**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

---

# BÁO CÁO

# THỰC TẬP CƠ SỞ

**Giảng viên: Kim Ngọc Bách**

**Tuần 7: 26/4/2025**

**Sinh viên: Nguyễn Khắc Trường B22DCCN884**

# MỤC LỤC

# A. Kiến thức học được ở tuần này

## Giới thiệu

Tiếp tục với kiến thức đã học được ở tuần trước, tuần này em tiếp tục tìm hiểu về Springboot

**Phần 1: Service trong Spring Boot**

Trong Spring Boot, Service là các lớp trung gian chịu trách nhiệm xử lý logic nghiệp vụ của ứng dụng. Chúng nằm giữa tầng Controller (giao tiếp với client) và Repository (truy xuất dữ liệu từ cơ sở dữ liệu). Các lớp Service được chú thích bằng annotation @Service, cho phép Spring quản lý vòng đời và tự động tiêm phụ thuộc (dependency injection) vào các thành phần khác trong hệ thống.

Service đóng vai trò quan trọng trong việc tổ chức và phân tách các quy tắc nghiệp vụ khỏi tầng trình bày (UI) và tầng truy xuất dữ liệu. Thay vì để Controller thực hiện toàn bộ xử lý, Service giúp hiện thực hóa nguyên tắc **tách biệt trách nhiệm** (Separation of Concerns) bằng cách tập trung vào các nghiệp vụ cốt lõi như kiểm tra tính hợp lệ của dữ liệu, thực hiện các phép tính phức tạp, hoặc xử lý quy trình nhiều bước. Điều này không chỉ giúp mã nguồn dễ bảo trì, dễ đọc mà còn hỗ trợ việc viết unit test độc lập cho từng phần logic.

- Trong thực tế, một lớp Service thường gọi đến một hoặc nhiều lớp Repository để truy vấn hoặc ghi dữ liệu vào cơ sở dữ liệu. Đồng thời, chúng có thể kết hợp với annotation @Transactional nhằm đảm bảo tính toàn vẹn dữ liệu, nhất là trong các thao tác thay đổi dữ liệu cần nhiều bước xử lý liên tiếp. Việc sử dụng transaction giúp hệ thống tránh được các tình huống dữ liệu bị sai lệch do lỗi giữa chừng, đặc biệt quan trọng trong các nghiệp vụ tài chính hoặc xử lý đơn hàng.

- Các Service còn có thể gọi đến những thành phần ngoài hệ thống như dịch vụ gửi email, thanh toán, API bên thứ ba hoặc các hệ thống microservice khác thông qua RESTful hoặc message queue. Do đó, Service chính là nơi thể hiện rõ nét nhất luồng nghiệp vụ, phản ánh các quy tắc hoạt động thực tế của tổ chức hoặc doanh nghiệp.
- Về mặt tổ chức mã nguồn, các lớp Service thường được đặt trong thư mục services hoặc service, tên lớp thường theo cú pháp [TênEntity]Service, ví dụ: UserService, OrderService, ProductService, v.v. Trong những ứng dụng có quy mô lớn, các Service có thể được chia thành nhiều lớp nhỏ hơn để phân tách chức năng, chẳng hạn như AuthService, NotificationService, hoặc FileStorageService.
- Việc xây dựng và duy trì hệ thống Service rõ ràng, mạch lạc giúp tăng khả năng mở rộng ứng dụng, đồng thời đảm bảo hệ thống có thể dễ dàng thích ứng với các thay đổi nghiệp vụ trong tương lai. Trong môi trường phát triển theo mô hình Agile hoặc microservice, các Service còn có thể được đóng gói lại thành những module riêng biệt, hỗ trợ tái sử dụng hoặc triển khai độc lập theo từng nhóm chức năng.

**Phần 2: Repository trong Spring Boot**

Trong Spring Boot, Repository là thành phần chịu trách nhiệm giao tiếp trực tiếp với cơ sở dữ liệu, đóng vai trò truy xuất, lưu trữ, cập nhật và xóa dữ liệu từ các bảng. Spring Data JPA cung cấp một cơ chế trừu tượng hóa rất mạnh mẽ thông qua các interface như CrudRepository, JpaRepository, hoặc PagingAndSortingRepository, giúp lập trình viên thao tác với dữ liệu mà không cần viết câu lệnh SQL thủ công.

