Farm Application

Overview

The Farm Application is a Spring Boot-based web application designed to manage various aspects of a farm, including products, orders, carts, and user accounts. The application utilizes a RESTful architecture, allowing for easy integration with frontend applications and other services.

Architecture

The application follows a layered architecture, which includes the following layers:

1. Controller Layer: Handles incoming HTTP requests and returns responses. It acts as an interface between the client and the service layer.
2. Service Layer: Contains the business logic of the application. It interacts with the repository layer to perform CRUD operations and other business-related tasks.
3. Repository Layer: Responsible for data access and manipulation. It interacts with the database using Spring Data JPA.
4. Model Layer: Contains the entity classes that represent the data structure of the application.
5. DTO Layer: Data Transfer Objects (DTOs) are used to transfer data between layers, especially between the controller and service layers.

Key Components

1. Models

The application defines several entity classes that represent the core data structures:

- **User **: Represents a user of the application, including fields for email, password, role, and confirmation status.
- Product: Represents a product available in the farm, including fields for name, description, price, quantity, and image.
- Cart: Represents a user's shopping cart, which contains a list of cart items.
- CartItem: Represents an item in the cart, linking a product to a cart and including the quantity.
- Order: Represents an order placed by a user, including fields for address, phone number, status, and a list of order items.
- OrderItem: Represents an item in an order, linking a product to an order and including the quantity and price.
- Comment: Represents a comment made by a user on a product, including content and score.

2. Repositories

The application uses Spring Data JPA repositories to interact with the database. Each repository interface extends `JpaRepository`, providing built-in methods for CRUD operations. Custom queries can be defined using the `@Query` annotation.

- User Repository: Handles user-related database operations.
- ProductRepository: Handles product-related database operations, including a custom query to fetch products without comments.
- CartRepository: Handles cart-related database operations.
- OrderRepository: Handles order-related database operations.
- CommentRepository: Handles comment-related database operations.

3. Services

The service layer contains classes that implement the business logic of the application. Each service class is annotated with `@Service` and uses dependency injection to access repositories and other services.

- User Service: Manages user registration, authentication, and email confirmation.
- ProductService: Manages product creation, updating, deletion, and retrieval.
- CartService: Manages cart operations, including adding items, retrieving the cart, and clearing the cart.
- OrderService: Manages order creation and retrieval, including sending order confirmation emails.
- CommentService: Manages comments on products.

4. Controllers

The controller layer contains RESTful endpoints that handle HTTP requests. Each controller is annotated with `@RestController` and maps requests to specific service methods.

- AuthController: Handles user authentication and registration.
- CartController: Manages cart-related operations.
- ProductController: Manages product-related operations.
- OrderController: Manages order-related operations.
- CommentController: Manages comments on products.

5. Security Configuration

The application uses Spring Security to secure endpoints and manage user authentication. The security configuration includes:

- JWT Authentication: The application uses JSON Web Tokens (JWT) for stateless authentication. The `JwtService` class handles token generation and validation.
- SecurityFilterChain: Configures security settings, including CORS, CSRF protection, and authorization rules for different endpoints.
- User DetailsService: Loads user-specific data for authentication.

6. CORS Configuration

The application includes a CORS configuration class that allows cross-origin requests from specified origins (e.g., `http://localhost:3000`). This is essential for frontend applications that need to communicate with the backend.

7. Error Handling

The application includes a global exception handler using `@ControllerAdvice` to handle exceptions and return meaningful error responses. Custom exceptions like `ResourceNotFoundException` and `InsufficientStockException` are defined to handle specific error scenarios.

8. Email Service

The application includes an `EmailService` class that uses Spring's `JavaMailSender` to send emails for user registration confirmation and order confirmations.

9. Database Configuration

The application uses MySQL as the database. The database connection properties are defined in the `application.properties` file, including the database URL, username, and password.

10. Testing

Unit tests are implemented using JUnit and can be run using Maven. The tests cover various aspects of the application, including service methods and controller endpoints.

Conclusion

The Farm Application is a comprehensive solution for managing farm-related operations. It leverages modern technologies and best practices to provide a robust and scalable application. The modular architecture allows for easy maintenance and future enhancements.