

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

BÁO CÁO ĐỒ ÁN CUỐI KỲ  
MÔN: MẬT MÃ HỌC

CLOUD-NATIVE API-BASED NETWORK APPLICATION  
SECURITY FOR SMALL COMPANY SERVICES

TP. HỒ CHÍ MINH, NĂM 2025

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

BÁO CÁO ĐỒ ÁN CUỐI KỲ  
MÔN: MẬT MÃ HỌC

CLOUD-NATIVE API-BASED NETWORK APPLICATION  
SECURITY FOR SMALL COMPANY SERVICES

GIẢNG VIÊN HƯỚNG DẪN  
TS. NGUYỄN NGỌC TỰ

SINH VIÊN THỰC HIỆN  
TRẦN DƯƠNG MINH ĐẠI - 22520183  
HOÀNG NGỌC KHÁNH - 23520717

TP. HỒ CHÍ MINH, NĂM 2025

## LỜI CẢM ƠN

Lời đầu tiên nhóm xin gửi cảm ơn tới thầy Nguyễn Ngọc Tự, người đã tận tình giảng dạy, giúp đỡ nhóm trong suốt quá trình thực hiện đề tài. Mỗi bài giảng của thầy đều giúp nhóm chúng em hiểu biết hơn về những vấn đề xung quanh và từ đó có thể đưa ra những vận dụng thực tiễn trong cuộc sống. Cảm ơn những chia sẻ vô cùng quý báu từ thầy, nhờ đó mà nhóm đã có thêm nhiều kiến thức và kỹ năng cần thiết để thực hiện đồ án lần này.

Trong quá trình thực hiện đồ án, bằng những kiến thức và kỹ năng của mình, nhóm đã rất cố gắng và nỗ lực hết mình để hoàn thành thật tốt đề tài đã chọn. Tuy nhiên, vì kinh nghiệm còn hạn chế nên nhóm không thể tránh khỏi những sai sót. Rất mong thầy thông cảm và chia sẻ những góp ý để nhóm có thể chỉnh sửa và hoàn thiện đề tài hơn trong tương lai. Chúc thầy luôn nhiều sức khỏe và tiếp tục chia sẻ thêm kiến thức bổ ích như vậy đến với các bạn sinh viên.

Nhóm xin chân thành cảm ơn!

**Nhóm thực hiện.**

## MỤC LỤC

Chương 1. TỔNG QUAN ĐỀ TÀI .....	3
1.1. Giới thiệu .....	3
1.2. Ngữ cảnh ứng dụng của đề tài.....	3
1.3. Các bên liên quan.....	3
1.3.1. Sơ đồ hệ thống ứng dụng .....	3
1.3.2. Chi tiết các bên liên quan .....	3
1.4. Các giải thuyết đặt ra và yêu cầu bảo mật.....	4
1.4.1. Các giải thuyết đặt ra.....	4
1.4.2. Yêu cầu bảo mật .....	4
1.5. Các mối đe dọa .....	5
Tấn công API (API Attacks): .....	5
Chương 2. HƯỚNG NGHIÊN CỨU CHO ĐỀ TÀI .....	5
2.1. Kiến trúc tổng thể của ứng dụng.....	5
2.2. Xác thực và phân quyền người dùng với JWT.....	6
2.3. Quản lý session của người dùng với JWT.....	7
2.4. Mã hóa đường truyền với TLS .....	7
2.5. Kịch bản triển khai trên môi trường Cloud.....	7
2.5.1. Định tuyến và bảo mật đầu vào .....	8
2.5.2. Mã hóa và quản lý khóa .....	8
2.5.3. Load Balancer và phân tách mạng.....	9
2.5.4. Triển khai các services trên ECS Fargate .....	9
Chương 3. HIỆN THỰC ỨNG DỤNG VÀ TRIỂN KHAI THỬ NGHIỆM .....	10

3.1.	Hiện thực kiến trúc ứng dụng .....	10
3.2.	Hiện thực chức năng xác thực và phân quyền ( Authentication & Authorization) .....	12
3.2.1.	Authentication .....	12
3.2.2.	Authorization với Role-based Access Control .....	27
3.3.	Quản lý phiên làm việc (Session Management) .....	39
3.4.	Mã hóa truyền tải ( TLS Encryption) .....	44
3.5.	Triển khai thử nghiệm.....	48
3.5.1.	Triển khai Frontend .....	48
3.5.2.	Triển khai Backend Microservices.....	52

## **DANH MỤC HÌNH**

Hình 1. Minh họa tổng thể kiến trúc ứng dụng của đề tài.....	6
Hình 2. Kiến trúc triển khai .....	8
Hình 3. Luồng hoạt động của Amazon Elastic Container Service (ECS) .....	10
Hình 4. Register thành công trên Postman .....	26
Hình 5. Login thành công trên Postman .....	27
Hình 6. Đăng nhập thành công trên Web UI .....	27
Hình 7. Đăng nhập để lấy Access Token .....	37
Hình 8. User minhdai1234 với role USER trên mongodb.....	38
Hình 9. Kết quả khi test với account có role là USER.....	38
Hình 10. User superadmin với role là SUPER_ADMIN trên mongodb .....	38
Hình 11. Kết quả get toàn bộ user với account SUPER_ADMIN .....	39
Hình 12. Thông báo popup gia hạn session trên vousx-platform.shop .....	44
Hình 13. Tạo public key từ private key vousx-platform.shop.key .....	44
Hình 14. Thông tin chi tiết vousx-platform.shop.csr .....	45
Hình 15. Kết quả ký 2 certificate trên ZeroSSL.....	45
Hình 16. Folder của domain api.voux-platform.shop .....	46
Hình 17. Folder của domain vousx-platform.shop .....	46
Hình 18. Certificate lưu trên AWS Certificate Manager .....	47
Hình 19. KMS Key tạo trên AWS KMS .....	47
Hình 20. Kết quả tạo Secret trên Secret Manager .....	47
Hình 21. Lưu thành công key trên Secret Manager .....	48
Hình 22. Kết quả tạo hosted zone với tên miền vousx-platform.shop .....	48
Hình 23. Chi tiết các Record trong vousx-platform.shop .....	50
Hình 24. Chi tiết setting của Cloudfront .....	50
Hình 25. Origin path của Cloudfront .....	51
Hình 26. Các object đã build của Frontend trong S3.....	51
Hình 27. S3 Website Endpoint tính năng chi tiết.....	51
Hình 28. Hai Certificate lưu ở AWS Certificate Manager .....	52

