





**BÁO CÁO TỔNG HỢP**

**Thực hiện: Nguyễn Đức Bảo**





**Hà Nội -2025**



1. Thiết kế hệ thống

1.1. Kiến trúc tổng thể

Gồm 3 lớp cơ bản : Presentation Layer, Service Layer, Data Access Layer

* Presentation Layer: Chứa các REST Controller chịu trách nhiệm về việc tiếp nhận yêu cầu, gọi đến lớp Service để xử lý nghiệp vụ, và định dạng dữ liệu trả về.
* Service Layer: Chứa các Service, nhận yêu cầu từ Presentation Layer, thực hiện các thao tác phức tạp, tương tác với Data Access Layer.
* Data Access Layer: Chứa các Repository, chịu trách nhiệm tương tác trực tiếp với cơ sở dữ liệu, thực hiện các thao tác CRUD và ánh xạ đối tượng Java sang các trường trong cơ sở dữ liệu.

1.2. Các thành phần chính

* Ứng dụng Spring Boot
* Chức năng:
* Cung cấp các API RESTful để quản lý người dùng
* Xử lý logic nghiệp vụ liên quan đến người dùng.
* Tương tác với cơ sở dữ liệu MySQL.
* Cấu hình:
* Sử dụng application.properties để cấu hình kết nối cơ sở dữ liệu
* Cơ sở dữ liệu MySQL
* Sử dụng MySQL 8.0
* Chức năng: Lưu trữ dữ liệu người dùng và các thực thể khác
* Docker Compose
* Chức năng: Điều phối các container Docker, cho phép định nghĩa và chạy ứng dụng đa container một cách dễ dàng.
* Cấu hình: Định nghĩa hai dịch vụ: MySQL và Spring Boot.
* Thiết lập mạng nội bộ giữa các container, cho phép app kết nối dến db bằng tên dịch vụ.

2. Luồng API

2.1. Một số API Endpoint chính

1. GET /api/admin/users

* Mô tả: Lấy danh sách tất cả người dùng trong hệ thống
* Yêu cầu xác thực/phân quyền: Yêu cầu vai trò ADMIN
* Luồng hoạt động :
* Client gửi yêu cầu GET đến /api/admin/users.
* Spring Security kiểm tra xem người dùng có vai trò ADMIN hay không. Nếu không, yêu cầu bị từ chối
* Nếu có quyền UserController gọi UserRepository.findAll()để lấy tất cả các đối tượng User từ cơ sở dữ liệu.
* Trả về danh sách người dùng

2. DELETE /api/admin/users/{id}

* Mô tả: Xóa một người dùng dựa trên ID.
* Yêu cầu xác thực/phân quyền: Yêu cầu vai trò ADMIN
* Luồng hoạt động:
* Client gửi yêu cầu DELETE đến /api/admin/users/{id}
* Spring Security kiểm tra xem người dùng có vai trò ADMIN hay không. Nếu không, yêu cầu bị từ chối
* Nếu có quyền, kiểm tra sự tồn tại của người dùng
* Nếu người dùng không tồn tại, trả về trạng thái HTTP 404 Not Found.
* Nếu người dùng tồn tại, id được gọi sẽ bị xóa
* Trả về trạng thái HTTP 204 No Content

3. GET /api/tasks

* Mô tả: xem tất cả các task của người dùng đó.
* Yêu cầu xác thực/phân quyền: Yêu cầu vai trò USER
* Luồng hoạt động:
* Client gửi yêu cầu GET đến /api/tasks
* Spring Security kiểm tra xem người dùng có vai trò USER hay không
* Nếu có quyền USER, hiện các task của người dùng đó
* Nếu không có quyền , trả về trạng thái HTTP 404 Not Found.

4. GET /api/admin/tasks

* Mô tả: xem tất cả các task của tất cả người dùng .
* Yêu cầu xác thực/phân quyền: Yêu cầu vai trò ADMIN
* Luồng hoạt động:
* Client gửi yêu cầu GET đến /api/admin/tasks
* Spring Security kiểm tra xem người dùng có vai trò ADMIN hay không
* Nếu có quyền ADMIN, hiện tất cả các task trong cơ sở dữ liệu.
* Nếu không có quyền , trả về trạng thái HTTP 404 Not Found.

2.2. Một số API Endpoint khác

* createUser(User user): Tạo một người dùng mới.
* getUserById(Long id): Lấy thông tin một người dùng theo ID.
* updateUser(Long id, User userDetails): Cập nhật thông tin người dùng.

2.3. Luồng dữ liệu và tương tác giữa các lớp

* Request từ Client: Yêu cầu HTTP đến Controller
* Xác thực/Phân quyền: Spring Security chặn và kiểm tra quyền trước khi yêu cầu đến phương thức của controller.
* Controller: Xử lý yêu cầu HTTP, gọi các phương thức nghiệp vụ trong service.
* Service :Chứa logic nghiệp vụ.
* Repository :Tương tác với cơ sở dữ liệu MySQL thông qua Spring Data JPA và Hibernate.
* Database :Thực hiện các thao tác SQL (CREATE TABLE, SELECT, DELETE) và trả về dữ liệu.
* Response về Client: Dữ liệu được trả về từ Repository, đi qua Service, sau đó Controller định dạng lại và gửi phản hồi HTTP về cho client.

