mkdir stock-management

cd stock-management

mkdir backend frontend

cd backend

npm init -y

npm install express mongoose cors dotenv helmet bcrypt jsonwebtoken

npm install -D nodemon

cd ../frontend

npx create-react-app client

cd client

npm install axios react-router-dom

// models/Product.js

const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({

name: { type: String, required: true },

description: { type: String },

sku: { type: String, required: true, unique: true },

category: { type: String, required: true},

price: { type: Number, required: true },

image: { type: String},

stockQuantity: { type: Number, default: 0 },

createdAt: { type: Date, default: Date.now },

});

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

// models/Category.js

const mongoose = require('mongoose');

const categorySchema = new mongoose.Schema({

name: { type: String, required: true, unique: true },

createdAt: { type: Date, default: Date.now },

});

module.exports = mongoose.model('Category', categorySchema);

// models/User.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

username: { type: String, required: true, unique: true },

password: { type: String, required: true },

role: { type: String, enum: ['admin', 'user'], default: 'user' },

createdAt: { type: Date, default: Date.now },

});

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

// models/StockMovement.js

const mongoose = require('mongoose');

const stockMovementSchema = new mongoose.Schema({

product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },

quantityChange: { type: Number, required: true },

type: { type: String, enum: ['addition', 'removal'], required: true },

createdAt: { type: Date, default: Date.now },

});

module.exports = mongoose.model('StockMovement', stockMovementSchema);

const mongoose = require('mongoose');

const connectDB = async () => {

try {

await mongoose.connect(process.env.MONGO\_URI);

console.log("MongoDB Connected...");

} catch (err) {

console.error(err.message);

//Exit process with failure

process.exit(1);

}

};

module.exports = connectDB;

// server.js

const express = require('express');

const cors = require('cors');

const helmet = require('helmet');

const dotenv = require('dotenv')

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

const productRoutes = require('./routes/products');

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

const categoryRoutes = require('./routes/categories')

dotenv.config()

const app = express();

connectDB();

app.use(cors());

app.use(helmet());

app.use(express.json());

app.use('/api/products', productRoutes);

app.use('/api/users', userRoutes);

app.use('/api/categories', categoryRoutes)

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

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

const express = require('express');

const router = express.Router();

const {

getProducts,

getProductById,

createProduct,

updateProduct,

deleteProduct,

addStock,

removeStock

} = require('../controllers/productController');

const { authenticate, authorize } = require('../middleware/authMiddleware');

router.get('/', authenticate, getProducts);

router.get('/:id', authenticate, getProductById);

router.post('/', authenticate,authorize('admin'),createProduct);

router.put('/:id',authenticate,authorize('admin'), updateProduct);

router.delete('/:id', authenticate,authorize('admin'), deleteProduct);

router.put('/:id/add', authenticate,authorize('admin'),addStock);

router.put('/:id/remove', authenticate,authorize('admin'), removeStock);

module.exports = router;

const express = require('express');

const router = express.Router();

const {

getCategories,

createCategory,

} = require('../controllers/categoryController');

const { authenticate, authorize } = require('../middleware/authMiddleware');

router.get('/', authenticate, getCategories);

router.post('/',authenticate, authorize('admin'), createCategory);

module.exports = router;

const express = require('express');

const router = express.Router();

const {

registerUser,

loginUser,

} = require('../controllers/userController');

router.post('/register', registerUser);

router.post('/login', loginUser);

module.exports = router;

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

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

exports.getProducts = async (req, res) => {

try {

const products = await Product.find();

res.json(products);

} catch (error) {

res.status(500).json({ message: 'Server Error' });

}

};

exports.getProductById = async (req, res) => {

try {

const product = await Product.findById(req.params.id);

if (!product) {

return res.status(404).json({ message: 'Product not found' });

}

res.json(product);

} catch (error) {

if (error.kind === 'ObjectId') {

return res.status(404).json({ message: 'Product not found' });

}

res.status(500).json({ message: 'Server Error' });

}

};

exports.createProduct = async (req, res) => {

try {

const newProduct = new Product(req.body);

await newProduct.save();

res.status(201).json({message: 'Product added', newProduct});

} catch (error) {

if (error.code === 11000) {

return res.status(400).json({ message: 'SKU already exists' });

}

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

}

};

exports.updateProduct = async (req, res) => {

try {

const updatedProduct = await Product.findByIdAndUpdate(

req.params.id,

req.body,

{ new: true }

);

if (!updatedProduct) {

return res.status(404).json({ message: 'Product not found' });

}

res.json({message: "Product updated successfully", updatedProduct});

} catch (error) {

if (error.kind === 'ObjectId') {

return res.status(404).json({ message: 'Product not found' });

}

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

}

};

exports.deleteProduct = async (req, res) => {

try {

const product = await Product.findByIdAndDelete(req.params.id);

if (!product) {

return res.status(404).json({ message: 'Product not found' });

}

res.status(204).json();

} catch (error) {

if (error.kind === 'ObjectId') {

return res.status(404).json({ message: 'Product not found' });

}

res.status(500).json({ message: 'Server error' });

}

};

exports.addStock = async (req, res) => {

try {

const product = await Product.findById(req.params.id);

if (!product) {

return res.status(404).json({ message: 'Product not found' });