Hình 29. Tham chiếu private key trên AWS Key Management Store .....	52
Hình 30. Kết quả tạo VPC.....	57
Hình 31. Các subnet có trong mạng VPC .....	58
Hình 32. Route table để route traffic.....	58
Hình 33. Kết quả tạo NAT Gatewat .....	58
Hình 34. Kết quả tạo Internet Gateway.....	59
Hình 35. Setting Service Connect trong Auth-service .....	64
Hình 36. Kết quả deploy các Services trên cụm ECS .....	65
Hình 37. Kết quả tạo Load Balancer .....	66
Hình 38. Chi tiết Resource Map của Load Balancer.....	66
Hình 39. Thêm Custom Certificate từ ZeroSSL vào Load Balancer Target Group .....	67
Hình 40. Web UI Chính của vousx-platform.shop .....	67
Hình 41. UI khi user đã đăng nhập.....	68
Hình 42. Certificate trên domain vousx-platform.shop .....	68
Hình 43. Phiên bản TLS sử dụng là TLS 1.3 cho domain vousx-platform.shop ...	69
Hình 44. Phiên bản TLS sử dụng là TLS 1.3 cho domain api.voux-platform.shop .....	69
Hình 45.Certificate trên api.voux-platform.shop .....	70

# **Chương 1. TỔNG QUAN ĐỀ TÀI**

## **1.1. Giới thiệu**

- Tên đề tài: Cloud-native API -based network application security for small company services.
- Tên môn học: Mật mã học (NT219.P21.ANTN).
- Nhóm sinh viên thực hiện :
  - + Hoàng Ngọc Khánh - 23529717
  - + Trần Dương Minh Đại - 22520183
- Phân chia công việc:

Họ và tên	
Trần Dương Minh Đại	Triển khai web app lên môi trường Cloud, viết báo cáo
Hoàng Ngọc Khánh	Implement các services, viết báo cáo

- Link project trên Github: <https://github.com/tdmida/NT219-Project-Cloud-API-Security>

## **1.2. Ngữ cảnh ứng dụng của đề tài**

- Trong thời đại chuyển đổi số, nhiều doanh nghiệp nhỏ ngày càng tận dụng công nghệ để phát triển các dịch vụ trực tuyến như thương mại điện tử, quản lý khách hàng, hay tiện ích số. Nhằm tiết kiệm chi phí và linh hoạt trong vận hành, họ chuyển sang kiến trúc cloud-native và microservice. Kiến trúc này cho phép xây dựng và triển khai ứng dụng hoàn toàn trên nền tảng đám mây, với khả năng mở rộng linh hoạt và tối ưu tài nguyên. Các microservice thường giao tiếp qua API, tạo nên một hệ sinh thái dễ tích hợp và mở rộng. Tuy nhiên, việc sử dụng API cũng kéo theo nhiều rủi ro bảo mật, đặc biệt là với các doanh nghiệp nhỏ thiếu kinh nghiệm và nguồn lực. Các mối đe dọa như tấn công API, đánh cắp dữ liệu, giả mạo đăng nhập, DoS/DDoS, và lối xác thực/ủy quyền có thể gây ảnh hưởng nghiêm trọng đến hoạt động và uy tín doanh nghiệp.

## **1.3. Các bên liên quan**

### **1.3.1. Sơ đồ hệ thống ứng dụng**

- Thêm ảnh

### **1.3.2. Chi tiết các bên liên quan**

- **Người dùng cuối (Users):**

Truy cập vào hệ thống từ máy tính hoặc điện thoại thông qua ứng dụng web .

- **API Gateway:**

Là điểm truy cập duy nhất của hệ thống, nhận các request từ người dùng cuối, chuyển tiếp đến các microservice phù hợp (như Auth, User, Cart, Voucher).

- **Auth Service:**

Chịu trách nhiệm xác thực người dùng. Khi nhận được thông tin đăng nhập (username, password) từ API Gateway, Auth Service kiểm tra tính hợp lệ và trả về token xác thực (thường là JWT).

- **Task Services (Các microservice chức năng):**

Bao gồm các dịch vụ như User Service, Cart Service, Voucher Service, v.v. Mỗi service đảm nhiệm một chức năng riêng biệt.

- **Certificate Authority (CA):**

Là bên thứ ba cung cấp chứng chỉ số (certificate) cho hệ thống, đảm bảo các kết nối giữa client và server, cũng như giữa các microservice với nhau được mã hóa và an toàn.

- **Attacker (Kẻ tấn công):**

Là các đối tượng cố gắng truy cập trái phép vào hệ thống.

## 1.4. Các giải thuyết đặt ra và yêu cầu bảo mật

### 1.4.1. Các giải thuyết đặt ra

- Server được đặt trên cloud.
- Các microservice trong hệ thống giao tiếp với nhau chủ yếu thông qua API.
- Kẻ tấn công có thể can thiệp giữa đường truyền.

### 1.4.2. Yêu cầu bảo mật

- Xác thực và phân quyền: Xác thực chính xác mọi người dùng trước khi truy cập tài nguyên.
- Bảo vệ API: API gateway phải kiểm soát, giới hạn lưu lượng, phát hiện và ngăn chặn các request bất thường.
- Mã hóa dữ liệu: Tất cả dữ liệu nhạy cảm đều được mã hóa khi truyền tải(sử dụng TLS) và khi lưu trữ.
- Bảo vệ Token: Token xác thực (JWT, OAuth2) phải được bảo vệ khỏi bị đánh cắp hoặc giả mạo.

- Quản lý certificate: Chứng chỉ số phải được quản lý chặt chẽ, đảm bảo luôn hợp lệ, không bị hết hạn hoặc bị giả mạo.

## 1.5. Các mối đe dọa

### Tấn công API (API Attacks):

- Gửi các request bất thường để dò tìm điểm yếu hoặc gây lỗi cho hệ thống.
- Khai thác lỗ hổng xác thực và phân quyền. (có thể truy cập các tài nguyên trái phép)
- Tấn công từ chối dịch vụ (Dos) qua API.