3. Mô hình dữ liệu

3.1. Các Entity chính

1. User

* Được sử dụng để lưu trữ thông tin người dùng.
* Các trường:
* Id : int - ID của người dùng
* Email : varchar(255)- Địa chỉ email của người dùng
* Password : varchar(255)- Mật khẩu của người dùng
* Username : varchar(255)- Tên đăng nhập
* Mối quan hệ:

1. Mối quan hệ với Role
2. Mối quan hệ với Task trong bảng Task là khóa ngoại

2. Task

* Được sử dụng để lưu trữ thông tin về các tác vụ.
* Các trường:
* Id : int - ID của tác vụ
* Create at : datetime - Thời gian tạo tác vụ
* Description : varchar(255)- Mô tả chi tiết tác vụ.
* due\_date: date- Ngày đến hạn của tác vụ
* status: enum ('COMPLETED','PENDING') - Trạng thái của tác vụ
* title: varchar(255)- Tiêu đề của tác vụ.
* assigned\_to\_user\_id: int - ID của người dùng được giao tác vụ này
* Mối quan hệ:

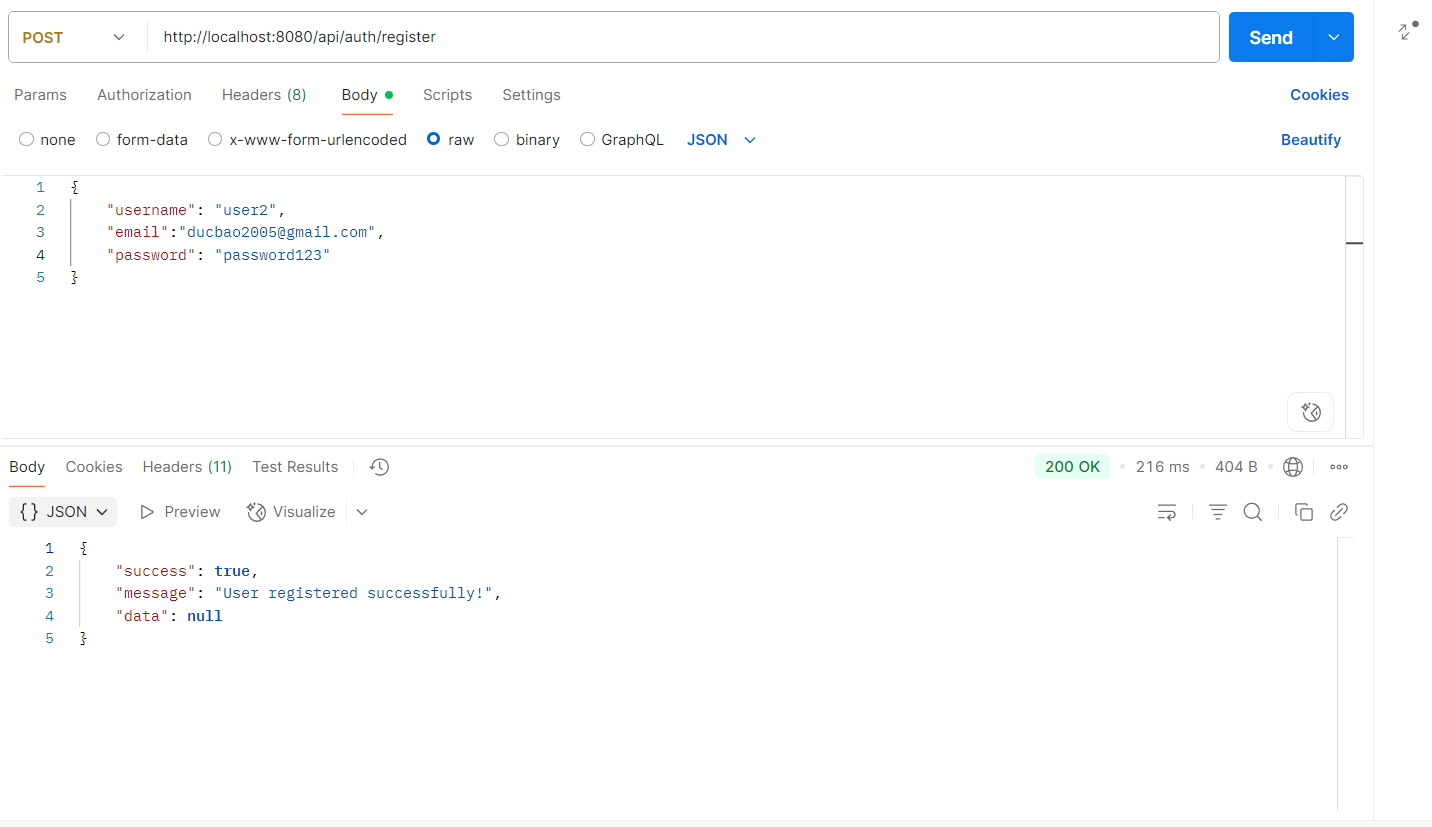
1. Mối quan hệ với User: Một người dùng có thể có nhiều vai trò, và một vai trò có thể được gán cho nhiều người dùng.

