscar pedido.' });*

*}*

*}*

*async* ***atualizarPedido****(req, res) {*

*try {*

*const pedido = await PedidoService.****atualizarPedido****(req.params.id, req.body);*

*if (!pedido) {*

*return res.****status****(404).****json****({ error: 'Pedido não encontrado.' });*

*}*

*res.****status****(peido);*

*} catch (error) {*

*res.****status****(400).****json****({ error: 'Erro ao atualizar o pedido.' });*

*}*

*}*

*async* ***deletarPedido****(req, res) {*

*try {*

*const result = await PedidoService.****deletarPedido****(req.params.id);*

*if (!result) {*

*return res.****status****(404).****json****({ error: 'Pedido não encontrado.' });*

*}*

*res.****json****({ message: 'Pedido deletado.' })*

*} catch (error) {*

*res.****status****(500).****json****({ error: 'Falha ao deletar o pedido.' })*

*}*

*}*

*}*

*module.exports = new* ***PedidoController****();*

**|produtoController.js**

*const ProdutoService =* ***require****('../services/produtoService');*

*class ProdutoController {*

*async* ***listProdutos****(req, res) {*

*try {*

*const produtos = await ProdutoService.****listProduto****();*

*res.****json****(produtos);*

*} catch (error) {*

*res.****status****(500).****json****({ error: 'Erro ao listar os produtos.' });*

*}*

*}*

*async* ***createProduto****(req, res) {*

*try {*

*const produto = await ProdutoService.****createProduto****(req.body);*

*res.****json****(produto);*

*} catch (error) {*

*res.****status****(400).****json****({ error: 'Erro ao criar o produto.' });*

*}*

*}*

*}*

*module.exports = new* ***ProdutoController****();*

**|userController.js**

*const UserService =* ***require****('../services/userService');*

*const { generateToken } =* ***require****('../utils/jwt');*

*const sendEmail =* ***require****('../email'); // Ajuste conforme a localização do seu arquivo*

*class UserController {*

*async* ***register****(req, res) {*

*try {*

*const user = await UserService.****registerUser****(req.body);*

*res.****status****(201).****json****(user);*

*} catch (error) {*

*res.****status****(400).****json****({ error: 'Erro ao registrar usuário' });*

*console.****log****(error);*

*}*

*}*

*async* ***login****(req, res) {*

*try {*

*const user = await UserService.****loginUser****(req.body.email, req.body.senha);*

*const token =* ***generateToken****(user.id);*

*res.****json****({ token });*

*} catch (error) {*

*res.****status****(400).****json****({ error: error.message });*

*}*

*}*

*async* ***getUserProfile****(req, res) {*

*try {*

*const userId = req.user.id;*

*const user = await UserService.****getUserProfile****(userId);*

*res.****json****(user);*

*} catch (error) {*

*res.****status****(404).****json****({ error: 'Usuário não encontrado' });*

*console.****log****(error);*

*}*

*}*

*async* ***updateUserProfile****(req, res) {*

*try {*

*const userId = req.user.id;*

*const updatedUser = await UserService.****updateUserProfile****(userId, req.body);*

*res.****json****(updatedUser);*

*} catch (error) {*

*res.****status****(400).****json****({ error: 'Erro ao atualizar perfil de usuário' });*

*console.****log****(error);*

*}*

*}*

*async* ***redefinirSenha****(req, res) {*

*try {*

*const { email, senha\_antiga, senha\_nova } = req.body;*

*const user = await UserService.****verifyUser****(email, senha\_antiga);*

*if (!user) {*

*return res.****status****(400).****json****({ error: 'Email ou senha antiga incorretos.' });*

*}*

*await UserService.****updatePassword****(email, senha\_nova);*

*await* ***sendEmail****(email);*

*res.****status****(200).****json****({ message: 'Senha redefinida com sucesso.' });*