- Một Repository thường được khai báo dưới dạng interface và được Spring tự động hiện thực hóa dựa trên tên phương thức và cú pháp quy ước. Ví dụ, phương thức findByUsername(String username) sẽ tự động ánh xạ thành câu lệnh SQL tương ứng để tìm kiếm bản ghi theo trường username. Điều này giúp giảm thiểu lượng mã lặp, tăng hiệu suất phát triển và tránh lỗi cú pháp SQL thông thường.

- Các lớp Repository thường được đặt trong thư mục repository hoặc repositories, đặt tên theo mẫu [Entity]Repository, chẳng hạn như UserRepository, ProductRepository, OrderRepository, v.v. Ngoài các phương thức mặc định như save, findById, findAll, deleteById, lập trình viên có thể định nghĩa thêm các query tùy chỉnh bằng cách sử dụng annotation @Query hoặc viết theo dạng method name convention.
- Trong những trường hợp phức tạp hơn, có thể sử dụng Native Query hoặc JPQL để truy vấn động, hoặc kết hợp với Specification, Criteria API để xử lý các truy vấn nâng cao và có điều kiện linh hoạt. Ngoài ra, Spring Data JPA còn hỗ trợ phân trang và sắp xếp rất hiệu quả thông qua các tham số Pageable và Sort.
- Repository không chứa logic nghiệp vụ mà chỉ đảm nhận vai trò cung cấp dữ liệu đầu vào cho Service hoặc nhận dữ liệu từ Service để lưu trữ. Việc duy trì ranh giới rõ ràng giữa Repository và Service là rất quan trọng trong việc tổ chức kiến trúc ứng dụng rõ ràng, dễ bảo trì và mở rộng.
- Sử dụng Repository theo cách nhất quán giúp tăng độ ổn định và chuẩn hóa việc truy cập cơ sở dữ liệu trong toàn hệ thống. Đặc biệt, trong các hệ thống có quy mô lớn, việc sử dụng Spring Data JPA còn giúp giảm thiểu thời gian phát triển backend và đảm bảo tính tương thích cao với nhiều loại cơ sở dữ liệu quan hệ khác nhau như MySQL, PostgreSQL, Oracle, SQL Server, v.v.

**Phần 3: Controller trong Spring Boot**

Controller là tầng giao tiếp giữa người dùng (hoặc các hệ thống khác) với ứng dụng Spring Boot. Nó tiếp nhận các HTTP request, xử lý dữ liệu đầu vào, gọi đến Service để thực hiện logic nghiệp vụ, sau đó trả về HTTP response. Các lớp Controller thường được chú thích bằng @RestController hoặc @Controller, kết hợp với các annotation như @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping để định nghĩa các endpoint cụ thể.

- Trong kiến trúc MVC (Model-View-Controller), Controller chính là "cửa ngõ" của hệ thống. Nó chịu trách nhiệm kiểm tra, chuyển đổi và xác thực dữ liệu từ client trước khi chuyển tiếp đến Service xử lý. Việc phân tách rõ ràng vai trò

của Controller giúp hệ thống có luồng xử lý rõ ràng, dễ dàng bảo trì và kiểm soát dữ liệu ra/vào hệ thống.

- Các Controller thường được tổ chức trong thư mục controller và đặt tên theo thực thể hoặc chức năng, ví dụ như UserController, AuthController, ProductController, v.v. Trong mỗi Controller, lập trình viên có thể khai báo các route tương ứng với các hành động CRUD (Create, Read, Update, Delete) hoặc các chức năng đặc thù khác như xác thực đăng nhập, upload file, gửi email, v.v.
- Để tăng tính bảo mật và kiểm soát request tốt hơn, Controller có thể kết hợp với các annotation như @Valid để kiểm tra tính hợp lệ của dữ liệu đầu vào, @RequestBody, @RequestParam, @PathVariable để ánh xạ các phần tử trong request, và @ResponseStatus để tùy chỉnh mã phản hồi HTTP. Ngoài ra, Spring còn hỗ trợ cơ chế xử lý lỗi toàn cục bằng @ControllerAdvice giúp nâng cao khả năng kiểm soát luồng lỗi và phản hồi thân thiện hơn đến client.
- Trong môi trường phát triển RESTful API, Controller đóng vai trò quyết định đến giao diện lập trình (API) của hệ thống, bao gồm cấu trúc đường dẫn, định dạng dữ liệu JSON/XML trả về và mã trạng thái HTTP. Một Controller được thiết kế rõ ràng, logic và nhất quán sẽ góp phần tạo nên một API dễ hiểu, dễ sử dụng và dễ tích hợp cho cả frontend lẫn các hệ thống bên ngoài.
- Việc xây dựng Controller có tổ chức, kết hợp chặt chẽ với Service và xử lý tốt các tình huống lỗi sẽ đảm bảo hệ thống Spring Boot hoạt động ổn định, thân thiện với người dùng và dễ mở rộng trong tương lai. Đặc biệt, khi kết hợp với Swagger hoặc OpenAPI, Controller còn có thể tự động sinh tài liệu API giúp việc phát triển và kiểm thử thuận tiện hơn.