4. Demo chức năng chính

Tạo yêu cầu GET, POST, PUT, DELETE

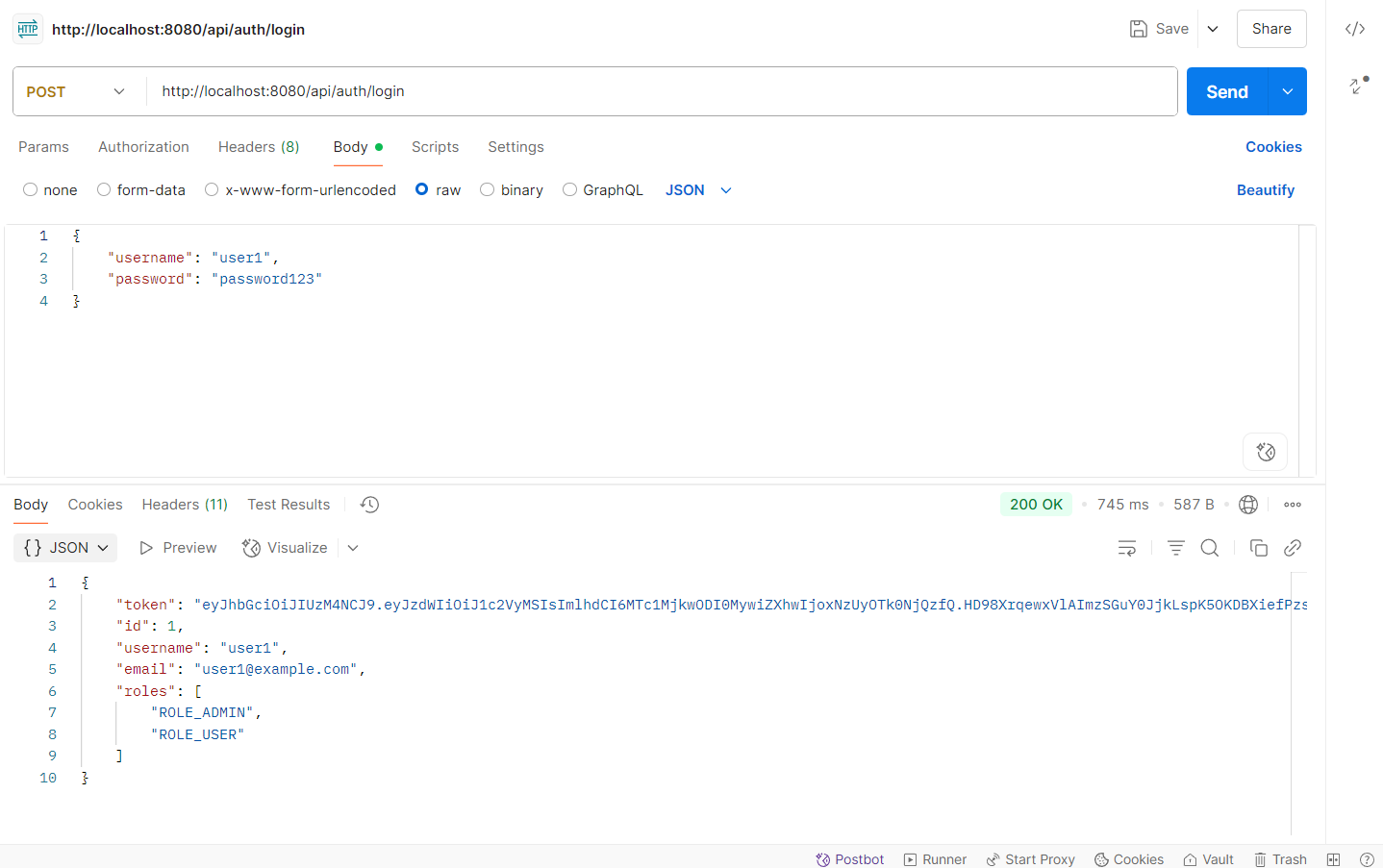
4.1 POST /api/auth/register

Đăng ký tài khoản mới



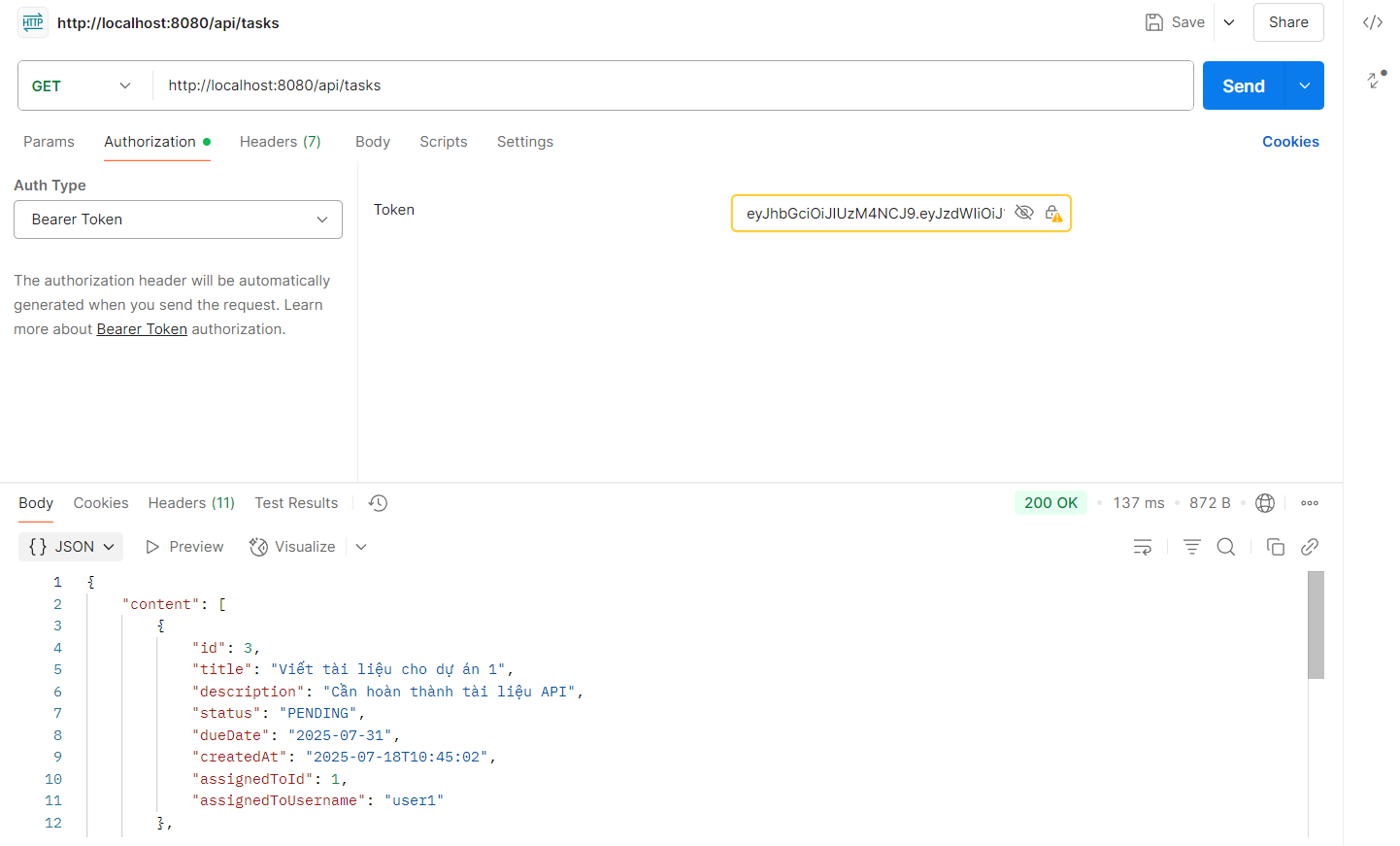