*} catch (error) {*

*res.****status****(400).****json****({ error: error.message });*

*console.****log****(error);*

*}*

*}*

*}*

*module.exports = new* ***UserController****();*

**|models/**

**|categoryModel.js**

*const pool =* ***require****('../config/database');*

*class CategoryModel {*

*static async* ***getAll****() {*

*const result = await pool.****query****('SELECT \* FROM categorias');*

*return result.rows;*

*}*

*static async* ***create****(category) {*

*const result = await pool.****query****(*

*'INSERT INTO categorias (descricao) VALUES ($1) RETURNING \*',*

*[category.descricao]*

*);*

*return result.rows[0];*

*}*

*}*

*module.exports = CategoryModel;*

**|pedidoModel.js**

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

*const pedidoSchema = new mongoose.****Schema****({*

*cliente: String,*

*data: Date,*

*total: Number,*

*status: String*

*});*

*class Pedido {*

*constructor() {*

*this.model = mongoose.****model****('Pedido', pedidoSchema);*

*}*

*async* ***criarPedido****(pedido) {*

*const novoPedido = new this.****model****(pedido);*

*return await novoPedido.****save****();*

*}*

*async* ***listarPedidos****() {*

*return await this.model.****find****();*

*}*

*async* ***atualizarPedido****(id, pedido) {*

*return await this.model.****findByIdAndUpdate****(id, pedido, { new: true });*

*}*

*async* ***deletarPedido****(id) {*

*return await this.model.****findByIdAndRemove****(id);*

*}*

*}*

*module.exports = Pedido;*

|produtoModel.js

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

*const* produtoSchema = new mongoose.**Schema**({

    nome: String,

    descricao: String,

    preco: Number,

    quantidade: Number

});

*class* Produto {

*constructor*() {

        this.model = mongoose.**model**('Produto', produtoSchema);

    }

    async **criarProduto**(*produto*) {

*const* novoProduto = new this.**model**(produto);

        return await novoProduto.**save**();

    }

    async **listarProdutos**() {

        return await this.model.**find**();

    }

    async **atualizarProduto**(*id*, *produto*) {

        return await this.model.**findByIdAndUpdate**(id, produto, { new: true });

    }

    async **deletarProduto**(*id*) {

        return await this.model.**findByIdAndRemove**(id);

    }

}

*module*.*exports* = Produto;

**|userModel.js**

*const pool =* ***require****('../config/database');*

*class UserModel {*

*static async* ***getByEmail****(email) {*

*const result = await pool.****query****('SELECT \* FROM usuarios WHERE email = $1', [email]);*

*return result.rows[0];*

*}*

*static async* ***create****(user) {*

*const result = await pool.****query****(*

*'INSERT INTO usuarios (nome, email, senha) VALUES ($1, $2, $3) RETURNING \*',*

*[user.nome, user.email, user.senha]*

*);*

*return result.rows[0];*

*}*

*static async* ***updatePassword****(email, novaSenha) {*

*const result = await pool.****query****(*

*'UPDATE usuarios SET senha = $1 WHERE email = $2 RETURNING \*',*

*[novaSenha, email]*

*);*

*return result.rows[0];*

*}*

*//obter o ID*

*static async* ***getById****(id) {*

*const result = await pool.****query****('SELECT \* FROM usuarios WHERE id = $1', [id]);*

*return result.rows[0];*

*}*

*static async* ***update****(user) {*

*const result = await pool.****query****(*

*'UPDATE usuarios SET nome = $1, email = $2 WHERE id = $3 RETURNING \*',*

*[user.nome, user.email, user.id]*

*);*

*return result.rows[0];*

*}*

*}*

*module.exports = UserModel;*

**|routs/**

**|categoryRoute.js**

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

*const* router = express.**Router**();

*const* CategoryController = **require**('../controllers/categoryController');

*const* { verifyToken } = **require**('../utils/jwt');

router.**get**('/', CategoryController.listCategories);

router.**post**('/', verifyToken, CategoryController.createCategory);

*module*.*exports* = router;

**|pedidoRoute.js**

//pedidoRoute.js

**//rotas dos pedidos**

*const* express = **require**("express");

*const* router = express.**Router**();

// Importando o controller de pedidos

*const* pedidoController = **require**("../controllers/pedidoController");

