O Catálogo do Multiverso (Produtos)

Agora, vamos criar o CRUD para os produtos.

1. Modelagem do Produto (models/Product.js)

models/Product.js

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

const ProductSchema = new mongoose.Schema({
name: { type: String, required: true },
description: { type: String, required: true },
price: { type: Number, required: true },
category: { type: String, required: true },
stock: { type: Number, required: true, default: 0 },
imageUrl: { type: String, required: true }
});

module.exports = mongoose.model('Product', ProductSchema);
```

2. Middlewares de Proteção (middleware/auth.js e middleware/admin.js)

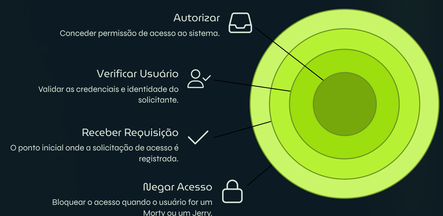
Precisamos de guardas para nossas rotas. Um para verificar se o usuário está logado e outro para verificar se ele é um Rick (admin).

middleware/auth.js

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

module.exports = function(req, res, next) {
const token = req.header('x-auth-token'); // ou req.header('Authorization').split(' ')[1];
if (!token) {
return res.status(401).json({ msg: 'Sem token, autorização negada.' });
}
try {
const decoded = jwt.verify(token, process.env.JWT_SECRET);
req.user = decoded.user;
next();
} catch (err) {
res.status(401).json({ msg: 'Token não é válido.' });
}
};
```

middleware/admin.js

```
module.exports = function(req, res, next) {
if (req.user && req.user.role === 'admin') {
next();
} else {
res.status(403).json({ msg: 'Acesso negado. Nível de acesso de Morty detectado.' });
}
};
```



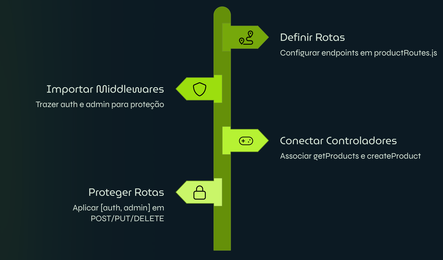
3. Controlador e Rotas de Produtos

A lógica do CRUD de produtos e as rotas que usam os middlewares.

controllers/productController.js

```
const Product = require('../models/Product');

// EXEMPLO: (CREATE) POST /products - Apenas para admins
exports.createProduct = async (req, res) => {
  try {
    // 1. Como está é uma rota de admin, não precisamos verificar quem está criando.
    // O middleware `admin` já fez essa verificação para nós.
    // 2. Crie uma nova instância do modelo Product com os dados do `req.body`.
    const newProduct = new Product(req.body);

    // 3. Salve o novo produto no banco de dados.
    const product = await newProduct.save();

    // 4. Retorne o produto criado com status 201 (Created).
    res.status(201).json(product);
  } catch (err) {
    res.status(500).send('Erro no servidor');
  }
};

// SUA VEZ: (READ) GET /products - Aberto para todos
exports.getProducts = async (req, res) => {
  try {
    // DICA: Esta é uma rota pública para listar produtos.
    // 1. Você pode opcionalmente verificar `req.query` por filtros, como `category`.
    //    Ex: const { category } = req.query;
    // 2. Crie um objeto de filtro. Se houver uma categoria, o filtro será `{ category: category }`. Se não, será um objeto vazio `{}`.
    // 3. Use `Product.find(filterObject)` para buscar os produtos no banco.
    // 4. Retorne a lista de produtos encontrados em formato JSON.
  } catch (err) {
    res.status(500).send('Erro no servidor');
  }
};
// Adicione aqui as outras funções do CRUD para produtos.
// exports.getProductById = async (req, res) => { ... }
// exports.updateProduct = async (req, res) => { ... }
// exports.deleteProduct = async (req, res) => { ... }
```

Rotas de Produtos



routes/productRoutes.js

```
const express = require('express');
const router = express.Router();
const auth = require('../middleware/auth');
const admin = require('../middleware/admin');
const { getProducts, createProduct /*, ...outras*/ } = require('../controllers/productController');

router.get('/', getProducts);
router.post('/', [auth, admin], createProduct);
// ... outras rotas PUT e DELETE protegidas por [auth, admin]
// ... rota GET /:id

module.exports = router;
```