4.2 POST /api/auth/login

Đăng nhập thành công tài khoản User



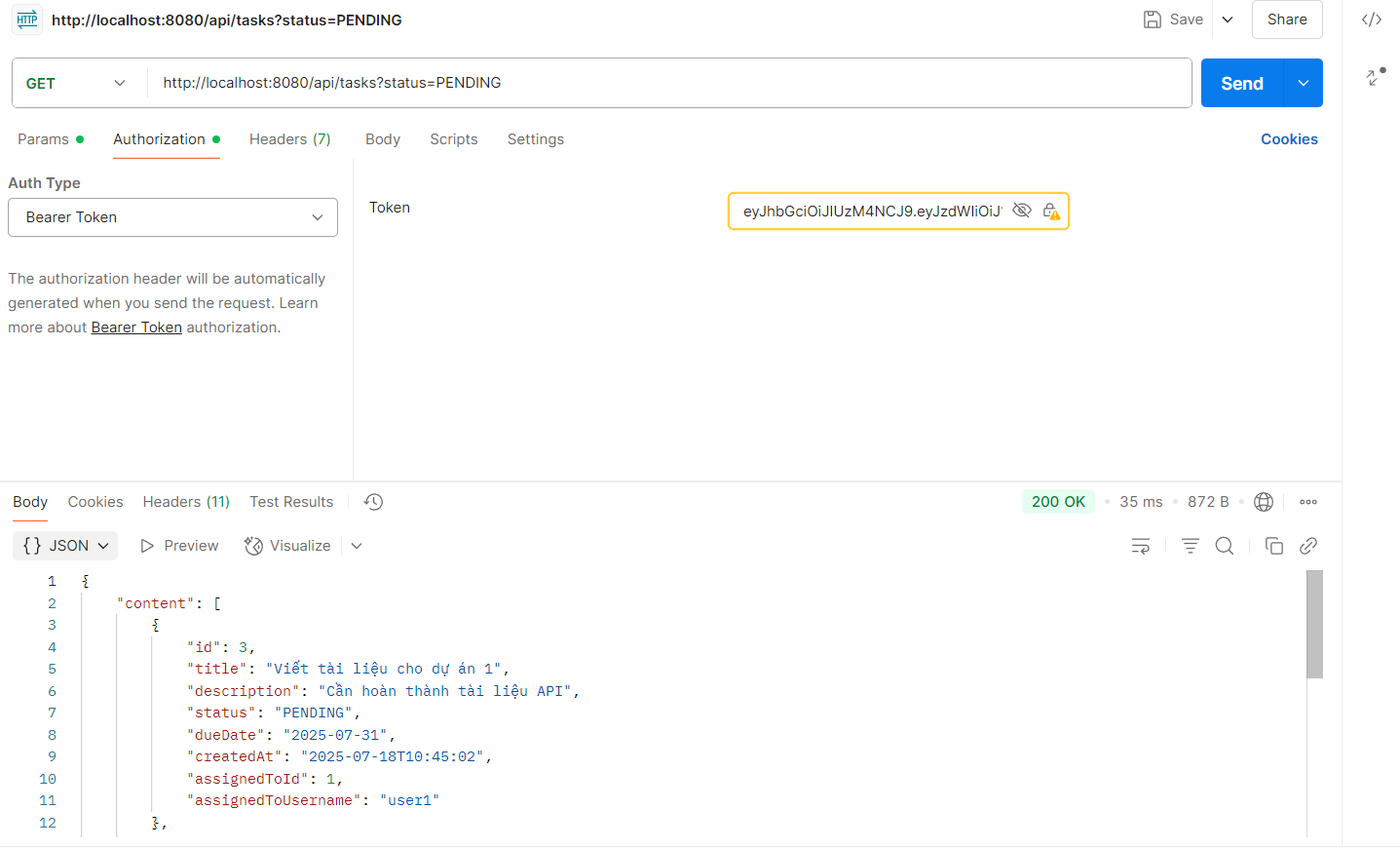
4.3 GET /api/tasks

Hiển thị dữ liệu tasks của user đó.