// Definindo as rotas de pedidos

router.**get**("/lista-de-pedidos", pedidoController.listPedidos);

router.**get**("/:id", pedidoController.buscarPedidoPorId);

router.**post**("/criar-pedido", pedidoController.createPedido);

router.**put**("/:id", pedidoController.atualizarPedido);

router.**delete**("/:id", pedidoController.deletarPedido);

*module*.*exports* = router;

**|produtoRouter.js**

**// src/routes/produtoRoute.js**

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

*const* router = express.**Router**();

*const* Produto = **require**('../models/produtoModel');

// Instância da classe Produto

*const* produtoService = new **Produto**();

**// Criar um produto**

router.**post**('/novo\_produto', async (*req*, *res*) *=>* {

    try {

        await produtoService.**criarProduto**(req.body);

        res.**status**(201).**send**({ message: 'Produto criado com sucesso!' });

    } catch (error) {

        res.**status**(400).**send**({ message: 'Erro ao criar produto' });

        console.**log**(error);

    }

});

// Listar todos os produtos

router.**get**('/lista-de-produtos', async (*req*, *res*) *=>* {

    try {

*const* produtos = await produtoService.**listarProdutos**();

        res.**send**(produtos);

    } catch (error) {

        res.**status**(500).**send**({ message: 'Erro ao listar produtos' });

    }

});

**// Buscar um produto por ID**

router.**get**('/:id', async (*req*, *res*) *=>* {

    try {

*const* produto = await produtoService.model.**findById**(req.params.id);

        if (!produto) {

            res.**status**(404).**send**({ message: 'Produto não encontrado' });

        } else {

            res.**send**(produto);

        }

    } catch (error) {

        res.**status**(500).**send**({ message: 'Erro ao buscar produto' });

    }

});

**// Atualizar um produto**

router.**put**('/:id', async (*req*, *res*) *=>* {

    try {

*const* produto = await produtoService.**atualizarProduto**(req.params.id, req.body);

        if (!produto) {

            res.**status**(404).**send**({ message: 'Produto não encontrado' });

        } else {

            res.**send**({ message: 'Produto atualizado com sucesso!' });

        }

    } catch (error) {

        res.**status**(400).**send**({ message: 'Erro ao atualizar produto' });

    }

});

**// Deletar um produto**

router.**delete**('/:id', async (*req*, *res*) *=>* {

    try {

        await produtoService.**deletarProduto**(req.params.id);

        res.**send**({ message: 'Produto deletado com sucesso!' });

    } catch (error) {

        res.**status**(500).**send**({ message: 'Erro ao deletar produto' });

    }

});

*module*.*exports* = router;

**|routes.js**

*const* express = **require**("express");

*const* routes = express.**Router**();

*const* userRoutes = **require**("./userRoutes");

*const* categoryRoutes = **require**("./categoryRoutes");

*const* produtoRoute = **require**("./produtoRoute");

*const* pedidoRoute = **require**("./pedidoRoute");

*const* produtoRoutes = **require**("../routs/produtoRoute")

routes.**use**("/user", userRoutes);

routes.**use**("/category", categoryRoutes);

routes.**use**("/produto", produtoRoute);

routes.**use**("/pedido", pedidoRoute);

routes.**use**("/produtos", produtoRoutes);

*module*.*exports* = routes;

**|userRoutes.js** *const* express = **require**('express');

*const* router = express.**Router**();

*const* UserController = **require**('../controllers/userController');

*const* { check, validationResult } = **require**('express-validator');

*const* { verifyToken } = **require**('../utils/jwt');

**// Validação de dados**

*const* validateRegister = [

**check**('nome').**not**().**isEmpty**().**withMessage**('Nome é obrigatório'),

**check**('email').**isEmail**().**withMessage**('E-mail inválido'),

**check**('senha').**not**().**isEmpty**().**withMessage**('Senha é obrigatória'),

];

*const* validateLogin = [

**check**('email').**isEmail**().**withMessage**('E-mail inválido'),

**check**('senha').**not**().**isEmpty**().**withMessage**('Senha é obrigatória'),

];

