Technical Task

Overview:
You are tasked with building a backend API using **NestJS** with **MongoDB** as the database. The API should have an **Authentication module** and a **CRUD module for managing products**. There will be two types of users: **Admin** and **User**. Only Admins should be able to delete products. JWT-based authentication and authorization must be implemented using **Guards** and **Pipes**.

Task Requirements:

1. **Authentication Module**:
    - Implement user registration and login functionality.
    - Use **JWT** for authentication.
    - Ensure the following fields for user registration:
        - `email` (required, unique)
        - `password` (required, hashed before storing)
        - `role` (either `Admin` or `User`)
    - On login, generate and return a JWT token.
    - Protect the routes using JWT guards, ensuring only authenticated users can access certain endpoints.
2. **Authorization**:
    - Use **Guards** to protect specific routes. Only allow users with the **Admin** role to delete products.
    - The roles should be stored in the MongoDB user document as `role: 'Admin' | 'User'`.
3. **Product CRUD Module**:
    - Implement the following operations for the product:
        - **Create** a product
        - **Retrieve** (fetch all products or a single product by ID)
        - **Update** a product
        - **Delete** a product (restricted to Admin only)
    - The Product schema should have the following fields:
        - `name` (required)
        - `description`
        - `price` (required)
        - `category`
        - `createdBy` (references the user who created the product)
4. **Roles and Access Control**:

- **User**: Can create, retrieve, and update products.
- **Admin**: Can perform all CRUD operations, including product deletion.
- Apply access control using **Guards** for authorization checks.

5. **Pipes**:
    - Use **Validation Pipes** to ensure the data is valid before processing it (e.g., product creation/update).
6. **MongoDB**:
    - Use **Mongoose** for MongoDB schema creation and interaction.
    - Implement a MongoDB schema for **Users** and **Products**.
7. **Other Considerations**:
    - Include appropriate error handling (e.g., invalid data, unauthorized access).
    - Use **DTOs** (Data Transfer Objects) for product creation and user registration.

API Endpoints:

1. **Auth Routes**:
    - `POST /auth/register`: Register a new user.
    - `POST /auth/login`: Log in and receive a JWT token.
2. **Product Routes**:
    - `POST /products`: Create a new product (User or Admin).
    - `GET /products`: Retrieve all products (User or Admin).
    - `GET /products/:id`: Retrieve a single product by ID (User or Admin).
    - `PATCH /products/:id`: Update a product (User or Admin).
    - `DELETE /products/:id`: Delete a product (Admin only).

Technologies to Use:

- **NestJS** framework
- **MongoDB** with **Mongoose**
- **JWT** for authentication
- **Guards** and **Pipes** for authorization and validation

Expected Deliverables:

- A NestJS project with the above functionalities implemented.
- Clear and structured code with comments where necessary.
- Proper use of NestJS modules, services, controllers, and guards.
- A **GitHub repository** containing the project code and README with instructions to run the project.
- A **Postman collection** for testing the API should also be provided.

Note on GitHub and Postman:

- After completing the task, upload your project to **GitHub**. Include a detailed **README** file explaining how to set up and run the project locally.
- Attach a **Postman collection** that includes all the necessary API endpoints for testing. Make sure to provide sample requests for authentication, product creation, update, and deletion, along with the appropriate authorization headers for the Admin and User roles.

Evaluation Criteria:

- Code clarity and structure.
- Proper implementation of authentication and authorization.
- Efficient use of NestJS modules and MongoDB (Mongoose).
- Correct use of Guards, Pipes, and DTOs for data validation and security.