Node.js backend test assignment

Objective

Develop a secure and scalable user authentication and article management system with role-based access control. This system should be built using NestJS, TypeORM, PostgreSQL, Docker, and Swagger for API documentation. It should support multiple roles per user, allow users to switch roles, enable account deletion with cascading resource removal, and provide functionalities for article management. The application will be containerized for easy deployment.

Technical Requirements

- **Backend Framework**: NestJS.
- **Database**: PostgreSQL.
- **ORM**: TypeORM with migrations enabled.
- **Authentication**: JWT access and refresh tokens for handling authentication.
- **Containerization**: Docker for the application packaging. Docker-Compose for orchestrating the NestJS app and the PostgreSQL database service.
- **API Documentation**: Swagger to document endpoints.

Core Features

1. **User Registration and Authentication**:
   - **Registration**: Allow new users to sign up with an email and password. Upon registration, assign them a default role of Viewer or let them choose from a set of allowed roles, excluding Admin.
   - **Login**: Develop a system that authenticates users with their email and password, issuing JWT access and refresh tokens and storing them in cookies. For users with multiple roles, add a step for role selection.
2. **Role Management:**
   - Define three roles with distinct access levels: Admin, Editor, and Viewer.
   - Allow users to switch between their assigned roles after logging in, ensuring this feature is accessible only to authenticated users.
   - Enable users to delete their accounts, implementing cascading deletion of all associated resources in a transactional manner.
3. **Roles privileges**:
   - Admin
     - Manage users (read, delete).
     - Manage articles (read, delete), regardless of who created them.
   - Editor
     - Create and read articles, update and delete their articles.
   - Viewer
     - Read-only access to articles.
4. **Pagination:**
   - Ensure that at least one GET endpoint returning collections of entities supports pagination
5. **Database Pre-Seeding**:
   - Ensure the database is pre-seeded with roles upon application startup to streamline the setup process.
   - Ensure the database is pre-seeded with the default admin user upon application startup.

6. **Documentation with Swagger**:
    - Document all API endpoints thoroughly using Swagger. This includes detailed descriptions, request bodies, parameters, response schemas, and examples of potential errors, facilitating easy understanding and integration for developers and users.

Deliverables

- Complete source code of the NestJS application.
- Dockerfile for the Node.js application and a Docker-Compose file for orchestrating the NestJS app and PostgreSQL service.
- A README file with setup instructions, including how to run the application using Docker and Docker-Compose.
- Accessible Swagger documentation for all API endpoints.

Evaluation Criteria

- **Functionality**: The application must fulfill at least registration, authentication, and role changes.
- **Code Quality**: The source code should be clean and well-organized.
- **Documentation**: Swagger and README documentation should be comprehensive, clear, and helpful.

This comprehensive task is designed to assess a developer's proficiency in building secure, scalable server-side applications with modern technologies and practices, focusing on authentication, role-based access control, database management, containerization, and API documentation.