## Chương 2. HƯỚNG NGHIÊN CỨU CHO ĐỀ TÀI

### 2.1. Kiến trúc tổng thể của ứng dụng

Để thực hiện đề tài, nhóm chúng em lựa chọn xây dựng một web application như là hướng tiếp cận chính. Lý do chọn web app là vì đây là một nền tảng phổ biến, dễ dàng triển khai và truy cập thông qua trình duyệt trên mọi thiết bị. Web app cũng giúp nhóm dễ dàng thử nghiệm, kiểm tra bảo mật API và mô phỏng các hành vi thực tế của người dùng trong hệ thống.

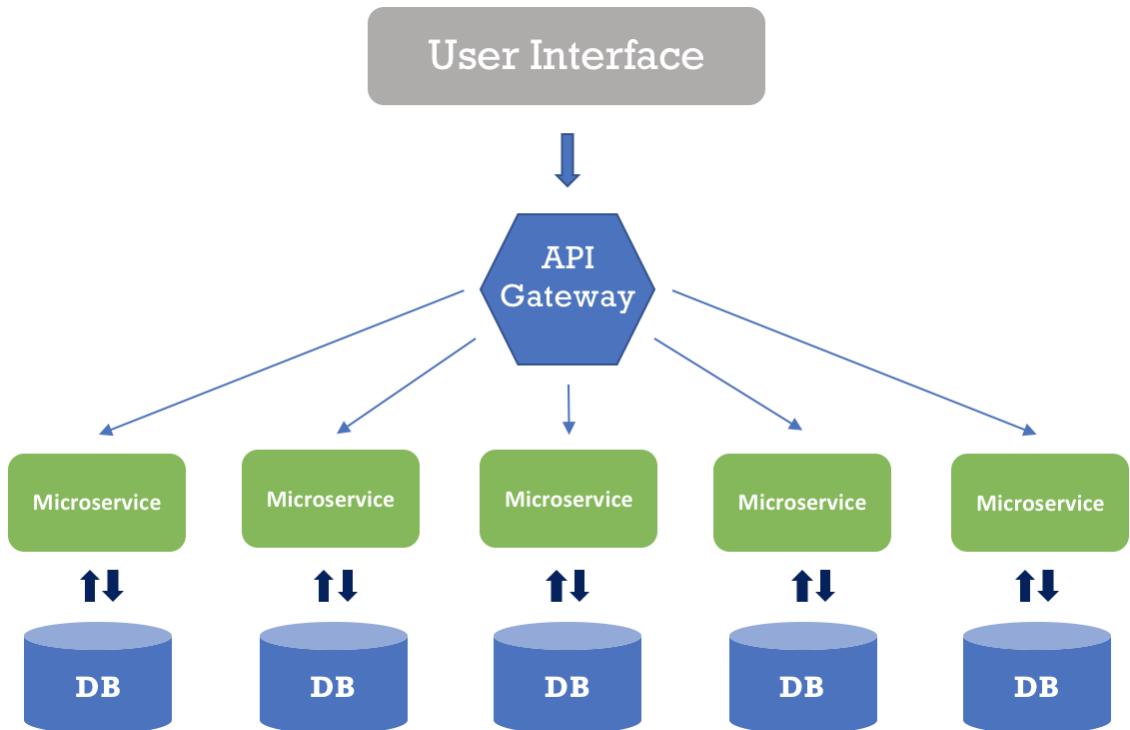
Kiến trúc tổng thể của ứng dụng được chia thành 4 phần chính như sau:

- **UI (Frontend):** Giao diện người dùng được xây dựng bằng ReactJS, chịu trách nhiệm hiển thị và tiếp nhận thao tác của người dùng.
- **API Gateway:** Là điểm vào duy nhất của hệ thống, nơi tiếp nhận mọi request từ phía frontend. Gateway sẽ thực hiện xác thực token, định tuyến các yêu cầu đến đúng microservice tương ứng và trả kết quả về cho client. Đây cũng là nơi triển khai các chức năng như kiểm soát truy cập, ghi log, giới hạn tốc độ (rate limiting), và bảo vệ hệ thống khỏi các cuộc tấn công phổ biến.
- **Backend Microservices:** Gồm bốn microservices chính là Auth Service, User Service, Voucher Service và Cart Service, mỗi service đảm nhiệm một nhóm chức năng nghiệp vụ cụ thể.
- **Database:** Là nơi lưu trữ dữ liệu phục vụ cho từng service. Nhóm sử dụng MongoDB vì tính linh hoạt, khả năng mở rộng và phù hợp với kiến trúc microservice (mỗi service có thể có database riêng biệt).

Luồng hoạt động cơ bản như sau: Người dùng sẽ truy cập vào giao diện web (UI) để thực hiện các thao tác như đăng nhập, xem thông tin cá nhân hoặc quản

lý giờ hàng. Các thao tác này sẽ được gửi đến API Gateway, nơi kiểm tra tính hợp lệ của yêu cầu (ví dụ: kiểm tra access token), sau đó định tuyến đến các backend service phù hợp để xử lý từng chức năng riêng. Cuối cùng, dữ liệu được truy xuất từ database, xử lý xong và phản hồi trở lại cho người dùng thông qua gateway.

Hình minh họa kiến trúc hệ thống bên dưới thể hiện rõ luồng xử lý giữa các thành phần nói trên.



Hình 1. Minh họa tổng thể kiến trúc ứng dụng của đề tài

## 2.2. Xác thực và phân quyền người dùng với JWT

Hệ thống sử dụng JSON Web Token (JWT) để xác thực và phân quyền người dùng. Khi người dùng đăng nhập thành công, Auth Service sẽ tạo ra một JWT chứa các thông tin định danh (user ID, vai trò, thời gian hiệu lực, v.v.) và trả về cho client. Mỗi lần người dùng gửi request đến hệ thống, JWT sẽ được đính kèm trong header (Authorization: Bearer <token>).

API Gateway và các microservice sẽ kiểm tra tính hợp lệ của JWT (chữ ký, thời gian hết hạn, v.v.) trước khi xử lý request. Dựa vào thông tin vai trò (role) trong JWT, hệ thống sẽ phân quyền truy cập tài nguyên phù hợp, đảm bảo người dùng chỉ được phép thực hiện các chức năng đúng với quyền hạn của mình (ví dụ: user thường không thể truy cập chức năng quản trị).