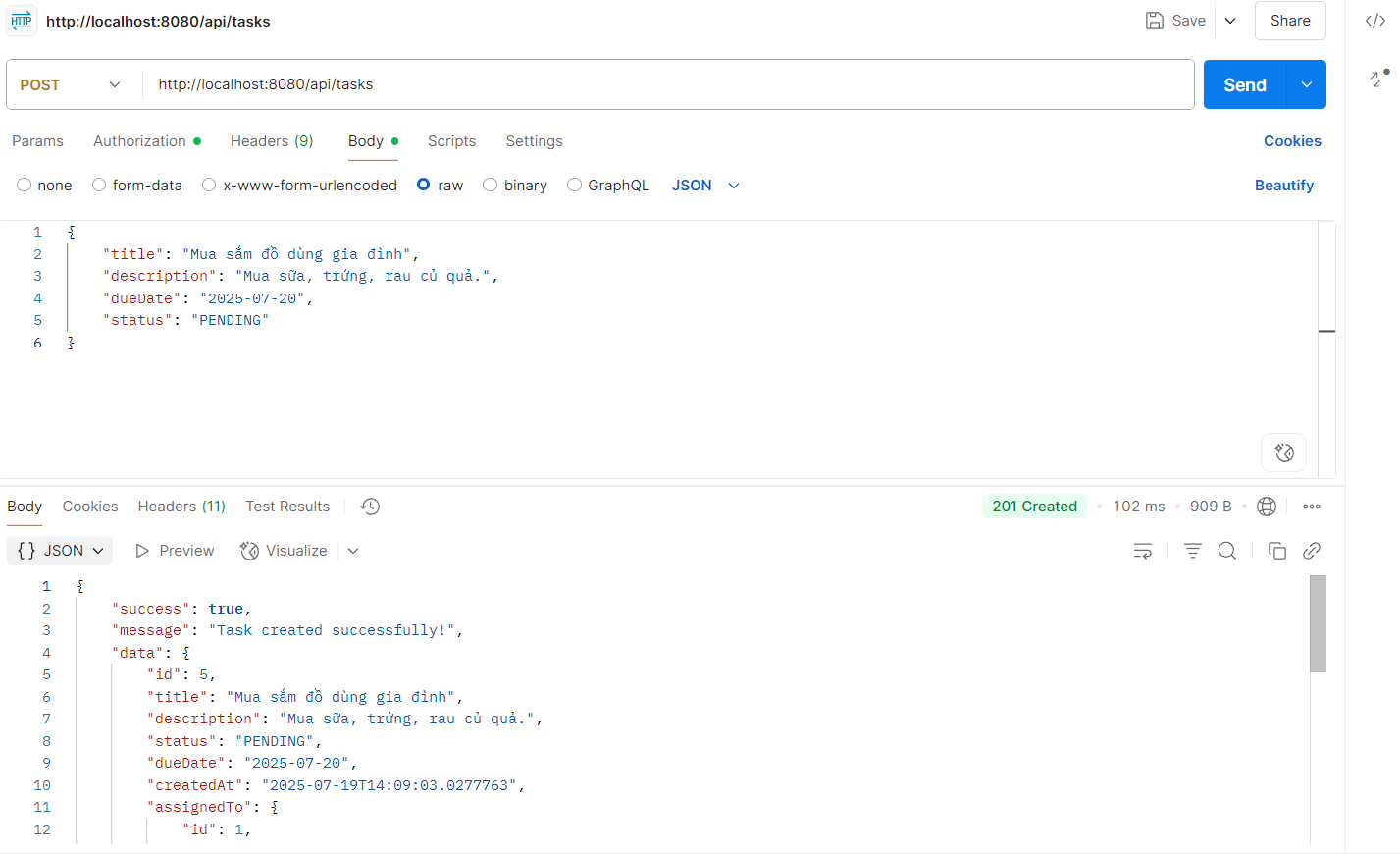
4.4 GET /api/tasks?status=PENDING

Hiển thị dữ liệu task đã lọc filter của user đó



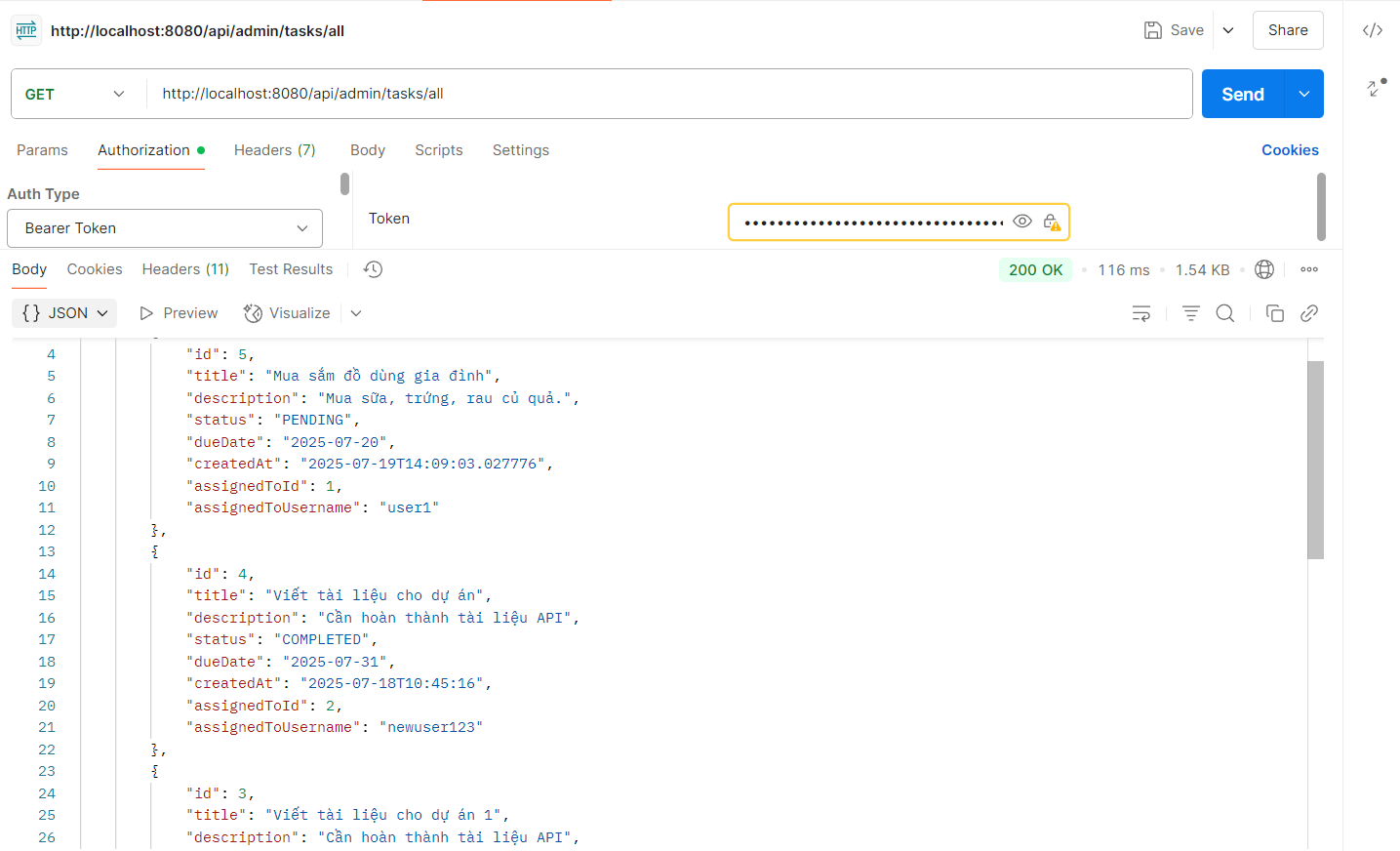
4.5 POST /api/tasks

Thêm dữ liệu tasks của user đó



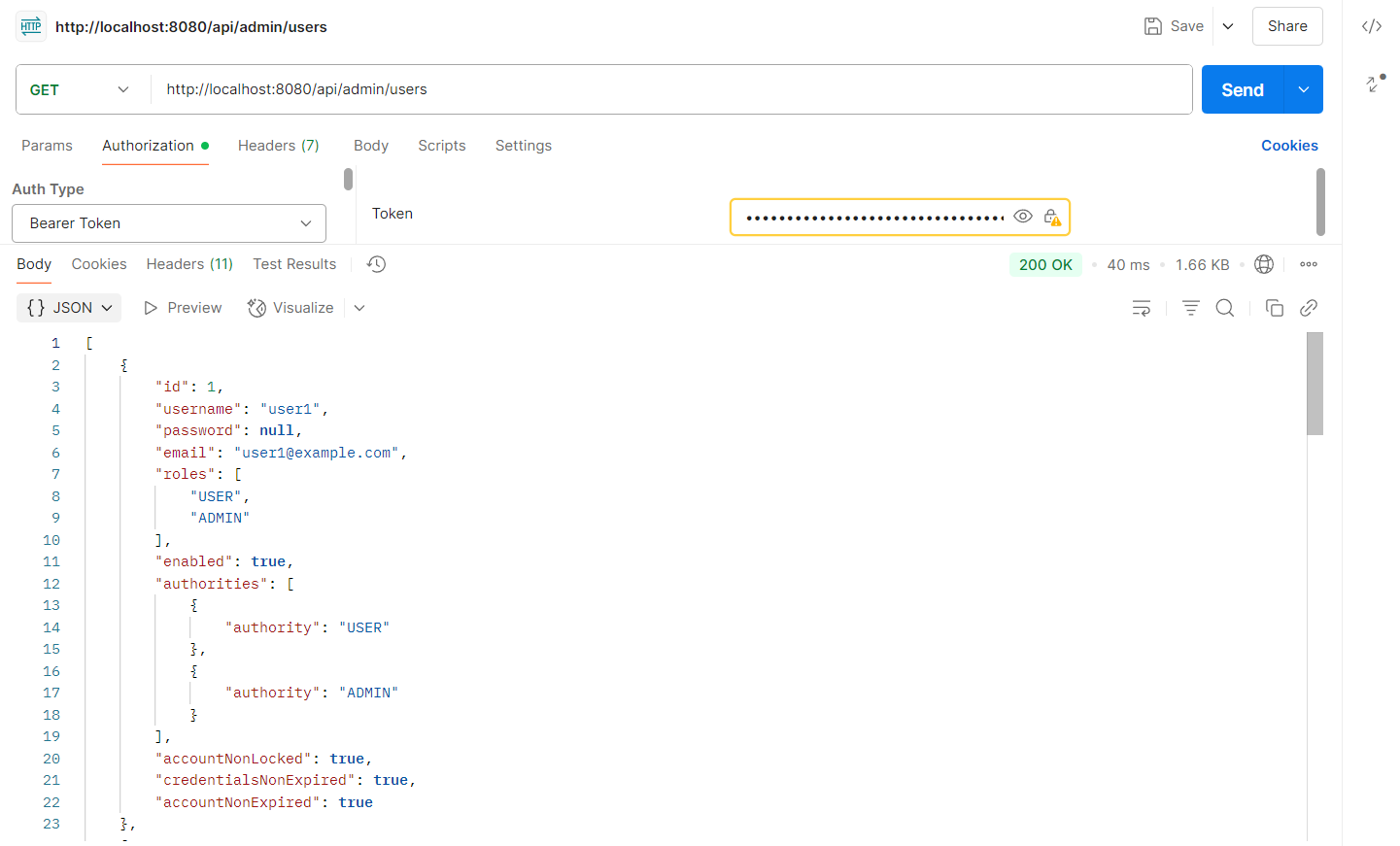
4.6 GET /api/admin/tasks/all

Đăng nhập vào tài khoản ADMIN và hiển thị tất cả các task của tất cả người dùng



