Technical Assignment for NestJS Backend Developer Position at Dinamo MEA

Mohamed Amir
mohamedamirr424@gmail.com
01001733917

Overview

This document outlines the technical assignment for developing a backend application using **NestJS** and **MongoDB**. The focus is on designing a database schema for core entities, implementing the schemas/models in a NestJS application, and proposing an authentication strategy.

Objectives

1. Design a Database Schema for the core entities:
    - Product
    - Vendor
    - User
    - Cart
2. Create a NestJS Application:
    - Set up a basic NestJS app.
    - Implement the schemas/models for the core entities.
3. List REST Endpoints:
    - Provide a list of RESTful API endpoints that would interact with these entities.
4. Propose an Authentication Solution suitable for serving multiple applications and user types.

Core Entities

The core entities for the application are as follows:

1. **User**
    - Represents application users, including their roles (admin, vendor, customer).
2. **Vendor**
    - Represents vendors who sell products in the application.
3. **Product**

    - Represents products available for purchase.
4. **Cart**
    - Represents the shopping cart functionality for users.

Extra Modules

In addition to core entities, extra modules can enhance the application:

1. **Order**
    - Represents user orders.
2. **Wishlist**
    - Represents user wishlists for saving favorite products.
3. **Review**
    - Represents product reviews written by users.

Entity Relationship Diagram (ERD)

The database schema design is based on the core entities and their relationships as outlined in the ERD. The schema is defined in Mongoose models and will include:

- **User Model**: Contains user attributes and relationships to cart, wishlist, and orders.
- **Vendor Model**: Contains vendor attributes and relationships to products.
- **Product Model**: Contains product attributes and relationships to vendor, cart, and reviews.
- **Cart Model**: Contains cart attributes and relationships to user and products.
- **Order Model**: Contains order attributes and relationships to user and products.
- **Wishlist Model**: Contains wishlist attributes and relationships to user and products.
- **Review Model**: Contains review attributes and relationships to user and product.

1. **User Entity**:
    - **Attributes**:
        - _id: ObjectID (Primary Key)
        - name: String
        - email: String (Unique)
        - password: String (Hashed for security)
        - role: String (e.g., customer, admin, vendor)
        - cart: Cart (One-to-One relation)
        - wishlist: [Product] (Many-to-Many relation)
        - orders: [Order] (One-to-Many relation)
    - **Relationships**:

        - Each user can have one cart.
        - Each user can have many orders.
        - Each user can have many products in their wishlist.

2. **Vendor Entity**:
    - **Attributes**:
        - _id: ObjectID (Primary Key)
        - name: String
        - email: String (Unique)
        - products: [Product] (One-to-Many relation)
    - **Relationships**:
        - Each vendor can list many products.

3. **Product Entity**:
    - **Attributes**:
        - _id: ObjectID (Primary Key)
        - name: String
        - price: Number
        - description: String
        - category: String
        - vendor: Vendor (Many-to-One relation)
        - stock: Number (Optional, for inventory management)
        - ratings: [Review] (One-to-Many relation)
    - **Relationships**:
        - Each product belongs to a vendor.
        - Each product can have many ratings (if the review module is included).
        - Each product can appear in many carts and wishlists.

4. **Cart Entity**:
    - **Attributes**:
        - _id: ObjectID (Primary Key)
        - user: User (One-to-One relation)

