Controle de Identidades (Usuários e Autenticação)

Vamos começar criando a base para nossos usuários.

1. Modelagem do Usuário (models/User.js)

Aqui definimos como um "Usuário" será salvo no banco. Adicionaremos um campo role para o controle de acesso (Rick vs. Morty) e um campo cart que usaremos mais tarde.

models/User.js

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

const UserSchema = new mongoose.Schema({
name: { type: String, required: true },
email: { type: String, required: true, unique: true },
password: { type: String, required: true },
role: { type: String, enum: ['user', 'admin'], default: 'user' },
cart: [{
product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product' },
quantity: { type: Number, default: 1 }
}]
});

// Hook para fazer o hash da senha ANTES de salvar no banco
UserSchema.pre('save', async function(next) {
if (!this.isModified('password')) {
return next();
}
const salt = await bcrypt.genSalt(10);
this.password = await bcrypt.hash(this.password, salt);
next();
});

module.exports = mongoose.model('User', UserSchema);
```

2. Controlador de Autenticação (controllers/authController.js)

Aqui fica a lógica para registrar e logar usuários.

controllers/authController.js

```
const User = require('../models/User');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

// EXEMPLO: (CREATE) POST /users/register
exports.register = async (req, res) => {
    try {
        const { name, email, password } = req.body;
        let user = await User.findOne({ email });
        if (user) {
            return res.status(400).json({ msg: 'Um usuário com este e-mail já existe nesta dimensão.' });
        }
        user = new User({ name, email, password });
        await user.save();
        // 5. Enviar uma resposta de sucesso
        res.status(201).json({ msg: 'Usuário registrado com sucesso!' });
    } catch (err) {
        // 6. Se algo der errado, capturar o erro e enviar uma resposta de erro genérica
        res.status(500).send('Erro no servidor');
    }
};

// SUA VEZ: POST /auth/login
exports.login = async (req, res) => {
    try {
        // DICA: O processo de login é muito parecido com o de registro.
        // 1. Extraia 'email' e 'password' do `req.body`.
        // 2. Encontre o usuário no banco de dados pelo email usando `User.findOne()`.
        // 3. Se o usuário não for encontrado, retorne um erro 400 com uma mensagem genérica de "Credenciais inválidas.".
        // 4. Se o usuário for encontrado, compare a senha enviada com a senha hashada no banco usando `bcrypt.compare()`.
        // 5. Se as senhas não baterem, retorne o mesmo erro 400 de "Credenciais inválidas.". Não seja específico sobre o que está errado (email ou senha) por segurança.
        // 6. Se as senhas baterem, crie o 'payload' para o token JWT. Ele deve conter informações úteis, como o ID e o 'role' do usuário.
        //    Ex: const payload = { user: { id: user.id, role: user.role } };
        // 7. Gere o token usando `jwt.sign()`, passando o payload, sua chave secreta (do .env) e uma data de expiração.
        // 8. Envie o token de volta para o cliente em formato JSON.

    } catch (err) {
        res.status(500).send('Erro no servidor');
    }
};
```

3. Rotas (routes/authRoutes.js)

Crie as rotas para expor os controladores.

routes/authRoutes.js

```
const express = require('express');
const router = express.Router();
const { register, login } = require('../controllers/authController');

router.post('/register', register);
router.post('/login', login);
module.exports = router;
```