Deadline: 2 Sep, Monday

Implement Product Management

Objective: The development of a product management feature within a Node.js application. The feature enables authenticated users to create, update, delete, and retrieve products. Products will include a name, description, price, category, and up to 5 images. Each product is associated with a single category. MongoDB is used for data storage, Mongoose for data modeling, and JWT for user authentication. Input data validation and image file handling are key components.

Scenario:

1. **User Authentication:**
    - Implement JWT-based authentication to secure the product management routes. Users must be logged in to access these functionalities.
    - Use middleware to verify the JWT token in the Authorization header of requests to protected routes.
2. **Routes:**

    - **POST /products:** Create a new product. Requires authentication. Expects a name, description, price, category, and up to 5 image files. The request body should include name, description, price, category, and up to 5 images.

    - **PUT /products/:id:** Update an existing product by its id. Requires authentication. Allows updating the name, description, price, category, and images. Validate that the request body contains valid fields and that no more than 5 images are uploaded.

    - **DELETE /products/:id:** Delete an existing product by its id. Requires authentication. Removes the product from the database. Ensure that the product exists before deletion.

    - **GET /products/user/:userId:** Retrieve all products created by a specific user. Requires authentication. Validate that the userId parameter is a valid MongoDB ObjectId.

- **GET /products/category/:categoryId:** Retrieve all products within a specific category. Allows users to filter products by category. Validate that the categoryId parameter is a valid MongoDB ObjectId.

- **GET /products/:id:** Retrieve detailed information about a specific product by its id. Includes all associated images.

- **GET /users/products:** Retrieve detailed information about all products created by a specific user. Requires authentication.

- **GET /products:** Retrieve a list of all products. Admins and other authenticated users can view this list.

3. **Validation:**

- Implement validation for product creation and update requests using a validation library Ensure that name, description, price, category, and image fields are valid.
- Validate that id, **userId**, and **categoryId** parameters are valid MongoDB ObjectIds.

4. **File Upload:**

- Handle image uploads in both product creation and update scenarios. Ensure that uploaded images are local storage, and that image URLs are stored in MongoDB.

5. **Data Storage:**

- Use Mongoose to define a Product schema with fields for name, description, price, category, and an array of image URLs. Include a reference to the user who created the product.
- Define a Category schema with name and an array of product references.
- Define a user schema with username, email, password, and an array of product references.

6. **Error Handling:**

- Handle errors such as invalid authentication, validation errors, file upload issues, non-existent products, invalid userId, categoryId, and issues with relations. Return appropriate HTTP status codes and error messages.

Example Workflow:

7. **User Authentication:**
    - User logs in and receives a JWT token.
8. **Create Product:**
    - Authenticated user sends a POST request to /products with a JWT token, name, description, price, category, and up to 5 image files.
    - Server validates the input, processes the images, saves them to storage, and creates a new product entry in MongoDB, linking it to the authenticated user and category.
9. **Update Product:**
    - Authenticated user sends a PUT request to /products/:id with a JWT token, name, description, price, category, and up to 5 image files.
    - Server validates the input, processes new or updated images, updates the existing product entry in MongoDB, and ensures the total number of images does not exceed 5.
10. **Delete Product:**
    - Authenticated user sends a DELETE request to /products/:id with a JWT token.

- Server verifies the JWT token, checks that the product exists, and removes the product from MongoDB.

11. **Get Products by User:**
    - User sends a GET request to /products/user/:userId with a JWT token.
    - Server verifies the JWT token, retrieves all products associated with the specified userId from MongoDB.
12. **Get Products by Category:**
    - User sends a GET request to /products/category/:categoryId with a JWT token.
    - Server verifies the JWT token, retrieves products within the specified category from MongoDB.
13. **Get Product Details:**
    - User sends a GET request to /products/:id with a JWT token.
    - Server retrieves the product by its id, populates user and category details, and returns product information.
14. **Get User's Products:**
    - User sends a GET request to /users/:userId/products with a JWT token.
    - Server retrieves all products created by the specified user, including user details.

15. **Get All Products:**
    - User sends a GET request to /products with a JWT token.
    - Server retrieves all products from MongoDB.
16. **Error Handling:**

- Handle errors related to authentication, validation, product management, and relationships. Return appropriate status codes and messages.

Joins and Relations:

- **Product Schema:**
    - Include a reference to the User and Category schemas.
    - Populate the user field to include user details when retrieving a product.
- **User Schema:**
    - Define a products field that stores product references.
- **Category Schema:**
    - Define a products field that stores product references.

- Use Mongoose's populate method to retrieve related user and category information.

Visual Representation of Database Collections

User Collection

Field	Type	Description
_id	ObjectId	Primary key **(Auto Generate)**
name	String	User's Name
email	String	User's Email
password	String	User's Password
products	[ObjectId]	Array of product references

Category Collection

Field	Type	Description
_id	ObjectId	Primary key **(Auto Generate)**
name	String	Name of Category
products	[ObjectId]	Array of product references

Product Collection

Field	Type	Description
_id	ObjectId	Primary key **(Auto Generate)**
name	String	Product name
description	String	Product description
price	Number	Product price
category	ObjectId	Reference to the category

images	[String]	Array of image URLs
user	ObjectId	Reference to the user who created the product

This layout shows the relationships between collections:

- **User Collection**: Stores user details and an array of product references.
- **Category Collection**: Stores category details and an array of product references.
- **Product Collection**: Contains details of each product, with references to the associated user and a single category. It also includes an array of image URLs.

Acceptance Criteria:

- Routes are secured with JWT authentication.
- Input data and image uploads are validated.
- Products can be managed (created, updated, deleted, viewed).
- Each product can belong to a single category.
- Images are stored and referenced correctly.
- User and category information is populated and retrieved.
- Errors are handled with appropriate responses.