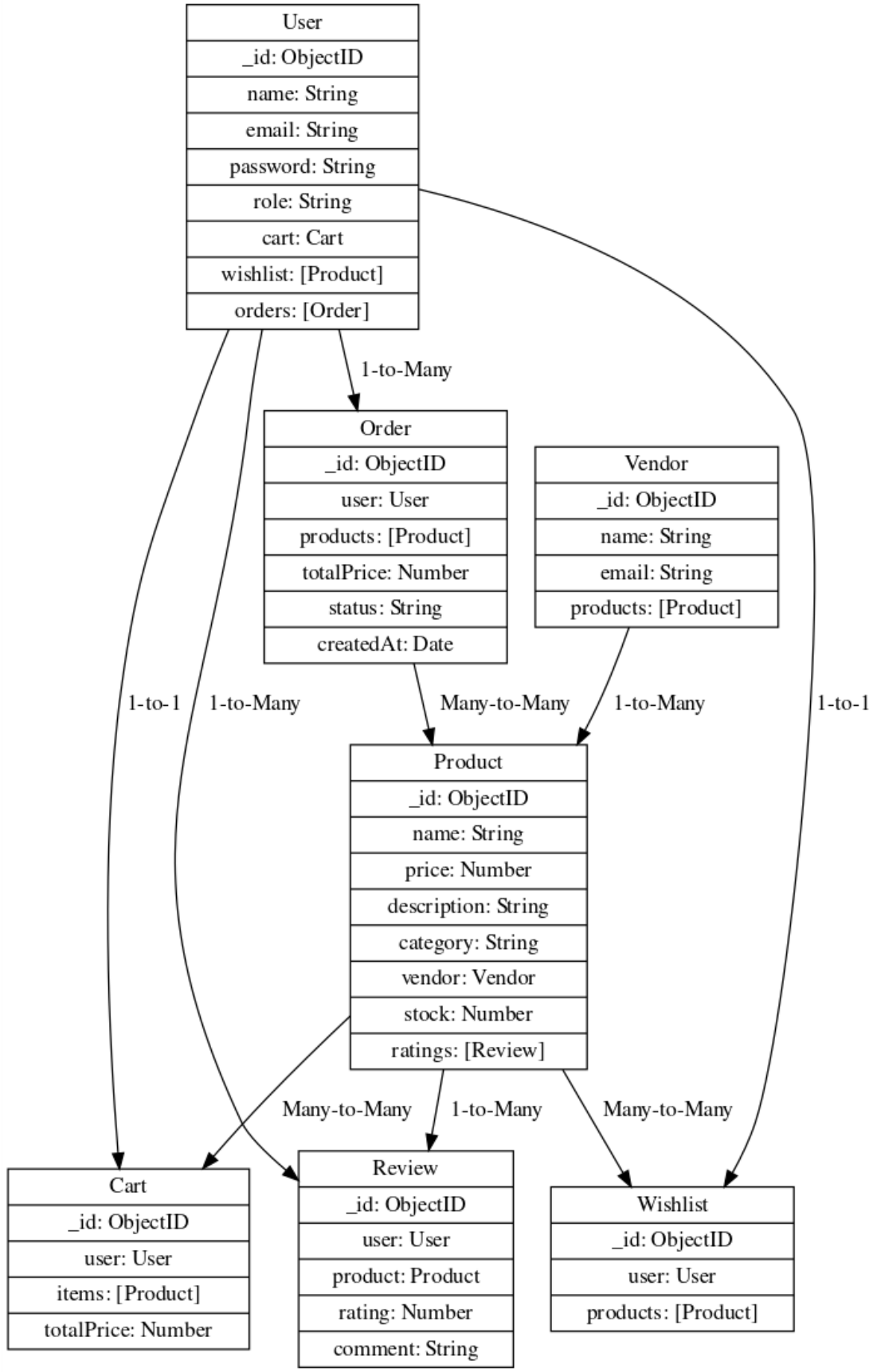
        - items: [{ product: Product, quantity: Number }] (Many-to-Many relation with products)
        - totalPrice: Number (Calculated field)
    - **Relationships**:
        - A cart belongs to one user.
        - A cart contains many products.

5. **Order Entity** (Extra Module):
    - **Attributes**:
        - _id: ObjectID (Primary Key)
        - user: User (Many-to-One relation)
        - products: [{ product: Product, quantity: Number }]
        - totalPrice: Number (Calculated field)
        - status: String (e.g., pending, completed, shipped)
        - createdAt: Date
    - **Relationships**:
        - An order belongs to one user.
        - An order contains many products.
6. **Wishlist Entity** (Extra Module):
    - **Attributes**:
        - _id: ObjectID (Primary Key)
        - user: User (One-to-One relation)
        - products: [Product] (Many-to-Many relation)
    - **Relationships**:
        - Each user can have a wishlist containing many products.
7. **Review Entity** (Extra Module):
    - **Attributes**:
        - _id: ObjectID (Primary Key)
        - user: User (Many-to-One relation)
        - product: Product (Many-to-One relation)
        - rating: Number

        - comment: String
    - **Relationships**:
        - Each review is linked to a user and a product.

Relationships Summary:

- **User ↔ Cart**: One-to-One (Each user has one cart).
- **User ↔ Order**: One-to-Many (Each user can have multiple orders).
- **User ↔ Wishlist**: One-to-One (Each user has one wishlist containing many products).
- **Vendor ↔ Product**: One-to-Many (Each vendor can list many products).
- **Product ↔ Cart**: Many-to-Many (Products can be added to many carts).
- **Product ↔ Wishlist**: Many-to-Many (Products can be added to many wishlists).
- **Product ↔ Review**: One-to-Many (A product can have many reviews).
- **User ↔ Review**: One-to-Many (A user can write many reviews).


