AdaptNxt Assignment

Assignment: Simple E-commerce API

Project Requirements: -

Product LisΘngs, The API should allow users to fetch a list of available products.

- Cart Management, Users should be able to manage their shopping cart (add, update, remove items).

- Order CreaΘon, Users should be able to create an order from their cart. - User Roles and AuthenΘcaΘon, Implement user authenΘcaΘon and two roles (customer, admin). - JWT (JSON Web Tokens) should be used for securing routes

- Customer: Can only view products, add them to the cart, and place orders. - Admin: Can also manage (add, update, delete) products. - Basic Frontend (OpΘonal but Recommended), Create a basic HTML page with forms or buΣons to
interact with the API. - AddiΘonal Features (OpΘonal for Extra Credit): - Implement paginaΘon for product lisΘng

- Add product search by name or category.

Solution

Simple E-commerce API using Node.js, Express, MongoDB, and JWT

```
const express = require('express');
 const mongoose = require('mongoose');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
 const bodyParser = require('body-parser');
const cors = require('cors');
require('dotenv').config();
const app = express();
app.use(bodyParser.json());
app.use(cors());
```

MongoDB Connection

```
 mongoose.connect('mongodb://localhost/ecommerce-api', { useNewUrlParser: true, useUnifiedTopology: true });
```

Models

```
 const UserSchema = new mongoose.Schema({
                        username: String,
                        password: String,
                         role: { type: String, enum: ['customer', 'admin'], default: 'customer' }, });
const ProductSchema = new mongoose.Schema({
                        name: String,
                        category: String,
                        price: Number,
                        description: String
 });
const CartSchema = new mongoose.Schema({
     userId: mongoose.Schema.Types.ObjectId,
     items: [{ productId: mongoose.Schema.Types.ObjectId, quantity: Number }]
 });
const OrderSchema = new mongoose.Schema({
    userId: mongoose.Schema.Types.ObjectId,
    items: [{ productId: mongoose.Schema.Types.ObjectId, quantity: Number }],
    createdAt: { type: Date, default: Date.now }
});

const User = mongoose.model('User', UserSchema);
 const Product = mongoose.model('Product', ProductSchema);
const Cart = mongoose.model('Cart', CartSchema);
const Order = mongoose.model('Order', OrderSchema);
```

Middleware

```
 function authMiddleware(req, res, next) {
  const token = req.headers['authorization'];
  if (!token) return res.status(403).send('Token required');
  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) return res.status(401).send('Invalid token');
    req.user = user;
     next(); }); }
```

```
function adminOnly(req, res, next) {
   if (req.user.role !== 'admin') return res.status(403).send('Admins only');
    next(); }
```

Routes

```
 app.post('/register', async (req, res) => {
    const { username, password, role } = req.body;
    const hash = await bcrypt.hash(password, 10);
    const user = new User({ username, password: hash, role });
    await user.save();
     res.send('User registered');
 });
app.post('/login', async (req, res) => {
  const { username, password } = req.body;
  const user = await User.findOne({ username });
if (!user || !await bcrypt.compare(password, user.password)) return res.status(401).send('Invalid credentials');
const token = jwt.sign({ id: user._id, role: user.role }, process.env.JWT_SECRET);
res.json({ token }); });
```

Products

```
app.get('/products', async (req, res) => {
   const { page = 1, limit = 10, search = '', category = '' } = req.query;
   const query = {
    name: { $regex: search, $options: 'i' },
   ...(category && { category })
 }; const products = await Product.find(query).limit(limit * 1).skip((page - 1) * limit);
res.json(products);
});
app.post('/products', authMiddleware, adminOnly, async (req, res) => { const product = new Product(req.body);
await product.save();
res.send('Product added');
});
app.put('/products/:id', authMiddleware, adminOnly, async (req, res) => {
  await Product.findByIdAndUpdate(req.params.id, req.body);
  res.send('Product updated');
});
app.delete('/products/:id', authMiddleware, adminOnly, async (req, res) => {
   await Product.findByIdAndDelete(req.params.id);
   res.send('Product deleted');
 });
```

Cart

```
app.get('/cart', authMiddleware, async (req, res) => {
   const cart = await Cart.findOne({ userId: req.user.id }).populate('items.productId');
   res.json(cart);
 });

app.post('/cart', authMiddleware, async (req, res) => {
const { productId, quantity } = req.body;
 let cart = await Cart.findOne({ userId: req.user.id });
 if (!cart) cart = new Cart({ userId: req.user.id, items: [] });
const item = cart.items.find(item => item.productId.equals(productId));
if (item) item.quantity += quantity;
 else cart.items.push({ productId, quantity });
 await cart.save();
 res.send('Cart updated');
 });

app.delete('/cart/:productId', authMiddleware, async (req, res) => {
 let cart = await Cart.findOne({ userId: req.user.id });
 cart.items = cart.items.filter(item => !item.productId.equals(req.params.productId));
await cart.save();
res.send('Item removed'); });
```

Order

```
app.post('/order', authMiddleware, async (req, res) => {
const cart = await Cart.findOne({ userId: req.user.id }); .
if (!cart || cart.items.length === 0) return res.status(400).send('Cart is empty');
const order = new Order({ userId: req.user.id, items: cart.items });
await order.save();
 await Cart.findOneAndDelete({ userId: req.user.id });
res.send('Order placed'); });
```

Start Server

```
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(Server running on port ${PORT}));
```