*const* validateRedefinirSenha = [

**check**('email').**isEmail**().**withMessage**('E-mail inválido'),

**check**('senha\_antiga').**not**().**isEmpty**().**withMessage**('Senha antiga é obrigatória'),

**check**('senha\_nova').**not**().**isEmpty**().**withMessage**('Senha nova é obrigatória'),

];

router.**get**('/usuario', verifyToken, UserController.getUserProfile);

router.**put**('/usuario', verifyToken, UserController.updateUserProfile);

router.**post**('/register', validateRegister, UserController.register);

router.**post**('/login', validateLogin, UserController.login);

router.**post**('/redefinir\_senha', validateRedefinirSenha, UserController.redefinirSenha);

**// Middleware de erro**

router.**use**((*err*, *req*, *res*, *next*) *=>* {

    console.**error**(err);

    res.**status**(500).**json**({ message: 'Erro interno' });

});

*module*.*exports* = router;

**|services**

**|categoryService.js**

*const CategoryModel =* ***require****('../models/categoryModel');*

*class CategoryService {*

*async* ***listCategories****() {*

*return await CategoryModel.****getAll****();*

*}*

*async* ***createCategory****(categoryData) {*

*return await CategoryModel.****create****(categoryData);*

*}*

*}*

*module.exports = new* ***CategoryService****();*

**|pedidoService.js**

*const PedidoModel =* ***require****('../models/pedidoModel');*

*class PedidoService {*

*static async* ***listPedidos****() {*

*return PedidoModel.****getAll****();*

*}*

*static async* ***createPedido****(data) {*

*return PedidoModel.****create****(data);*

*}*

*}*

*module.exports = PedidoService;*

**|produtoService.js**

*const ProdutoModel =* ***require****('../models/produtoModel');*

*class ProdutoService {*

*static async* ***listProduto****() {*

*return ProdutoModel.****getAll****();*

*}*

*static async* ***createProduto****(data) {*

*return ProdutoModel.****create****(data);*

*}*

*}*

*module.exports = ProdutoService;*

**|userService.js**

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

*const UserModel =* ***require****('../models/userModel');*

*class UserService {*

*static async* ***registerUser****(user) {*

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

*return UserModel.****create****(user);*

*}*

*static async* ***loginUser****(email, senha) {*

*const user = await UserModel.****getByEmail****(email);*

*if (!user) throw new* ***Error****('Usuário não encontrado');*

*const validPassword = await bcrypt.****compare****(senha, user.senha);*

*if (!validPassword) throw new* ***Error****('Senha incorreta');*

*return user;*

*}*

*static async* ***getUserProfile****(userId) {*

*return UserModel.****getById****(userId);*

*}*

*static async* ***updateUserProfile****(userId, updatedUser) {*

*const user = await UserModel.****getById****(userId);*

*if (!user) throw new* ***Error****('Usuário não encontrado');*

*if (!updatedUser.nome || !updatedUser.email || !updatedUser.senha) {*

*throw new* ***Error****('Nome, email e senha são obrigatórios');*

*}*

*const existingUser = await UserModel.****getByEmail****(updatedUser.email);*

*if (existingUser && existingUser.id !== userId) {*

*throw new* ***Error****('O email já está em uso por outro usuário');*

*}*

*if (updatedUser.senha) {*

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

*}*

*Object.****assign****(user, updatedUser);*

*return UserModel.****update****(user);*

*}*

*static async* ***updatePassword****(email, novaSenha) {*

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

*await UserModel.****updatePassword****(email, hashedPassword);*

*}*

*}*

*module.exports = UserService;*

**|utils**

**|bcrypt.js**

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

*function* ***hashPassword****(password) {*

*return bcrypt.****hash****(password, 10);*

*}*

*function* ***comparePassword****(password, hashedPassword) {*

*return bcrypt.****compare****(password, hashedPassword);*

*}*

*module.exports = { hashPassword, comparePassword };*

**|jwt.js**

*const jwt =* ***require****('jsonwebtoken');*

*const { jwtSecret } =* ***require****('../config/enviroment');*

*const* ***generateToken*** *= (userId) => {*

*return jwt.****sign****({ userId }, jwtSecret, { expiresIn: '1h' });*

*};*

*const* ***verifyToken*** *= (req, res, next) => {*

*const token = req.headers['authorization'];*

*if (!token) {*

*return res.****status****(401).****json****({ error: 'Token não fornecido' });*

*}*

*jwt.****verify****(token.****split****(' ')[1], jwtSecret, (err, decoded) => {*

*if (err) {*

*return res.****status****(401).****json****({ error: 'Token inválido' });*

*}*

*req.user = { id: decoded.userId };*

***next****();*

*});*

*};*

*module.exports = { generateToken, verifyToken };*

**|conexao.js**

require('dotenv').config();