User
_id: ObjectID
name: String
email: String
password: String
role: String
cart: Cart
wishlist: [Product]
orders: [Order]
1-to-Many
Order
_id: ObjectID
user: User
products: [Product]
totalPrice: Number
status: String
createdAt: Date
Vendor
_id: ObjectID
name: String
email: String
products: [Product]
1-to-1
1-to-Many
Many-to-Many
1-to-Many
1-to-1
Product
_id: ObjectID
name: String
price: Number
description: String
category: String
vendor: Vendor
stock: Number
ratings: [Review]
Many-to-Many
1-to-Many
Many-to-Many
Cart
_id: ObjectID
user: User
items: [Product]
totalPrice: Number
Review
_id: ObjectID
user: User
product: Product
rating: Number
comment: String
Wishlist
_id: ObjectID
user: User
products: [Product]

RESTful API Endpoints

User Endpoints

- POST /users: Register a new user.
- GET /users/:id: Retrieve user details.
- PUT /users/:id: Update user details.
- DELETE /users/:id: Delete a user.

Vendor Endpoints

- POST /vendors: Create a new vendor.
- GET /vendors/:id: Retrieve vendor details.
- GET /vendors: Retrieve all vendors.

Product Endpoints

- POST /products: Create a new product.
- GET /products/:id: Retrieve product details.
- GET /products: Retrieve all products.
- PUT /products/:id: Update product details.
- DELETE /products/:id: Delete a product.

Cart Endpoints

- GET /carts/:userId: Retrieve the user's cart.
- POST /carts: Add product to cart.
- PUT /carts/:userId: Update cart items.
- DELETE /carts/:userId: Clear the cart.

Order Endpoints (Extra)

- POST /orders: Create a new order.
- GET /orders/:userId: Retrieve user orders.

Wishlist Endpoints (Extra)

- GET /wishlists/:userId: Retrieve user's wishlist.
- POST /wishlists: Add product to wishlist.
- DELETE /wishlists/:userId: Clear wishlist.

Review Endpoints (Extra)

- POST /reviews: Add a review for a product.
- GET /reviews/:productId: Retrieve all reviews for a product.

Method	Endpoint	Description
POST	/users	Register a new user.
GET	/users/:id	Retrieve user details.
PUT	/users/:id	Update user details.
DELETE	/users/:id	Delete a user.
POST	/vendors	Create a new vendor.
GET	/vendors/:id	Retrieve vendor details.
GET	/vendors	Retrieve all vendors.
POST	/products	Create a new product.
GET	/products/:id	Retrieve product details.
GET	/products	Retrieve all products.
PUT	/products/:id	Update product details.
DELETE	/products/:id	Delete a product.
GET	/carts/:userId	Retrieve the user's cart.
POST	/carts	Add product to cart.
PUT	/carts/:userId	Update cart items.
DELETE	/carts/:userId	Clear the cart.
POST	/orders	Create a new order.
GET	/orders/:userId	Retrieve user orders.
GET	/wishlists/:userId	Retrieve user's wishlist.
POST	/wishlists	Add product to wishlist.
DELETE	/wishlists/:userId	Clear wishlist.
POST	/reviews	Add a review for a product.
GET	/reviews/:productId	Retrieve all reviews for a product.

Authentication Strategy

Overview

The authentication strategy will ensure secure access to the API for different user types (customers, vendors, and admins). The following approach will be implemented:

1. **User Registration**: Users will register by providing their email and password. Passwords will be hashed using bcrypt before being stored in the database.
2. **User Login**: Users will log in with their email and password. Upon successful login, a JSON Web Token (JWT) will be issued.
3. **Token-Based Authentication**:
    - JWTs will be used to authenticate requests to protected routes.
    - Each request to a protected endpoint must include the JWT in the `Authorization` header.
4. **Authorization Middleware**: Middleware will check the validity of the JWT and ensure that the user has the required permissions to access specific resources.
5. **Role-Based Access Control (RBAC)**:
    - Different roles (admin, vendor, customer) will have different permissions.
    - Admins will have full access, vendors will manage their products, and customers will be able to view and purchase products.

Development Instructions

1. **Set Up NestJS Application**: Initialize a new NestJS project using the CLI.

   ```
   nest new dinamo
   cd dinamo
   ```

2. **Install Required Packages**: Install Mongoose and other necessary dependencies.

   ```
   npm install @nestjs/mongoose mongoose bcrypt jsonwebtoken @nestjs/passport passport passport-jwt
   ```

3. **Create Mongoose Models**: Implement the defined schemas for core entities and extra modules.
4. **Implement REST API Endpoints**: Create controllers and services for each entity and implement the defined API endpoints.
5. **Implement Authentication**: Set up authentication using JWT, including registration, login, and middleware.

Testing

- Use Jest for unit testing of all functions and API endpoints.
- Ensure all endpoints are thoroughly tested, including positive and negative test cases.