### **2.3. Quản lý session của người dùng với JWT**

Khác với session truyền thống lưu trên server, hệ thống sử dụng JWT để quản lý phiên đăng nhập(session) của người dùng một cách phi trạng thái.

Mỗi JWT có thời gian hiệu lực nhất định (expiration). Khi token hết hạn, người dùng cần đăng nhập lại hoặc sử dụng cơ chế refresh token để lấy token mới.

- ⇒ Việc sử dụng JWT giúp hệ thống dễ dàng mở rộng (scalable), giảm tải cho server vì không cần lưu trữ session phía backend.

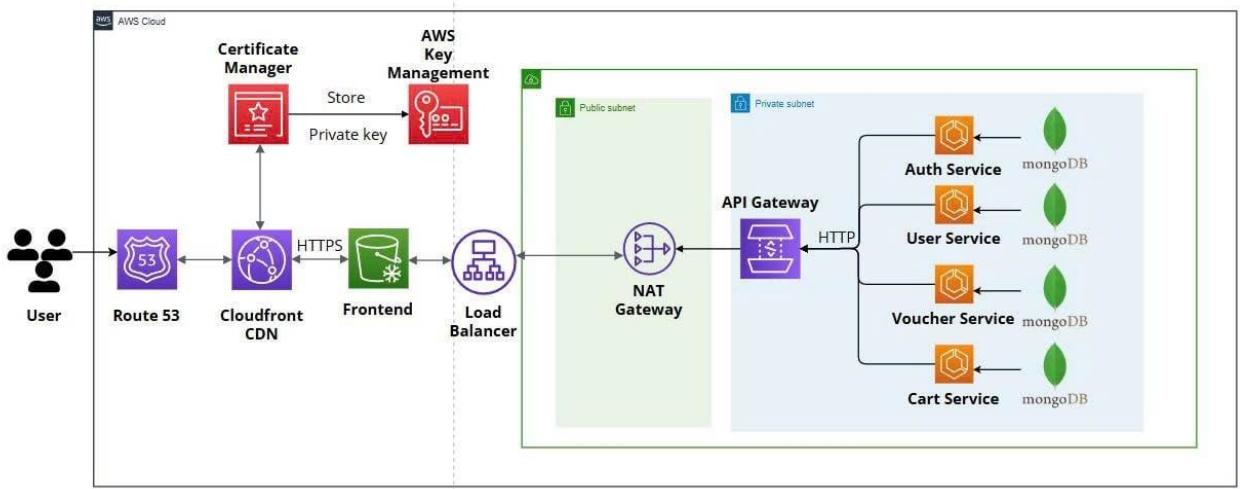
### **2.4. Mã hóa đường truyền với TLS**

TLS đảm bảo mọi dữ liệu trao đổi đều được mã hóa, ngăn chặn các cuộc tấn công nghe lén (eavesdropping) hoặc giả mạo (man-in-the-middle).

Hệ thống sử dụng chứng chỉ số (SSL/TLS certificate) do Certificate Authority (CA) uy tín cấp phát.

### **2.5. Kịch bản triển khai trên môi trường Cloud**

Hệ thống được triển khai trên nền tảng AWS Cloud nhằm tận dụng các dịch vụ hạ tầng sẵn có, đảm bảo khả năng mở rộng, tính sẵn sàng cao và bảo mật chặt chẽ. Như thể hiện trong Hình hệ thống được tổ chức thành ba lớp chính: lớp giao tiếp người dùng (Frontend), lớp trung gian định tuyến và xử lý API (API Gateway), và lớp các dịch vụ phía sau (Microservices và cơ sở dữ liệu).



Hình 2. Kiến trúc triển khai

### 2.5.1. Định tuyến và bảo mật đầu vào

- Người dùng truy cập hệ thống thông qua dịch vụ **Route 53**, đây là dịch vụ DNS có khả năng phân giải tên miền với độ trễ thấp và độ tin cậy cao. So với các dịch vụ DNS truyền thống, Route 53 hỗ trợ tích hợp tốt với các tài nguyên AWS như CloudFront, giúp định tuyến linh hoạt đến các vùng địa lý gần người dùng.

Để tối ưu hiệu năng và bảo mật truyền tải, nhóm sẽ sử dụng **CloudFront CDN (Content Delivery Network)**. CloudFront đóng vai trò phân phối nội dung tĩnh như HTML, CSS, JS đến người dùng cuối với độ trễ tối thiểu. Origin của CloudFront sẽ nối tới **AWS S3**, nơi để host giao diện web của hệ thống. Việc lựa chọn S3 làm nơi lưu trữ frontend giúp tối ưu chi phí, đảm bảo performance. Đồng thời, Cloudfront còn đảm bảo rằng mọi giao tiếp giữa người dùng và frontend đều được mã hóa qua HTTPS, sử dụng chứng chỉ custom SSL/TLS của nhóm tự upload lên, được ký bởi ZeroSSL và quản lý bởi **AWS Certificate Manager (ACM)**.

### 2.5.2. Mã hóa và quản lý khóa

Nhóm sẽ lưu 2 certificate được ký bởi ZeroSSL trên **AWS Certificate Manager (ACM)** là certificate cho kết nối giữa User ngoài internet đến Cloudfront (CN name là `voux-platform.shop`) và từ Cloudfront đến API Gateway (CN name là `api.voux-platform.shop`). Khóa bí mật (private key) được lưu trữ và bảo vệ bằng **AWS Key Management Service (KMS)** – một giải pháp mã hóa do AWS cung cấp, hỗ trợ quản lý vòng đời khóa, kiểm soát truy cập và tích hợp dễ dàng với các dịch vụ khác.

Nhóm lựa chọn KMS thay vì tự lưu trữ khóa là do KMS tuân thủ các tiêu chuẩn bảo mật như FIPS 140-2 và hỗ trợ audit log thông qua CloudTrail.

### 2.5.3. Load Balancer và phân tách mạng

Sau khi người dùng được xác thực và request được định tuyến qua CloudFront, yêu cầu được chuyển tiếp đến Load Balancer. Load Balancer giúp phân phối lưu lượng đều đến các tài nguyên backend, tránh tình trạng quá tải và tăng tính sẵn sàng.

Trong kiến trúc này, nhóm sử dụng Application Load Balancer (ALB) để hỗ trợ routing thông minh theo URL path (ví dụ: /auth, /user), phù hợp với kiến trúc microservice.