*const* knex = require('knex')({

    client: 'pg',

    connection: {

    host: process.env.DB\_HOST,

    port: process.env.DB\_PORT,

    user: process.env.DB\_USER,

    password: process.env.DB\_PASS,

    database: process.env.DB\_NAME,

    ssl: { rejectUnauthorized: false },

  },

});

*module*.*exports* = knex;

**|email.js**

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

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

*dotenv.****config****();*

*async function* ***sendEmail****(email) {*

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

*host: process.env.MAIL\_HOST,*

*port: process.env.MAIL\_PORT,*

*secure: process.env.MAIL\_PORT === '465', // true para 465, false para outros*

*auth: {*

*user: process.env.MAIL\_USER,*

*pass: process.env.MAIL\_PASS*

*}*

*});*

*let mailOptions = {*

*from: process.env.MAIL\_USER,*

*to: email,*

*subject: "Redefinição de senha",*

*html: "<p>Sua senha foi redefinida com sucesso.</p>"*

*};*

*try {*

*await transporter.****sendMail****(mailOptions);*

*console.****log****('E-mail enviado com sucesso');*

*} catch (error) {*

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

*throw error;*

*}*

*}*

*module.exports = sendEmail;*

**|pdv.sql**

create table **usuarios** (

        id *serial* primary key,

        nome *text* not null,

        email *text* not null unique,

        senha *text* not null

);

create table **categorias** (

        id *serial* primary key,

        descricao *text* not null unique

);

insert into categorias (descricao) values

('Informática'),

('Celulares'),

('Beleza e Perfurmaria'),

('Mercado'),

('Livros e Papelaria'),

('Brinquedos'),

('Moda'),

('Bebê'),

('Games');

create table **produtos** (

        id *serial* primary key,

        descricao *text* not null unique,

        quantidade\_estoque *integer* not null,

        valor *integer* not null,

        categoria\_id *integer* references categorias(id)

);

create table **clientes** (

        id *serial* primary key,

        nome *text* not null,

        email *text* not null unique,

        cpf *varchar*(11) unique,

        cep *varchar*(8),

        rua *text*,

        numero *text*,

        bairro *text*,

        cidade *text*,

        estado *varchar*(2)

);

**|server.js**

require('dotenv').config();

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

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

*const* userRoutes = require('./routes/userRoutes');

*const* app = express();

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

app.use(bodyParser.json());

app.use('/user', userRoutes);

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

    console.log(`Server running on port ${PORT}`);

});

|.env

PORT = 3000

DB\_HOST = localhost

DB\_PORT = 5432

DB\_USER = postgres

DB\_NAME = pdv

DB\_PASS = b030515

JWT\_SECRET = senha\_jwt

MAIL\_HOST=smtp.gmail.com

MAIL\_PORT=587

MAIL\_USER=bugtrakers@gmail.com

MAIL\_PASS= fxaa avww tpqd jayw

MAIL\_NAME="BugTrakers"

MAIL\_FROM="bugtrakers@gmail.com"

**|index.js**

require('dotenv').config()

*const* app = require('./PDV/SRC/config/app');

*const* { port } = require('./PDV/SRC/config/enviroment');

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

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

});

**|package-lock.json**

**|package.json**

**|README.md**

**|Setup\_db.js**

*const* { Pool } = **require**('pg');

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

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

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

dotenv.**config**();

*const* pool = new **Pool**({

    user: process.env.DB\_USER,

    host: process.env.DB\_HOST,

    database: process.env.DB\_NAME,

    password: process.env.DB\_PASS,

    port: process.env.DB\_PORT,

});

*const* **setupDb** = async () *=>* {

*const* sql = fs.**readFileSync**(path.**join**(\_\_dirname, 'setup\_db.sql')).**toString**();

    try {

        await pool.**query**(sql);

        console.**log**('Database setup completed successfully.');

