Estruturas de diretórios

cupcake-store/

├── backend/

│ ├── controllers/

│ ├── models/

│ ├── routes/

│ ├── app.js

│ ├── config.js

│ └── server.js

├── frontend/

│ ├── public/

│ ├── src/

│ ├── index.html

│ ├── package.json

│ └── webpack.config.js

└── README.md

const express = require('express');

const mongoose = require('mongoose');

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

const app = express();

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

app.use(express.json());

app.use('/api/cupcakes', require('./routes/cupcakeRoutes'));

mongoose.connect(config.mongoURI, {

useNewUrlParser: true,

useUnifiedTopology: true,

}).then(() => {

console.log('MongoDB connected');

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

}).catch(err => console.error(err));

module.exports = {

mongoURI: 'your-mongodb-uri-here'

};

const mongoose = require('mongoose');

const CupcakeSchema = new mongoose.Schema({

name: {

type: String,

required: true

},

price: {

type: Number,

required: true

},

description: {

type: String,

required: true

},

image: {

type: String,

required: true

}

});

module.exports = mongoose.model('Cupcake', CupcakeSchema);

const express = require('express');

const router = express.Router();

const Cupcake = require('../models/Cupcake');

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

try {

const cupcakes = await Cupcake.find();

res.json(cupcakes);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

router.post('/', async (req, res) => {

const { name, price, description, image } = req.body;

const newCupcake = new Cupcake({ name, price, description, image });

try {

const savedCupcake = await newCupcake.save();

res.status(201).json(savedCupcake);

} catch (err) {

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

}

});

module.exports = router;

import React, { useEffect, useState } from 'react';

import './App.css';

function App() {

const [cupcakes, setCupcakes] = useState([]);

useEffect(() => {

fetch('/api/cupcakes')

.then(res => res.json())

.then(data => setCupcakes(data))

.catch(err => console.error(err));

}, []);

return (

<div className="App">

<h1>Cupcake Store</h1>

<div className="cupcake-list">

{cupcakes.map(cupcake => (

<div className="cupcake" key={cupcake.\_id}>

<img src={cupcake.image} alt={cupcake.name} />

<h2>{cupcake.name}</h2>

<p>{cupcake.description}</p>

<p>${cupcake.price.toFixed(2)}</p>

</div>

))}

</div>

</div>

);

}

export default App;

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Cupcake Store</title>

</head>

<body>

<div id="root"></div>

<script src="/dist/bundle.js"></script>

</body>

</html>

const path = require('path');

module.exports = {

entry: './src/App.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'bundle.js'

},

module: {

rules: [

{

test: /\.js$/,

exclude: /node\_modules/,

use: {

loader: 'babel-loader'

}

},

{

test: /\.css$/,

use: ['style-loader', 'css-loader']

}

]

},

devServer: {

contentBase: path.join(\_\_dirname, 'public'),

proxy: {

'/api': 'http://localhost:5000'

}

}

};

Modelo nesecario

Criar moldelo de usúario

npm install bcryptjs jsonwebtoken

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

}

});

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);

1. Criar o modelo de usuário ( 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

}

});

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);

1. Criar rotas de autenticação ( routes/authRoutes.js):

const express = require('express');

const router = express.Router();

const jwt = require('jsonwebtoken');

const bcrypt = require('bcryptjs');

const User = require('../models/User');

router.post('/register', async (req, res) => {

const { name, email, password } = req.body;

try {

const user = new User({ name, email, password });

await user.save();

const token = jwt.sign({ id: user.\_id }, 'your\_jwt\_secret', { expiresIn: '1h' });

res.status(201).json({ token });

} catch (err) {

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

}

});

router.post('/login', async (req, res) => {

const { email, password } = req.body;

try {

const user = await User.findOne({ email });

if (!user || !(await bcrypt.compare(password, user.password))) {

return res.status(400).json({ message: 'Invalid credentials' });

}

const token = jwt.sign({ id: user.\_id }, 'your\_jwt\_secret', { expiresIn: '1h' });

res.json({ token });

} catch (err) {

res.status(500).json({ message: err.message });

}

});

module.exports = router;

1. Proteger rotas com autenticação ( middleware/auth.js):

const jwt = require('jsonwebtoken');

function auth(req, res, next) {

const token = req.header('x-auth-token');

if (!token) return res.status(401).json({ message: 'No token, authorization denied' });

try {

const decoded = jwt.verify(token, 'your\_jwt\_secret');

req.user = decoded;

next();

} catch (err) {

res.status(400).json({ message: 'Token is not valid' });

}

}

module.exports = auth;

1. Criar o modelo de carrinho ( models/Cart.js):

const mongoose = require('mongoose');

const CartSchema = new mongoose.Schema({

user: {

type: mongoose.Schema.Types.ObjectId,

ref: 'User',

required: true

},

items: [

{

product: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Cupcake',

required: true

},

quantity: {

type: Number,

required: true,

default: 1

}

}

]

});

module.exports = mongoose.model('Cart', CartSchema);