## Phần 4: Tổng kết – Luồng hoạt động của Entity, Repository, Service và Controller trong Spring Boot

Trong một ứng dụng Spring Boot tiêu chuẩn, các thành phần Entity, Repository, Service, và Controller cùng nhau hợp thành một kiến trúc nhiều tầng (multi-layered architecture), đảm bảo việc xử lý dữ liệu, thực thi nghiệp vụ và giao tiếp với người dùng diễn ra hiệu quả, rõ ràng và dễ bảo trì.

Controller là điểm khởi đầu trong chuỗi xử lý. Khi người dùng gửi yêu cầu HTTP (qua trình duyệt hoặc API client), Controller sẽ tiếp nhận request, thực hiện

kiểm tra dữ liệu đầu vào (validation), sau đó chuyển giao nhiệm vụ xử lý cho tầng Service. Nó không trực tiếp truy xuất cơ sở dữ liệu hay thực hiện các logic nghiệp vụ phức tạp để đảm bảo nguyên tắc tách biệt trách nhiệm.

Service tiếp nhận yêu cầu từ Controller, thực hiện các quy tắc nghiệp vụ, kiểm tra tính hợp lệ, thực hiện tính toán hoặc gọi đến các dịch vụ khác nếu cần. Nếu nghiệp vụ cần thao tác dữ liệu, Service sẽ gọi đến tầng Repository để lấy hoặc ghi dữ liệu. Service chính là nơi phản ánh các quy tắc thực tế của hệ thống như "người dùng không được mua hàng khi chưa đăng nhập", "chỉ admin mới được xóa sản phẩm", hoặc "nếu số lượng hàng trong kho nhỏ hơn đơn hàng thì báo lỗi".

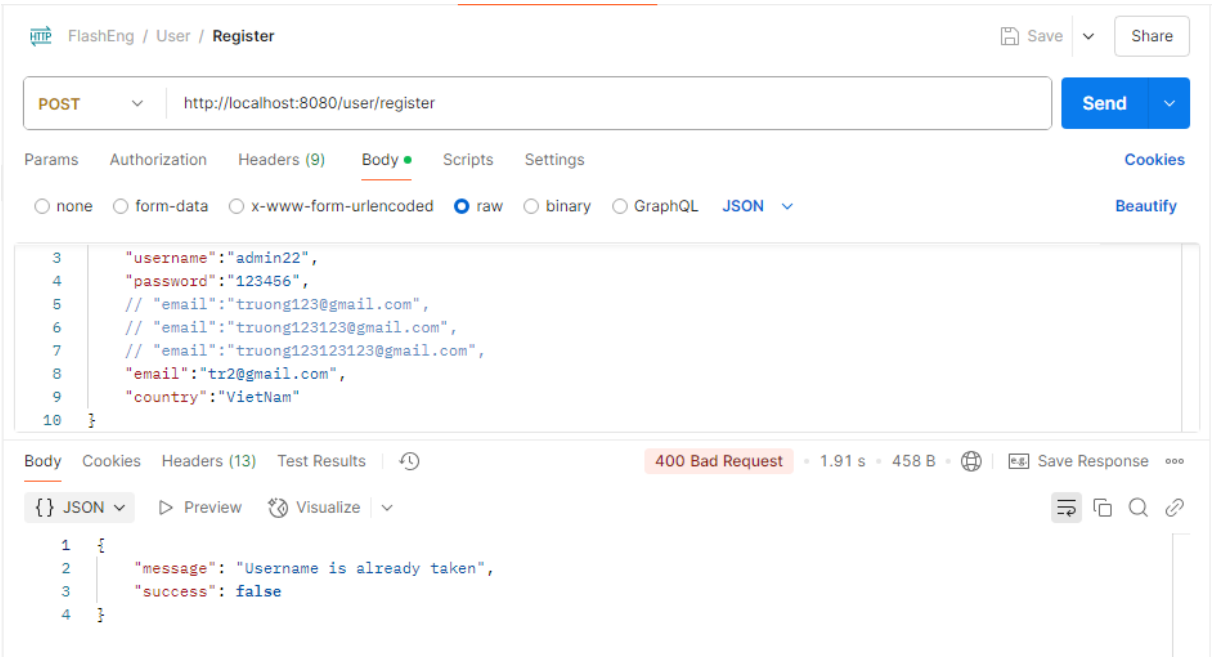
Repository là nơi trực tiếp làm việc với cơ sở dữ liệu. Dưới dạng các interface được Spring Data JPA tự động hiện thực hóa, Repository giúp trích xuất, lưu trữ, cập nhật và xóa dữ liệu từ các bảng. Mọi thao tác với dữ liệu đều đi qua Repository thay vì viết SQL thủ công. Repository giúp hệ thống truy xuất dữ liệu một cách an toàn, thống nhất và hiệu quả.