    } catch (err) {

        console.**error**('Error setting up the database:', err);

    } finally {

        pool.**end**();

    }

};

**setupDb**();

**DIVISÃO DE TAREFAS – 2° SPRINT (entregar dia 13\_Terça-feira)**

**(Feedback até Domingo)**

**Comandos git:**

**- git branch (mostra a branch atual)  
- git checkout -r (mostra todas as branchs)**

**- git remote -v**

**Banco de Dados**

Crie as seguintes tabelas e colunas abaixo:

**ATENÇÃO! Os nomes das tabelas e das colunas a serem criados devem seguir exatamente os nomes listados abaixo.**

* **produtos**
  + id
  + descricao
  + quantidade\_estoque
  + valor
  + categoria\_id
* clientes
  + id
  + nome
  + email (campo único)
  + cpf (campo único)
  + cep
  + rua
  + numero
  + bairro
  + cidade
  + estado

**ENDPOINTS DE PRODUTO**

* **Cadastrar Produto**
  + POST /produto
  + Essa é a rota que permite o usuário logado cadastrar um novo produto no sistema.
  + Critérios de aceite:
  + - Validar os campos obrigatórios:
  + - descricao
  + - quantidade\_estoque
  + - valor
  + - **categoria\_id**
  + - A categoria informada na qual o produto será vinculado deverá existir**.**
* **Editar dados do produto ( Pedro )**
  + PUT /produto/:id
  + Essa é a rota que permite o usuário logado a atualizar as informações de um produto cadastrado.
  + Critérios de aceite:
  + - Validar se existe produto para o id enviado como parâmetro na rota.
  + - Validar os campos obrigatórios:
  + - descricao
  + - quantidade\_estoque
  + - valor
  + - categoria\_id
  + - A categoria informada na qual o produto será vinculado deverá existir.
* **Listar Produtos**
* **Detalhar Produto ( Bruno )**
  + GET /produto/:id
  + Essa é a rota que permite o usuário logado obter um de seus produtos cadastrados.
  + Critérios de aceite:
  + - Validar se existe produto para o id enviado como parâmetro na rota.
* **Excluir Produto por ID ( Pedro )**
  + DELETE /produto/:id
  + Essa é a rota que será chamada quando o usuário logado quiser excluir um de seus produtos cadastrados.
  + Critérios de aceite:
  + - Validar se existe produto para o id enviado como parâmetro na rota.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ENDPOINTS DE CLIENTE**

* **Cadastrar Cliente ( Malu )**
  + POST /cliente
  + Essa é a rota que permite usuário logado cadastrar um novo cliente no sistema.
  + Critérios de aceite:
  + - Validar os campos obrigatórios:
  + - nome
  + - email
  + - cpf
  + - O campo e-mail no banco de dados deve ser único para cada registro, não permitindo dois clientes possuírem o mesmo e-mail.
  + - O campo cpf no banco de dados deve ser único para cada registro, não permitindo dois clientes possuírem o mesmo cpf.
* **Editar dados do cliente ( Marianne )**
  + PUT /cliente/:id
  + Essa é a rota que permite o usuário realizar atualização de um cliente cadastrado.
  + Critérios de aceite:
  + - Validar se existe cliente para o id enviado como parâmetro na rota.
  + - Validar os campos obrigatórios:
  + - nome
  + - email
  + - cpf
  + - O campo e-mail no banco de dados deve ser único para cada registro, não permitindo dois clientes possuírem o mesmo e-mail.
  + - O campo cpf no banco de dados deve ser único para cada registro, não permitindo dois clientes possuírem o mesmo cpf.
* **Listar Clientes ( Fábio )**
  + GET /cliente
  + Essa é a rota que será chamada quando o usuário logado quiser listar todos os clientes cadastrados.
* **Detalhar Cliente ( Malu )**
* GET /cliente/:id
* Essa é a rota que será chamada quando o usuário logado quiser obter um de seus clientes cadastrados.
* Critérios de aceite:
* - Validar se existe cliente para o id enviado como parâmetro na rota