1. Criar rotas de carrinho ( routes/cartRoutes.js):

const express = require('express');

const router = express.Router();

const auth = require('../middleware/auth');

const Cart = require('../models/Cart');

router.get('/', auth, async (req, res) => {

try {

const cart = await Cart.findOne({ user: req.user.id }).populate('items.product');

res.json(cart);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

router.post('/', auth, async (req, res) => {

const { productId, quantity } = req.body;

try {

let cart = await Cart.findOne({ user: req.user.id });

if (!cart) {

cart = new Cart({ user: req.user.id, items: [] });

}

const itemIndex = cart.items.findIndex(item => item.product.toString() === productId);

if (itemIndex > -1) {

cart.items[itemIndex].quantity += quantity;

} else {

cart.items.push({ product: productId, quantity });

}

await cart.save();

res.status(201).json(cart);

} catch (err) {

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

}

});

router.delete('/:itemId', auth, async (req, res) => {

try {

const cart = await Cart.findOne({ user: req.user.id });

cart.items = cart.items.filter(item => item.\_id.toString() !== req.params.itemId);

await cart.save();

res.json(cart);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

module.exports = router;

**Integração de Pagamento (Stripe)**

1. Instalar o pacote Stripe:

npm install stripe

1. Configurar Stripe ( config.js):

module.exports = {

mongoURI: 'your-mongodb-uri-here',

jwtSecret: 'your\_jwt\_secret',

stripeSecretKey: 'your\_stripe\_secret\_key'

};

1. Criar rotas de pagamento ( routes/paymentRoutes.js):

const express = require('express');

const router = express.Router();

const auth = require('../middleware/auth');

const stripe = require('stripe')(require('../config').stripeSecretKey);

const Cart = require('../models/Cart');

router.post('/checkout', auth, async (req, res) => {

try {

const cart = await Cart.findOne({ user: req.user.id }).populate('items.product');

if (!cart) {

return res.status(400).json({ message: 'No items in cart' });

}

const total = cart.items.reduce((sum, item) => sum + item.product.price \* item.quantity, 0);

const session = await stripe.checkout.sessions.create({

payment\_method\_types: ['card'],

line\_items: cart.items.map(item => ({

price\_data: {

currency: 'usd',

product\_data: {

name: item.product.name,

images: [item.product.image]

},

unit\_amount: item.product.price \* 100

},

quantity: item.quantity

})),

mode: 'payment',

success\_url: 'your\_success\_url',

cancel\_url: 'your\_cancel\_url'

});

res.json({ id: session.id });

} catch (err) {

res.status(500).json({ message: err.message });

}

});

module.exports = router;

#### Autenticação de Usuário

1. Adicionar formulários de login e registro ( src/components/Auth.js):

import React, { useState } from 'react';

function Auth({ onAuth }) {

const [isLogin, setIsLogin] = useState(true);

const [form, setForm] = useState({ name: '', email: '', password: '' });

const handleChange = (e) => {

setForm({ ...form, [e.target.name]: e.target.value });

};

const handleSubmit = async (e) => {

e.preventDefault();

const endpoint = isLogin ? '/api/auth/login' : '/api/auth/register';

const response = await fetch(endpoint, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(form)

});

const data = await response.json();

if (data.token) {

onAuth(data.token);

}

};

return (

<div>

<h2>{isLogin ? 'Login' : 'Register'}</h2>

<form onSubmit={handleSubmit}>

{!isLogin && <input type="text" name="name" placeholder="Name" onChange={handleChange} />}

<input type="email" name="email" placeholder="Email" onChange={handleChange} />

<input type="password" name="password" placeholder="Password" onChange={handleChange} />

<button type="submit">{isLogin ? 'Login' : 'Register'}</button>

</form>

<button onClick={() => setIsLogin(!isLogin)}>

{isLogin ? 'Create an account' : 'Have an account? Login'}

</button>

</div>

);

}

export default Auth;

#### Carrinho de compras

1. Adicionar componente do carrinho ( src/components/Cart.js):

import React, { useEffect, useState } from 'react';

function Cart({ token }) {

const [cart, setCart] = useState(null);

useEffect(() => {

const fetchCart = async () => {

const response = await fetch('/api/cart', {

headers: { 'x-auth-token': token }

});

const data = await response.json();

setCart(data);

};

fetchCart();

}, [token]);

const handleCheckout = async () => {

const response = await fetch('/api/payment/checkout', {

method: 'POST',

headers: { 'Content-Type': 'application/json', 'x-auth-token': token }

});

const data = await response.json();

window.location.href = data.url;

};

if (!cart) return <div>Loading...</div>;

return (

<div>

<h2>Cart</h2>

{cart.items.map(item => (

<div key={item.\_id}>

<h3>{item.product.name}</h3>

<p>{item.quantity} x ${item.product.price.toFixed(2)}</p>

</div>

))}

<button onClick={handleCheckout}>Checkout</button>

</div>

);

}

export default Cart;