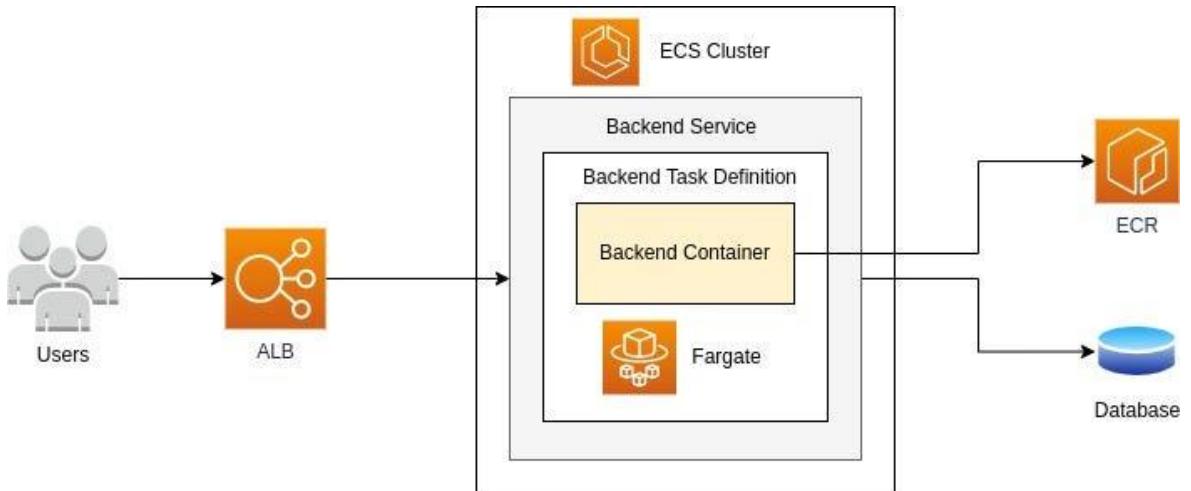
Đằng sau Load Balancer là một NAT Gateway trong được triển khai VPC đóng vai trò kết nối các tài nguyên trong subnet riêng (private subnet) ra ngoài Internet một cách an toàn, mà không phơi bày các dịch vụ nội bộ.

### 2.5.4. Triển khai các services trên ECS Fargate

Các microservice của hệ thống gồm Auth Service, User Service, Voucher Service và Cart Service sẽ được đóng gói dưới dạng Docker image và triển khai trên **Amazon ECS** sử dụng chế độ Fargate. Việc sử dụng ECS Fargate giúp đơn giản hóa quá trình triển khai vì không cần quản lý hạ tầng máy chủ cũng như tiết kiệm chi phí vì Fargate hoạt động theo serverless (chỉ trả tiền khi dùng).

Các Docker image của từng service sẽ được lưu trữ trong **Amazon ECR (Elastic Container Registry)**. Đây là nơi lưu trữ image an toàn, dễ tích hợp với ECS, đồng thời hỗ trợ kiểm soát truy cập bằng IAM.

Như minh họa hình 3, tất cả các yêu cầu từ phía người dùng sau khi đi qua Load Balancer và sẽ được chuyển tới các services chạy trên ECS, mỗi services sẽ chạy 1 task được định nghĩa sẵn gọi là Task Definition, các Task này sẽ lấy các Docker Image từ ECR về. Tại đây, request được định tuyến đến đúng container tương ứng đang chạy trên ECS.



*Hình 3. Luồng hoạt động của Amazon Elastic Container Service (ECS)*

### **Chương 3. HIỆN THỰC ỨNG DỤNG VÀ TRIỂN KHAI THỬ NGHIỆM**

#### **3.1. Hiện thực kiến trúc ứng dụng**

Như đã đề cập ở hướng nghiên cứu của đề tài, kiến trúc tổng thể ứng dụng của nhóm sẽ gồm 3 phần : Frontend, Backend microservices và Database.

Đầu tiên, về mặt giao diện người dùng (UI), nhóm sử dụng **ReactJS** – một thư viện JavaScript nổi tiếng được phát triển bởi Facebook, hỗ trợ xây dựng giao diện người dùng động, hiện đại và dễ mở rộng. ReactJS giúp nhóm chia nhỏ các thành phần giao diện thành các component độc lập, thuận tiện trong việc tái sử dụng, kiểm thử và duy trì mã nguồn. Ngoài ra, React cũng có hệ sinh thái phong phú, tích hợp tốt với các thư viện như Axios (gọi API), React Router (điều hướng), và các thư viện quản lý trạng thái như Redux hoặc Context API.

Về phía backend, nhóm sử dụng ngôn ngữ Python kết hợp với framework **FastAPI**. FastAPI là một framework hiện đại, nhẹ, hiệu suất cao, hỗ trợ chuẩn OpenAPI và có khả năng tạo tài liệu API tự động. FastAPI cũng tích hợp tốt với cơ chế *async* trong Python, giúp hệ thống có thể xử lý song song nhiều yêu cầu HTTP, cải thiện tốc độ phản hồi. Ngoài ra, cú pháp đơn giản và rõ ràng của FastAPI cũng giúp nhóm dễ dàng hiện thực các middleware phục vụ bảo mật như xác thực, phân quyền, kiểm soát truy cập,...

Để quản lý và tổ chức hệ thống một cách rõ ràng và có thể mở rộng về sau, nhóm chia hệ thống backend thành bốn microservices chính như sau:

- API Gateway Service
  - Đây là cổng giao tiếp trung tâm giữa client và các service bên trong hệ thống. API Gateway chịu trách nhiệm điều phối yêu cầu, định tuyến request đến đúng microservice tương ứng (như Auth, User, Voucher, Cart), tổng hợp phản hồi và trả về kết quả cho client.
- Auth Service
  - Chịu trách nhiệm xử lý đăng ký, đăng nhập, cấp phát token (access token và refresh token), băm mật khẩu, và xác thực người dùng. Service này cũng là nơi triển khai Authentication và một phần Session Management.
- User Service
  - Xử lý các chức năng liên quan đến thông tin người dùng, cập nhật thông tin cá nhân, kiểm tra trạng thái tài khoản. Các endpoints trong User Service sẽ cần được bảo vệ bởi các cơ chế Authorization và Role-Based Access Control (RBAC) để giới hạn quyền truy cập dựa trên vai trò.
- Voucher Service
  - Quản lý mã giảm giá (voucher) như tạo, cập nhật, xóa và áp dụng voucher cho người dùng. Do dữ liệu của voucher thường linh hoạt (các trường có thể thay đổi tùy theo từng loại chiến dịch),
- Cart Service
  - Chịu trách nhiệm quản lý toàn bộ hoạt động liên quan đến giỏ hàng của người dùng, bao gồm thêm, xoá, cập nhật sản phẩm, tính tổng giá trị đơn hàng và áp dụng mã giảm giá. Service này tương tác với User Service để xác thực người dùng và với Voucher Service để kiểm tra tính hợp lệ của mã khuyến mãi.

