E-Commerce Project Documentation

Project Overview

This project expands an existing Inventory Management System (IMS) into a full-featured e-commerce platform. Built with Node.js, Express.js, MongoDB, React, and Tailwind CSS, it includes inventory management, user authentication, product listings, and enhanced functionalities to support an e-commerce flow. The backend is deployed on Render, and the frontend on Netlify.

---

1. Existing Project Structure and Setup

Backend (Node.js, Express.js, MongoDB)

- The backend server is based on Express.js for routing and MongoDB with Mongoose for database management.
- It includes **user authentication**, **product management**, **category management**, and **middleware** for securing routes.
- **Environment Variables** (stored in `server/.env`):
  - `MONGO_URI`: MongoDB connection URI.
  - `JWT_SECRET`: Secret for JWT signing.
- **Key Commands**:
  - Start the server: `npm start` or `npm run server` (runs nodemon for development).
  - Base URL: `localhost:4000`.

Frontend (React with Tailwind CSS)

- Designed to display products, categories, and user profiles, with integration for secure routes.
- **Commands**:
  - Start the frontend: `npm start`.
  - Base URL: `localhost:3000`.

---

2. Existing Functionalities

User Authentication and Profile Management

- **User Registration and Login**: Users can register and log in, with passwords hashed using bcrypt for secure storage.

- **JWT-Based Authentication**: A token-based system ensures secure access to protected resources.
- **Profile Management**:
    - Users can view their profile details and update their information.

Product and Category Management

- **Product CRUD**: Admins can create, read, update, and delete products, each with fields like name, category, price, and stock level.
- **Category CRUD**: Admins can manage categories, with each product linked to a category by ID.
- **Stock Management**:
    - **Low Stock Notification**: Admins receive alerts when products fall below a certain stock level.
- **Product Listing**:
    - All products can be retrieved, with options to filter by category.

Error Handling and Security

- Standardized error handling across routes, with common error responses:
    - **401 Unauthorized**: For missing or invalid JWT tokens.
    - **404 Not Found**: When requested resources (e.g., products, categories) don't exist.
- **Password Hashing and Verification**: All passwords are stored securely using bcrypt hashing.
- **JWT Validation**: Tokens are signed with `JWT_SECRET` for verifying user authenticity on secure routes.

---

3. New Functionalities for E-Commerce Expansion

To expand the project into a fully functional e-commerce platform, the following features will be added:

3.1 User Features

Cart System

- **Add to Cart**: Users can add products to their cart from the product listing or detail page.
- **Cart Quantity Adjustment**: Users can update quantities or remove items from the cart.
- **Persisted Cart State**: Save the cart state for logged-in users, allowing them to return to their cart later.

- **Total Price Calculation**: Display the total price in the cart, automatically updating as items are added or removed.

Checkout and Payment Options

- **Checkout Flow**:
    - Upon checkout, users review their order, choose a payment method, and confirm.
- **Payment Gateway Integration**:
    - Stripe integration for online payments with secure payment processing.
    - **Cash on Delivery**: Option for users to pay upon delivery.

Shipping Address Management

- **Multiple Shipping Addresses**: Users can save multiple shipping addresses and select one during checkout.
- **Address CRUD**: Users can add, edit, or delete addresses from their profile.

Profile Management

- **Profile Page**:
    - View and edit personal information (name, email, contact info).
    - Manage multiple addresses for future orders.

Search and Filtering

- **Search Functionality**: Keyword-based search across product names and descriptions.
- **Sorting Options**:
    - Sort search results by date added, ratings, reviews, and price.
- **Filter by Availability**:
    - Mark products as “Out of Stock” if they have zero quantity.
    - Display only in-stock items as default, with an option to view out-of-stock products.

Product Ratings and Reviews

- **Rating System**: Enable users to rate products on a scale of 1-5.
- **Reviews**:
    - Users can leave written reviews with each rating.
    - Display average ratings and total reviews on each product page.

3.2 Admin Features

Enhanced Product Management

- **Product Details**:
  - Add fields for product images, average ratings, and reviews.
- **Low Stock Notifications**:
  - Existing low stock notifications for admins will be retained.
  - Allow admins to set custom low-stock thresholds per product.

Order Management

- **View Orders**: Admins can access a dashboard showing all placed orders.
- **Order Status Management**: Ability to update order statuses (e.g., pending, shipped, delivered).

3.3 General Functionalities

User Notifications

- **Order Confirmation Emails**: Send confirmation emails upon order placement.
- **Order Status Updates**: Notify users by email when the status of their order changes.

Deployment and Testing

- **Backend Deployment**: Deploy to Render for a managed Node.js environment, with MongoDB Atlas for remote database access.
- **Frontend Deployment**: Deploy to Netlify, setting environment variables to connect with the backend seamlessly.

---

4. Database Schema Enhancements

To support the new e-commerce functionalities, additional schema updates are needed:

User Schema

- `username`: Unique login identifier.
- `password`: Hashed with bcrypt.
- `isAdmin`: Boolean for role-based access control.
- `addresses`: Array to store multiple shipping addresses.

Product Schema

- `name`: Product name.
- `category`: Reference to Category model.

- `price`: Product price.
- `stock_level`: Tracks stock quantity.
- **New Fields**:
    - `images`: Array to store URLs or paths to product images.
    - `ratings`: Number field to track average ratings.
    - `reviews`: Array to store review references.

Review Schema

- `user`: Reference to User model.
- `rating`: Rating from 1-5.
- `comment`: User's comment on the product.
- `timestamp`: Date when the review was created.

Order Schema

- `user`: Reference to User model.
- `products`: Array of products ordered, with details such as quantity and individual price.
- `total_amount`: Total order cost.
- `payment_status`: Tracks if payment is complete.
- `shipping_address`: Selected address from the user’s profile.
- `status`: Tracks order progress (e.g., pending, shipped, delivered).

---

5. API Endpoints and Controller Details

The following new and modified endpoints will be required:

User Endpoints

- `POST /api/user/register`: Register a new user.
- `POST /api/user/login`: User login.
- `GET /api/user/profile`: Retrieve profile data.
- `PUT /api/user/profile`: Update profile and manage addresses.

Product Endpoints

- `GET /api/products`: Fetch all products with sorting and filter options.
- `POST /api/products/:id/review`: Add review and rating to a product.
- `GET /api/products/search`: Search products by keyword.

Cart and Order Endpoints

- `POST /api/cart`: Add item to the cart.

- `GET /api/cart`: Retrieve items in cart.
- `POST /api/order`: Place an order and proceed to checkout.
- `GET /api/order/:id`: Retrieve specific order details.

Admin Endpoints

- `GET /api/admin/products/low-stock`: Retrieve low-stock products.
- `GET /api/admin/orders`: Access all orders for tracking.

---

6. Testing and Quality Assurance

To ensure system reliability, the following testing plan is proposed:

- **Unit Testing**: Test all CRUD operations, authentication, and business logic.
- **Integration Testing**: Test end-to-end workflows, including search, add-to-cart, checkout, and payment.
- **Payment Gateway Testing**: Verify payment flow using Stripe's sandbox environment.
- **User Interface Testing**: Confirm that the UI functions as expected across multiple devices.

---

7. Deployment

- **Backend**: Deploy on Render using MongoDB Atlas for database access.
- **Frontend**: Deploy on Netlify with environment configurations to connect with the backend.