Entity là lớp mô hình hóa dữ liệu, ánh xạ trực tiếp với các bảng trong cơ sở dữ liệu. Mỗi instance của Entity đại diện cho một bản ghi (row), và các mối quan hệ giữa các bảng (one-to-many, many-to-many...) cũng được mô tả rõ ràng bằng các annotation như @OneToMany, @ManyToOne. Khi Repository lấy dữ liệu từ DB, nó trả về các đối tượng Entity để Service xử lý tiếp.

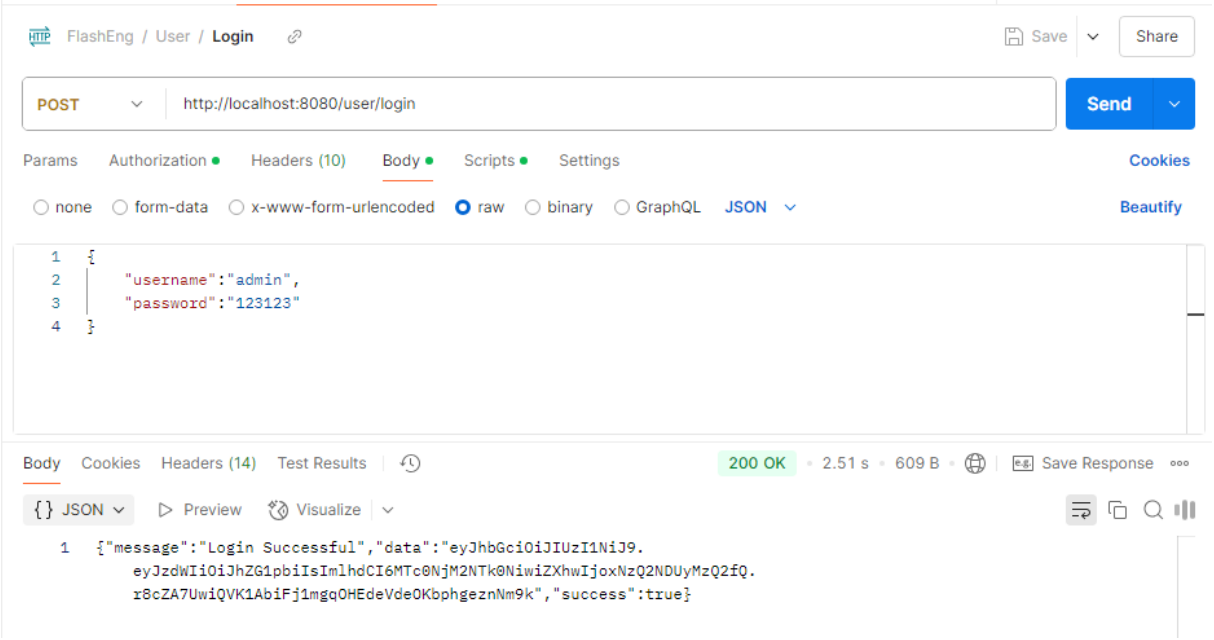
# B. Tiến độ dự án

- Hoàn thiện các API liên quan đến tài khoản người dùng:

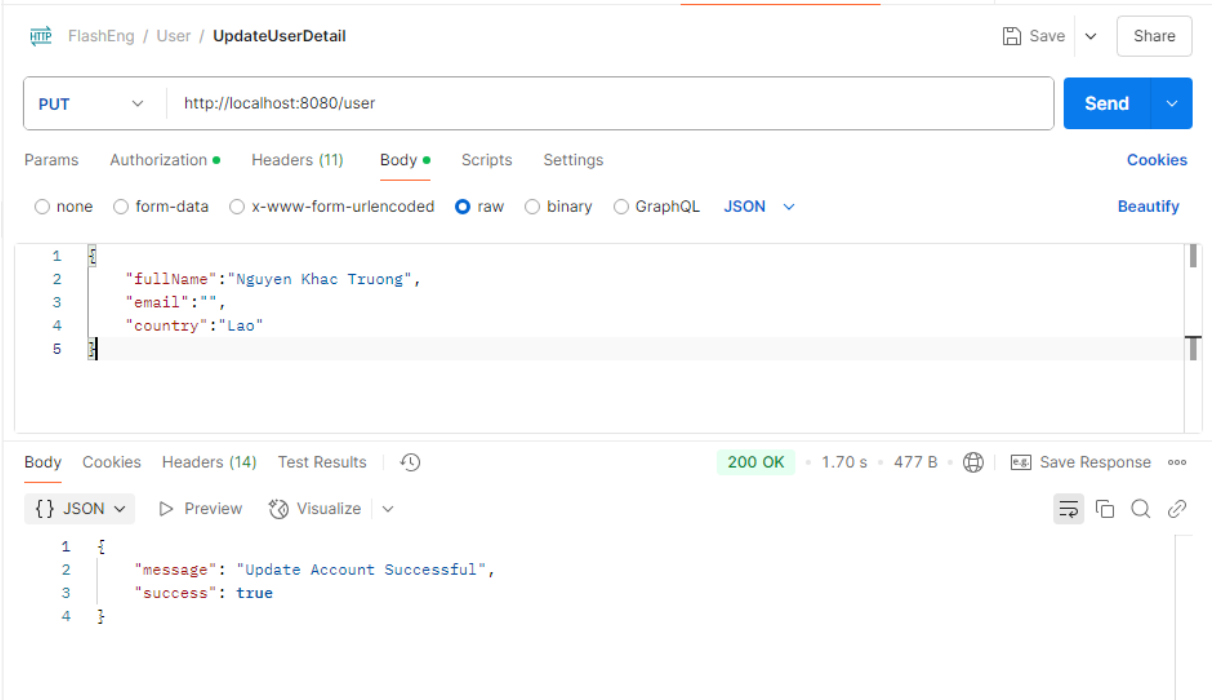
+ Đăng ký: Người dùng gửi username, email, password, country để tạo tài khoản, nếu email hay username tồn tại thì sẽ trả về lỗi