Do dữ liệu của voucher thường linh hoạt và thay đổi theo từng loại đợt giảm giá, nhóm chọn sử dụng **MongoDB** – một hệ quản trị cơ sở dữ liệu NoSQL dạng document – vì không yêu cầu schema cố định, cho phép lưu trữ cấu trúc dữ liệu lồng nhau hiệu quả, dễ dàng mở rộng ngang và đặc biệt phù hợp với kiến trúc microservice, nơi mỗi service có thể sở hữu database riêng biệt.

Tiếp theo, do để tài tập trung vào khía cạnh bảo mật cho API, nhóm sẽ tập trung triển khai và phân tích bốn chức năng bảo mật chính, bao gồm:

- Xác thực và phân quyền (**Authentication & Authorization**)

- Quản lý phiên làm việc (**Session Management**)
- Mã hóa truyền tải bằng TLS (**TLS Encryption**)

Các chức năng này sẽ được trình bày chi tiết trong từng mục nhỏ tương ứng ở các phần bên dưới. Mỗi phần sẽ mô tả vai trò, cách triển khai, cũng như các kỹ thuật hoặc công nghệ được sử dụng nhằm đảm bảo an toàn cho hệ thống API trong môi trường Cloud.

### **3.2. Hiện thực chức năng xác thực và phân quyền ( Authentication & Authorization)**

#### **3.2.1. Authentication**

Với phần hiện thực Authentication, nhóm sẽ dùng **JSON Web Token (JWT)** để xác thực người dùng trong hệ thống một cách an toàn và không trạng thái (stateless). Cơ chế này đảm bảo rằng mỗi request đều đi kèm một token đại diện cho danh tính người dùng, giúp loại bỏ nhu cầu lưu session trên server. Thuật toán được sử dụng để tạo chữ ký cho JWT là **HS256 (HMAC với SHA-256)**, đây là một thuật toán đối xứng, nghĩa là cùng một secret sẽ được sử dụng cho cả việc ký và kiểm tra chữ ký. Khóa bí mật này được đọc từ biến môi trường `JWT_ACCESS_KEY`, cấu hình sẵn trong file `.env`.

Luồng Authentication sẽ hoạt động như sau:

Client gửi thông tin đăng nhập → AuthController xác thực người dùng → JWT Access Token và Refresh Token được tạo → Token được trả về cho client → Với mỗi request sau đó, client gửi Access Token trong header → JWT Middleware kiểm tra tính hợp lệ → Nếu hợp lệ, request được xử lý, nếu không, client sẽ cần sử dụng Refresh Token để lấy token mới.

#### **JWT Middleware (shared/middleware.py)**

- Lớp AuthMiddleware chịu trách nhiệm kiểm tra tính hợp lệ của token ở mỗi request. Nó sử dụng SessionManager để quản lý session và kiểm tra blacklist. `jwt_secret` được lấy từ biến môi trường và mặc định là "your-secret-key" để tránh lỗi khi chạy ở môi trường dev.
- Source code:

```

- class AuthMiddleware:
-     """Enhanced JWT Authentication middleware with session
management"""
-
-     def __init__(self):

```



```
-         "rbac_role": payload.get("rbac_role", "USER"),
-         "auth_type": "jwt"
-     }

-
-     # Add RBAC info
-     user_data["rbac_role"] =
RBACManager.get_user_role(user_data)
-     user_data["permissions"] =
list(RBACManager.get_user_permissions(user_data))

-
-     return user_data

-
-     except jwt.ExpiredSignatureError:
-         raise HTTPException(
-             status_code=status.HTTP_401_UNAUTHORIZED,
-             detail={
-                 "success": False,
-                 "message": "Phiên đăng nhập đã hết hạn! Vui
lòng đăng nhập lại.",
-                 "error_code": "TOKEN_EXPIRED",
-                 "session_expired": True,
-                 "action_required": "LOGIN AGAIN"
-             }
-         )
-     except jwt.InvalidTokenError:
-         raise HTTPException(
-             status_code=status.HTTP_401_UNAUTHORIZED,
-             detail={
-                 "success": False,
-                 "message": "Token không hợp lệ! Vui lòng đăng
nhập lại.",
-                 "error_code": "INVALID_TOKEN",
-                 "action_required": "LOGIN AGAIN"
-             }
-         )
-
-     except HTTPException:
-         raise
-     except Exception as e:
-         logger.error(f"JWT authentication failed: {e}")
-         raise HTTPException(
-             status_code=status.HTTP_401_UNAUTHORIZED,
-             detail={
-                 "success": False,
-                 "message": "Authentication failed",
-                 "error": str(e)
-
```

```

-         }
-     )
-
-     def generate_jwt_token(self, user_data: Dict[str, Any], expires_delta: Optional[timedelta] = None) -> str:
-         """Generate JWT token"""
-         try:
-             if expires_delta:
-                 expire = datetime.utcnow() + expires_delta
-             else:
-                 expire = datetime.utcnow() + timedelta(seconds=45) # Default to 45 seconds (30s + 15s grace)
-
-             payload = {
-                 "user_id": user_data.get("user_id"),
-                 "username": user_data.get("username"),
-                 "rbac_role": user_data.get("rbac_role", "USER"),
-                 "exp": expire,
-                 "iat": datetime.utcnow(),
-                 "token_type": "access"
-             }
-
-             token = jwt.encode(payload, self.jwt_secret,
-                                 algorithm=self.jwt_algorithm)
-             return token
-
-         except Exception as e:
-             logger.error(f"JWT token generation failed: {e}")
-             raise Exception("Failed to generate JWT token")
-
```