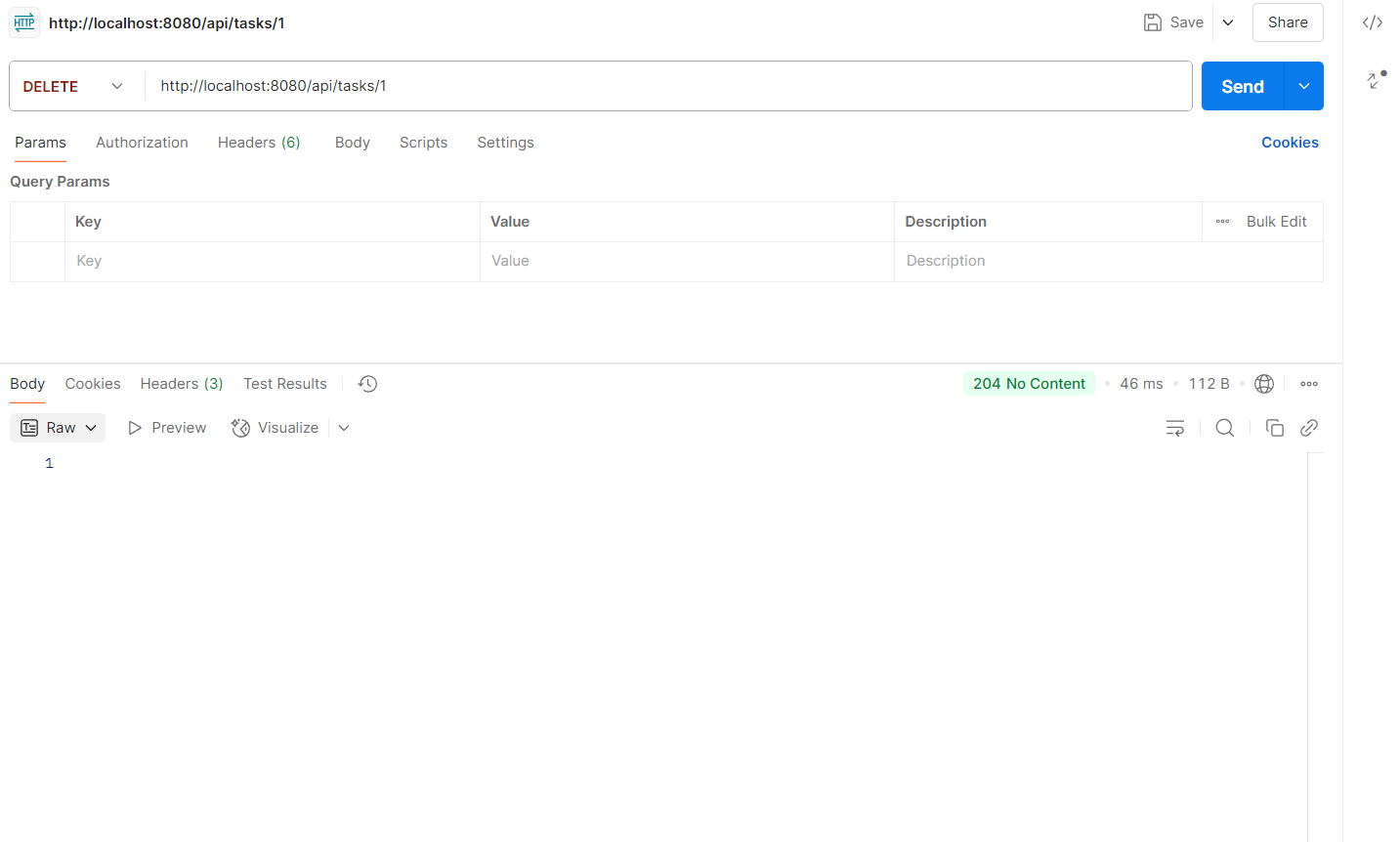
4.7 GET /api/admin/users

Dùng vai trò ADMIN để hiển thị tất cả các tài khoản user



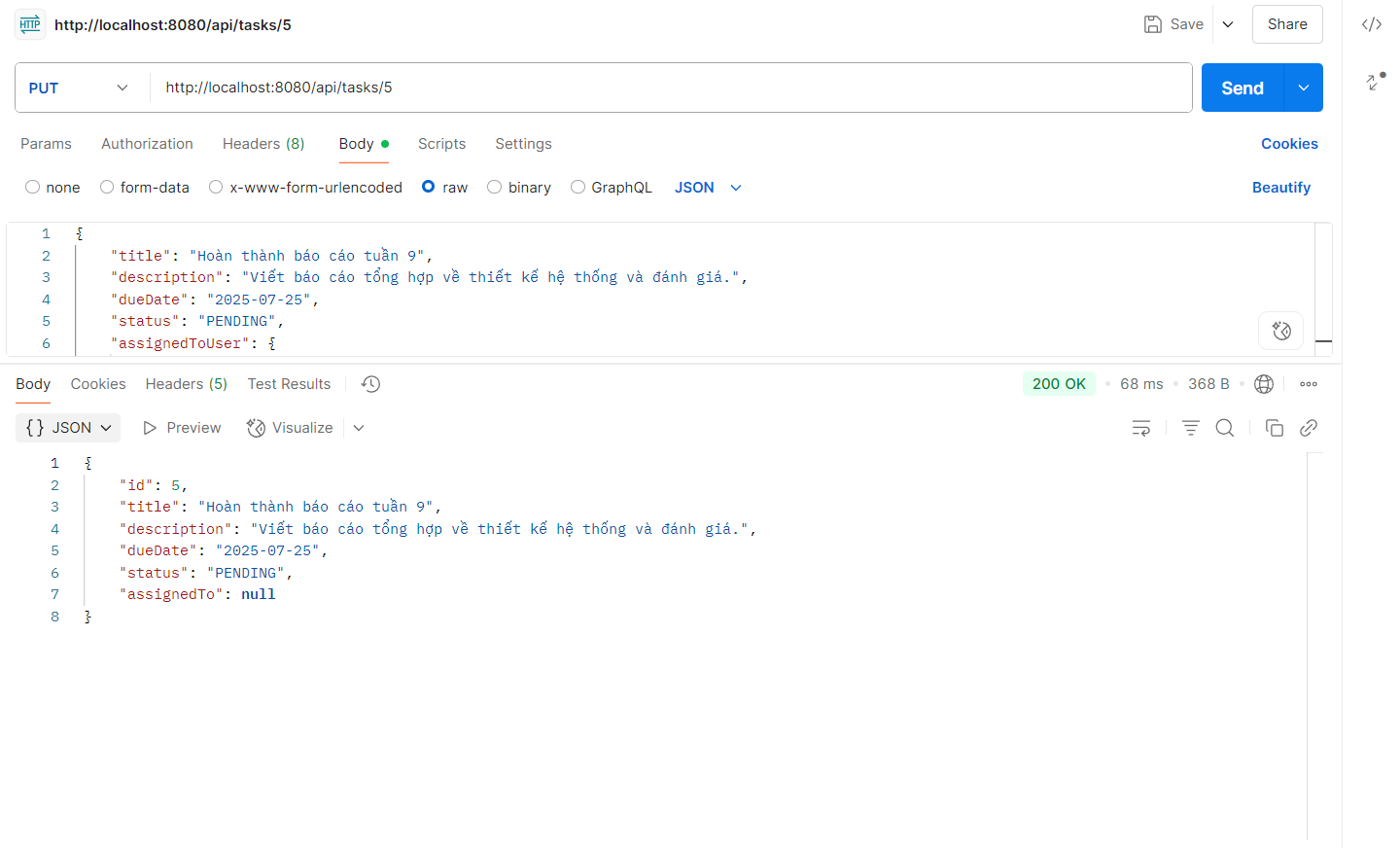
4.8 DELETE /api/tasks/{id}

Xóa task bằng id đã chọn



4.9 PUT api/tasks/{id}

Sửa task bằng id đã chọn



5. Đánh giá

5.1 Clean Code

* Ưu điểm :
* Tuân thủ quy ước đặt tên: tên lớp , phương thức.
* Tách biệt trách nhiệm: Phân tách rõ ràng giữa controller, service, repository.
* Sử dụng Dependency Injection: Dùng @Autowired để tiêm các dêpndency
* Nhược điểm
* Chưa xử lý ngoại lệ chi tiết .
* Validate chưa chặt chẽ đầu vào

5.2 Bảo mật

* Ưu điểm :
* Xác thực/Phân quyền cơ bản.
* Không trả về mật khẩu: mật khẩu khi được hiện lên danh sách sẽ là null
* Mật khẩu được mã hóa Bcrypt.
* Nhược điểm
* Chưa xử lý thành thạo JWT Token
* Còn nhiều lỗi nhỏ về bảo mật.

5.3 Hiệu năng

* Ưu điểm :
* Spring Data JPA: Giảm thiểu boilerplate code, giúp phát triển nhanh chóng
* wait-for-it.sh: Giúp đảm bảo các dịch vụ khởi động theo đúng thứ tự và MySQL sẵn sàng
* Nhược điểm
* Thiếu cơ chế caching
* Dựa hoàn toàn vào thông tin được cung cấp