}

const {quantity} = req.body;

if (!quantity || quantity <= 0) {

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

}

product.stockQuantity += Number(quantity)

await product.save();

const newStockMovement = new StockMovement({

product: product.\_id,

quantityChange: quantity,

type: 'addition',

});

await newStockMovement.save();

res.json({message:'Stock added successfully', product});

} catch (error) {

if (error.kind === 'ObjectId') {

return res.status(404).json({ message: 'Product not found' });

}

res.status(500).json({ message: 'Server error' });

}

};

exports.removeStock = async (req, res) => {

try {

const product = await Product.findById(req.params.id);

if (!product) {

return res.status(404).json({ message: 'Product not found' });

}

const {quantity} = req.body;

if (!quantity || quantity <= 0) {

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

}

if(product.stockQuantity < quantity){

return res.status(400).json({message:"Not enough stock available"})

}

product.stockQuantity -= Number(quantity)

await product.save();

const newStockMovement = new StockMovement({

product: product.\_id,

quantityChange: quantity,

type: 'removal',

});

await newStockMovement.save();

res.json({message:'Stock removed successfully', product});

} catch (error) {

if (error.kind === 'ObjectId') {

return res.status(404).json({ message: 'Product not found' });

}

res.status(500).json({ message: 'Server error' });

}

};

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

exports.getCategories = async (req, res) => {

try {

const categories = await Category.find();

res.json(categories);

} catch (error) {

res.status(500).json({ message: 'Server Error' });

}

};

exports.createCategory = async (req, res) => {

try {

const newCategory = new Category(req.body);

await newCategory.save();

res.status(201).json({message: 'Category added', newCategory});

} catch (error) {

if (error.code === 11000) {

return res.status(400).json({ message: 'Category already exists' });

}

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

}

};

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

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

exports.registerUser = async (req, res) => {

try {

const { username, password } = req.body;

const userExists = await User.findOne({ username });

if(userExists){

return res.status(400).json({message: 'Username already exists'})

}

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

const user = new User({ username, password: hashedPassword });

await user.save();

res.status(201).json({message:'User registered successfully', user});

} catch (error) {

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

}

};

exports.loginUser = async (req, res) => {

try {

const { username, password } = req.body;

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

if (!user) {

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

}

const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) {

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

}

const payload = {

user:{

id: user.id,

role: user.role,

}

};

const token = jwt.sign(

payload,

process.env.JWT\_SECRET,

{expiresIn: '1h'}

);

res.json({message:'Logged in successfully', token, user});

} catch (error) {

res.status(500).json({ message: 'Server error' });

}

};

const jwt = require('jsonwebtoken');

exports.authenticate = (req, res, next) => {

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

if (!token) {

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

}

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.user = decoded.user;

next();

} catch (err) {

res.status(401).json({ msg: 'Token is not valid' });

}

};

exports.authorize = (role) => {

return (req, res, next) => {

if (req.user && req.user.role === role) {

next();

} else {

res.status(403).json({ message: 'Forbidden' });

}

};

};

// services/api.js

import axios from 'axios';

const api = axios.create({

baseURL: 'http://localhost:5000/api', // Backend URL

});

export const setAuthToken = (token) =>{

if(token){

api.defaults.headers.common['x-auth-token'] = token;

}else{

delete api.defaults.headers.common['x-auth-token']

}

}

export default api;

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

import api from '../services/api';

import ProductCard from '../components/ProductCard';

const ProductList = () => {

const [products, setProducts] = useState([]);

useEffect(() => {

const fetchProducts = async () => {

try {

const response = await api.get('/products');

setProducts(response.data);

} catch (error) {

console.error('Error fetching products:', error);

}

};

fetchProducts();

}, []);

return (

<div>

<h2>Product List</h2>

<div className='product-list'>

{products.map((product) => (

<ProductCard key={product.\_id} product={product} />

))}

</div>

</div>

);

};

export default ProductList;

// App.js

import React from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import Login from './pages/Login';

import Register from './pages/Register';

import ProductList from './pages/ProductList';

import ProductCreate from './pages/ProductCreate';

import Categories from './pages/Categories';

import AddCategory from './pages/AddCategory';

import Dashboard from './pages/Dashboard';

import ProductEdit from './pages/ProductEdit';

import { AuthProvider } from './context/AuthContext';

import PrivateRoute from './components/PrivateRoute';

const App = () => {

return (

<AuthProvider>

<Router>

<Routes>

<Route path="/login" element={<Login />} />

<Route path="/register" element={<Register />} />

<Route path="/" element={<PrivateRoute><Dashboard /></PrivateRoute>}/>

<Route path="/products" element={<PrivateRoute> <ProductList /></PrivateRoute>}/>

<Route path="/products/create" element={<PrivateRoute> <ProductCreate /></PrivateRoute>}/>

<Route path="/products/edit/:id" element={<PrivateRoute> <ProductEdit /></PrivateRoute>}/>

<Route path="/categories" element={<PrivateRoute><Categories /></PrivateRoute>} />

<Route path="/categories/create" element={<PrivateRoute><AddCategory /></PrivateRoute>} />

</Routes>

</Router>

</AuthProvider>

);

};

export default App;