### Hàm verify\_jwt\_token(...)

- Đầu tiên, middleware kiểm tra xem header Authorization có tồn tại hay không. Nếu thiếu, trả về lỗi 401. Nếu có, token sẽ được trích xuất từ header. Sau đó, token sẽ được kiểm tra xem có nằm trong danh sách blacklist hay không. Nếu bị blacklist (ví dụ sau khi logout), middleware sẽ từ chối truy cập.
- Tiếp theo, token sẽ được decode bằng jwt.decode(...) sử dụng jwt\_secret và thuật toán HS256. Middleware sẽ kiểm tra thời gian hết hạn (exp) để đảm bảo token vẫn còn hiệu lực. Nếu hợp lệ, middleware sẽ trả về payload bao gồm user\_id, username, role và auth\_type. Nếu không, các lỗi như ExpiredSignatureError hay InvalidTokenError sẽ được xử lý riêng biệt với thông báo cụ thể và ghi log chi tiết nhưng không làm lộ thông tin nội bộ ra ngoài.

## Token Generation

- Hàm generate\_jwt\_token(...) có nhiệm vụ tạo access token cho người dùng. Nó cho phép cấu hình thời gian sống của token qua biến expires\_delta. Nếu không truyền vào, mặc định token sẽ có thời hạn 1 giờ.
- Payload của token bao gồm:
  - o user\_id: ID của người dùng
  - o username: tên đăng nhập
  - o role: quyền hạn của người dùng (mặc định là "guest" nếu không có)
  - o exp: thời điểm hết hạn
  - o iat: thời điểm phát hành
  - o type: loại token (ở đây là "access")

Sau đó payload này sẽ được mã hóa bằng jwt.encode(...) và trả lại token dạng chuỗi.

- **User Registration và Login (auth\_controller.py)**
- Lớp AuthController xử lý logic đăng ký, đăng nhập và quản lý phiên đăng nhập. Nó sử dụng các thành phần như:
  - o AuthDatabase: tương tác với cơ sở dữ liệu người dùng
  - o AuthMiddleware: tạo và xác thực JWT
  - o SessionManager: quản lý session/refresh token
  - o RBACManager: quản lý phân quyền theo vai trò
  - o Các hàm chính như login(), register(), logout()

```
class AuthController:  
    """JWT-based authentication controller with session management"""  
  
    def __init__(self):  
        self.auth_middleware = AuthMiddleware()  
        self.jwt_secret = os.getenv("JWT_ACCESS_KEY", "your-secret-key")
```

## Hàm Register(): Hàm đăng ký người dùng mới

```
-  
-     async def register(self, user_data: UserCreate) -> Dict[str, Any]:  
-         """User registration with validation"""  
-         try:  
-             # Basic validation is handled by Pydantic  
-
```

```
-     # Password strength validation
-     if len(user_data.password) < 8:
-         raise HTTPException(
-             status_code=status.HTTP_400_BAD_REQUEST,
-             detail={
-                 "success": False,
-                 "message": "Mật khẩu phải có ít nhất 8 ký tự!"
-             }
-         )
-
-     # Check if user already exists
-     users_collection = AuthDatabase.get_collection("users")
-
-     existing_user = await users_collection.find_one({
-         "$or": [
-             {"username": user_data.username},
-             {"email": user_data.email}
-         ]
-     })
-
-     if existing_user:
-         raise HTTPException(
-             status_code=status.HTTP_400_BAD_REQUEST,
-             detail={
-                 "success": False,
-                 "message": "Username hoặc email đã tồn tại!"
-             }
-         )
-
-     # Hash password with salt
-     salt = bcrypt.gensalt(rounds=12)
-                                         hashed_password =
-     bcrypt.hashpw(user_data.password.encode('utf-8'), salt)
-
-     # Create new user with default role
-     user_dict = {
-         "username": user_data.username,
-         "email": user_data.email,
-         "password": hashed_password.decode('utf-8'),
-         "admin": False,
-         "roles": ["user"], # Default role
-         "created_at": datetime.utcnow(),
-         "updated_at": datetime.utcnow(),
-         "last_login": None,
-         "login_count": 0
```

```

    }

    result = await users_collection.insert_one(user_dict)

    if result.inserted_id:
        # Prepare user event data
        user_event_data = {
            "user_id": str(result.inserted_id),
            "username": user_data.username,
            "email": user_data.email,
            "admin": False,
            "roles": ["user"],
            "created_at": datetime.utcnow().isoformat(),
            "updated_at": datetime.utcnow().isoformat()
        }

        # Async publish event (non-blocking - fire and forget)
        try:
            asyncio.create_task(
                event_manager.publish_event("user.registered",
                user_event_data)
            )
            logger.info(f"Event publishing scheduled for user: {user_data.username}")
        except Exception as e:
            logger.warning(f"Could not schedule event publishing: {e}")
            # Continue - not critical for registration success

        return {
            "success": True,
            "message": "Đăng ký tài khoản thành công! 🎉"
        }
    else:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail={
                "success": False,
                "message": "Lỗi khi tạo tài khoản!"
            }
        )

except HTTPException:
    raise
except Exception as error:

```

```

-         logger.error(f"Registration error: {error}")
-         raise HTTPException(
-             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
-             detail={
-                 "success": False,
-                 "message": "Lỗi server! Vui lòng thử lại sau."
-             }
-         )

```

- Hàm register() chịu trách nhiệm xử lý quá trình đăng ký tài khoản. Đầu tiên, hệ thống sẽ nhận vào dữ liệu người dùng mới (username, email, password) và kiểm tra cơ bản thông qua Pydantic. Sau đó, một bước kiểm tra độ mạnh mật khẩu sẽ đảm bảo rằng mật khẩu có ít nhất 8 ký tự, nhằm tăng tính an toàn cho người dùng.
- Tiếp theo, hệ thống truy vấn database để kiểm tra xem username hoặc email đã tồn tại chưa. Nếu trùng, quá trình sẽ bị từ chối với thông báo lỗi phù hợp. Nếu không có xung đột, mật khẩu sẽ được băm bằng bcrypt với salt và lưu vào cơ sở dữ liệu cùng các thông tin mặc định như roles, admin, created\_at, login\_count,...
- Nếu việc tạo người dùng thành công, một sự kiện user.registered sẽ được khởi tạo và gửi đi theo dạng bất đồng bộ để phục vụ mục đích logging hoặc tracking sau này. Dù sự kiện có lỗi khi gửi, quá trình đăng ký vẫn hoàn tất vì không ảnh hưởng đến kết quả chính.

