Exercise: Building a Product Module with Node.js

Objective:

The aim of this exercise is to develop a Product module for a web application using Node.js, Express.js. The module should cover CRUD (Create, Read, Update, Delete) operations on Product records, product image upload, input validation, authentication using JWT (JSON Web Tokens), ORM usage with Sequelize / Mongoose, and error handling.

Requirements:

1. Develop a RESTful API for managing Product records.
2. Implement user registration and login for authentication and then CRUD operations for Product records using that authentication.
3. Implement input validation and sanitization.
4. Encrypt and securely store passwords using bcrypt.
5. Utilize either MySQL with Sequelize ORM or MongoDB with Mongoose for storing Product records. Participants can choose the database system based on their preference and familiarity.
6. Implement JWT-based authentication for accessing protected routes.
7. Include error handling middleware to gracefully handle errors.
8. Organize your project into separate directories for config, controllers, routes, models, services, middleware, and have a single-entry point (e.g., app.js) to initialize and configure your application.
9. To keep a clean and modular codebase, use a **`.env`** file to manage environment variables across different environments (e.g., development, staging, production). This allows for easy configuration and enhances security by keeping sensitive information, such as database credentials and API keys, out of your codebase.

Product Module Specifications:

1. **User Fields (For registration)**
    - Name
        - Required
    - Email
        - Required
        - Should be valid email
    - Password
        - Required
        - Minimum 8 Chars
        - Must have One Capital character, One small character & One Symbol

2. **Product Fields:**
    - Name
        - Required
        - Should be an Alphanumeric string
    - Price
        - Required
        - Should be Decimal
    - Description (Optional)
        - Optional
        - Should not be more than 255 Chars
    - Product Type
        - Required
        - Should be one of the following
            - Print Product
            - Promotional Product
    - Product Image
        - Optional
        - Allow images only

3. **Endpoints:**
    - **Registering a New User**
        - **Endpoint:** POST /api/auth/register
        - **Description:** Allows a new customer to register by providing their name, email, and password
        - **Authentication:** Not required.
    - **Logging in**
        - **Endpoint:** POST /api/auth/login
        - **Description:** Allows a registered customer to log in by providing their email and password. Upon successful authentication, a JWT token is returned.
        - **Authentication:** Not required.
    - **Retrieving a List of Products**
        - **Endpoint:** GET /api/products
            - Should have pagination
            - Search By Product Name & Description
        - **Description:** Retrieves a list of all Products.
        - **Authentication:** Required. Only authenticated users should be able to access this endpoint.
    - **Retrieving a Specific Product**

        - **Endpoint:** GET /api/products/:id
        - **Description:** Retrieves information about a specific Product based on their ID.
        - **Authentication:** Required. Only authenticated users should be able to access this endpoint.
    - **Creating a New Product**
        - **Endpoint:** POST /api/products
        - **Description:** Creates a new Product record.
        - **Authentication:** Required. Only authenticated users should be able to create new Product records.
    - **Updating an Existing Product**
        - **Endpoint:** PUT /api/ products/:id
        - **Description:** Updates information for an existing Product based on their ID.
        - **Authentication:** Required. Only authenticated users should be able to update Product records.
    - **Deleting a Product**
        - **Endpoint:** DELETE /api/product/:id
        - **Description:** Removes a Product record based on their ID.
        - **Authentication:** Required. Only authenticated users should be able to remove Product records.

4. **Product Image Upload:**
    - Allow user to upload an image during creating / updating a product.
    - Store the uploaded images securely.
    - Provide endpoints for fetching and displaying Product profile images.
5. **Password Hashing:**
    - Hash passwords using bcrypt before storing them in the database.
6. **Input Validation and Sanitization:**
    - **Express-Validator**: We will use the **`express-validator`** middleware to validate and sanitize incoming request data before it reaches our route handlers. This will ensure that the data meets the required criteria and is safe for further processing.
    - **Sequelize/Mongoose Model Validation**: In addition to API-level validation, we will utilize Sequelize/Mongoose's built-in fields validation features to enforce constraints on our data at the ORM level. This ensures that only valid data is stored in the database, providing an extra layer of validation, and keeping data integrity.
7. **JWT Authentication:**
    - Implement JWT-based authentication for protecting routes that require authorization.
8. **ORM with Sequelize (MySQL) or Mongoose (MongoDB):**

- Define Sequelize models for Product records if using MySQL. OR Define Mongoose schemas for Product records if using MongoDB

9. **Error Handling:**
    - Implement error handling middleware to catch and handle errors gracefully.
    - Return proper error responses with meaningful messages.
10. **Async/Await**: Utilize async/await syntax for handling asynchronous operations, such as database queries or external API requests, to improve code readability and maintainability.
11. **Unit Testing (Optional)**: Write unit tests using frameworks like Jest or Mocha along with assertion libraries like Chai to test individual functions and modules for correctness.

Instructions:

1. **Code Originality:** Write the code for this exercise from scratch without directly copying and pasting from external sources or earlier projects. Reusing code snippets for learning purposes is acceptable but strive to understand and adapt them to the exercise's requirements.
2. **GitHub Repository:** Create a public repository on GitHub to host your project code. Commit your code and push it to the repository to track changes.
3. **Working Model:** The exercise must result in a fully functional Product module that meets the specified requirements. Test the API endpoints thoroughly to ensure they perform as expected and handle various scenarios gracefully. The exercise will only be considered successfully completed if the implemented module functions correctly.
4. **Ethical Conduct:** Be sure to write your own code and follow the rules. Do not copy from others and respect their work.
5. **Documentation:** Document your code thoroughly, including comments and README files, to explain the purpose, functionality, and usage of each part. Provide clear instructions for running the application and testing the API endpoints.
6. **Deployment (Optional):** Optionally, deploy your application on a platform like Heroku or any other platform that supports Node.js deployment easily. Include instructions for deployment in your README file.

Conclusion:

By completing this exercise, participants will gain practical experience in building a secure and functional Product module with Node.js. They will develop ability in fundamental concepts such as CRUD operations, authentication, input validation, ORM usage, and error handling, which are essential for building robust web applications.