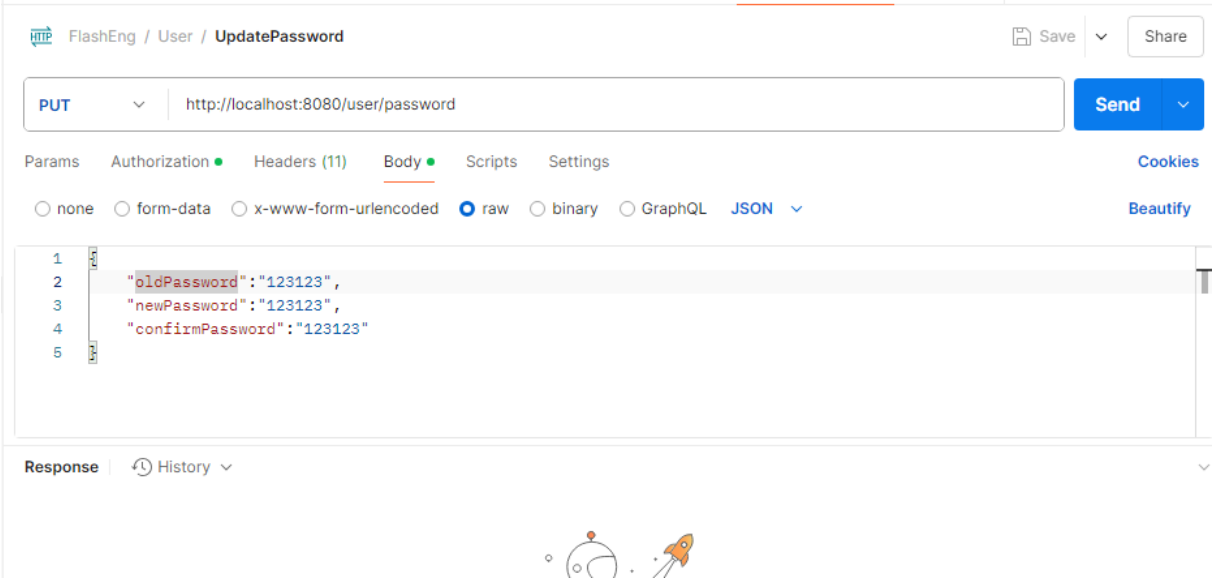

+ Đăng nhập: Người dùng gửi username và password để đăng nhập sau đó sẽ nhận lại token để việc đăng nhập lần tiếp là tự động, nếu sai 1 trong 2 trường thì sẽ trả về lỗi



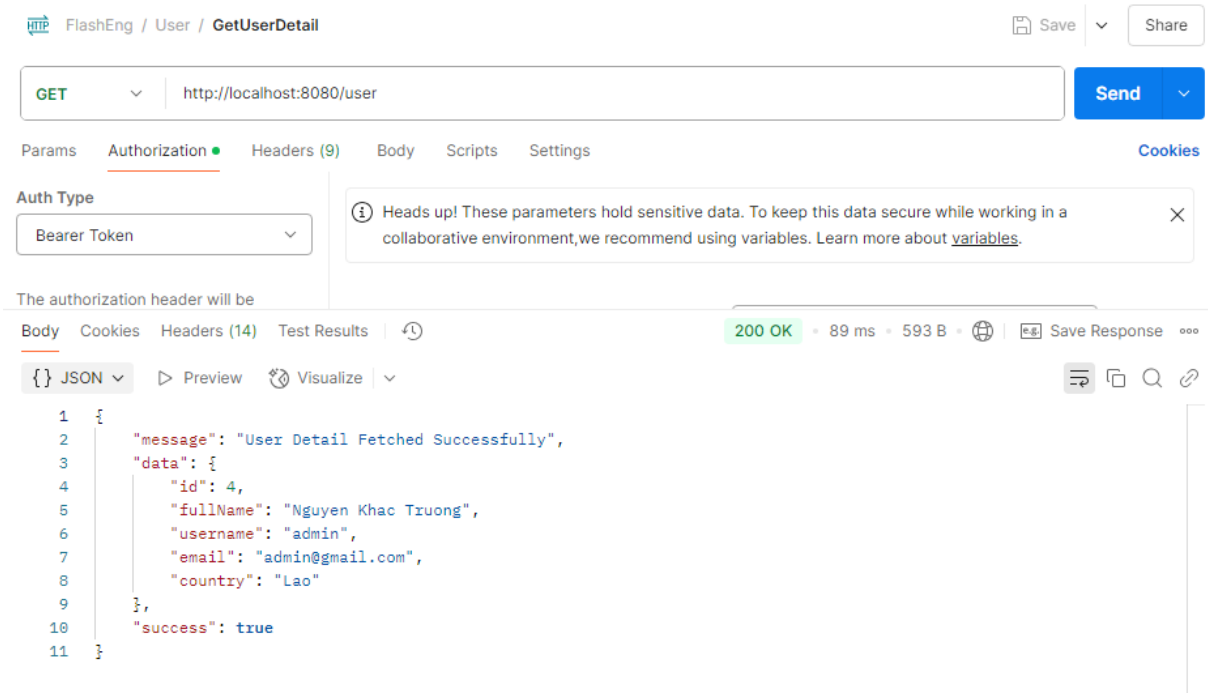
+ Sửa thông tin cá nhân: Người dùng gửi thông tin cần sửa kèm token để xác minh danh tinh, nếu thành công thì sẽ nhận được thông báo thành công.




+ Sửa mật khẩu: Người dùng gửi token để xác minh danh tính đồng thời nhập mật khẩu cũ và mới




+ Lấy thông tin cá nhân: Người dùng gửi token để xác mình danh tính rồi có thể lấy thông tin.


FlashEng / User / GetUserDetail
Save
Share
GET
http://localhost:8080/user
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
Auth Type
Bearer Token
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.
The authorization header will be
Body
Cookies
Headers (14)
Test Results
200 OK
89 ms
593 B
Save Response
JSON
Preview
Visualize
{
"message": "User Detail Fetched Successfully",
"data": {
"id": 4,
"fullName": "Nguyen Khac Truong",
"username": "admin",
"email": "admin@gmail.com",
"country": "Lao"
},
"success": true
}