## Hàm login(): Hàm đăng nhập và sinh token

```

-     async def login(self, login_data: LoginRequest, request: Request)
-     -> Dict[str, Any]:
-         """Enhanced JWT-based login with refresh tokens"""
-         try:
-             logger.info(f"Login attempt for user: {login_data.username}")
-
-             # Authenticate user
-             users_collection = AuthDatabase.get_collection("users")
-             user = await users_collection.find_one({"username": login_data.username})
-
-             if not user:
-                 logger.warning(f"Login failed - user not found: {login_data.username}")
-                 raise HTTPException(

```

```
-             status_code=status.HTTP_401_UNAUTHORIZED,
-             detail={
-                 "success": False,
-                 "message": "Tên đăng nhập không đúng!"
-             }
-         )
-
-         # Verify password
-         if not bcrypt.checkpw(login_data.password.encode('utf-8'),
-             user["password"].encode('utf-8')):
-             logger.warning(f"Login failed - wrong password for
- user: {login_data.username}")
-             raise HTTPException(
-                 status_code=status.HTTP_401_UNAUTHORIZED,
-                 detail={
-                     "success": False,
-                     "message": "Mật khẩu không đúng!"
-                 }
-             )
-
-         # Update login statistics
-         try:
-             await users_collection.update_one(
-                 {"_id": user["_id"]},
-                 {
-                     "$set": {"last_login": datetime.utcnow()},
-                     "$inc": {"login_count": 1}
-                 }
-             )
-             logger.info(f"Updated login stats for user:
{login_data.username}")
-         except Exception as e:
-             logger.warning(f"Could not update login stats: {e}")
-             # Continue - not critical
-
-         # Generate tokens
-         user_id = str(user["_id"])
-         token_data = {
-             "user_id": user_id,
-             "username": user["username"],
-             "admin": user.get("admin", False),
-             "roles": user.get("roles", ["user"])
-         }
```

```
-             # Generate access token (45 seconds: 30s session + 15s
-             grace period for user decision)
-             try:
-                 access_token = self.auth_middleware.generate_jwt_token(
-                     token_data,
-                     expires_delta=timedelta(seconds=45)
-                 )
-                 logger.info(f"Generated access token for user:
{login_data.username}")
-             except Exception as e:
-                 logger.error(f"Failed to generate access token: {e}")
-                 raise HTTPException(
-                     status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
-                     detail={
-                         "success": False,
-                         "message": "Lỗi tạo token truy cập!"
-                     }
-                 )
-
-             # Generate refresh token (30 days or 7 days)
-             try:
-                 expiry_days = 30 if login_data.remember_me else 7
-                 logger.info(f"Creating refresh token for user:
{login_data.username}")
-                 refresh_token = await
- session_manager.create_refresh_token(
-                     user_id,
-                     device_info=request.headers.get("user-agent"),
-                     ip_address=request.client.host if hasattr(request,
- 'client') and request.client else None
-                 )
-                 logger.info(f"Successfully created refresh token for
user: {login_data.username}")
-             except Exception as e:
-                 logger.error(f"Failed to create refresh token for user
{login_data.username}: {e}")
-                 raise HTTPException(
-                     status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
-                     detail={
-                         "success": False,
-                         "message": "Lỗi tạo refresh token!",
-                         "error": str(e)
-                     }
-                 )
-
```

```
-     # Prepare secure user response (only essential information)
-     secure_user_info = {
-         "id": user_id,
-         "username": user["username"],
-         "admin": user.get("admin", False),
-             "avatar_url": user.get("avatar_url", "/default-
-         avatar.png"),
-             "theme": user.get("theme", "light")
-     }
-
-     # Add RBAC information
-     try:
-         rbac_role = RBACManager.get_user_role(token_data)
-                         permissions =
list(RBACManager.get_user_permissions(token_data))
-         except Exception as e:
-             logger.warning(f"RBAC error for user
{login_data.username}: {e}")
-             rbac_role = None
-             permissions = []
-
-             logger.info(f"Login successful for user:
{login_data.username}")
-
-             return {
-                 "success": True,
-                 "message": "Đăng nhập thành công! 🎉 (Phiên làm việc:
30 giây)",
-                 "user": secure_user_info,
-                 "access_token": access_token,
-                 "refresh_token": refresh_token,
-                 "token_type": "Bearer",
-                 "expires_in": 30, # Client sees 30 seconds, but token
valid for 45s
-                 "actual_token_expiry": 45, # Actual backend expiry
-                 "refresh_expires_in": 600, # 10 minutes (600 seconds)
-                 "auth_type": "jwt_with_refresh",
-                 "session_warning": "⚠ Bạn sẽ được hỏi có muốn tiếp
tục sau 30 giây!",
-                 "rbac": {
-                     "role": rbac_role.value if rbac_role else "user",
-                     "permissions": permissions
-                 }
-             }
-
```

```

-     except HTTPException:
-         raise
-     except Exception as error:
-         logger.error(f"Login error for user {login_data.username}:
{error}")
-         raise HTTPException(
-             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
-             detail={
-                 "success": False,
-                 "message": "Lỗi server khi đăng nhập!",
-                 "error": str(error)
-             }
-         )
-
```

- Hàm login nhận vào tên người dùng và mật khẩu, sau đó truy xuất dữ liệu người dùng từ MongoDB. Nếu không tìm thấy người dùng, hoặc mật khẩu không trùng khớp (kiểm tra bằng bcrypt.checkpw), hệ thống sẽ trả về lỗi 401 cùng với thông báo hợp lý, tránh rò rỉ thông tin.
- Khi xác thực thành công, hệ thống sẽ cập nhật lại last\_login và tăng login\_count. Sau đó, hệ thống tiến hành sinh access token (hết hạn sau 45 giây – 30 giây làm việc và 15 giây “grace period”) và một refresh token tùy thuộc vào lựa chọn “remember me” (30 ngày hoặc 7 ngày).
- Ngoài ra, hàm này cũng trả về thông tin người dùng an toàn như username, avatar\_url, theme... và đi kèm quyền hạn được lấy từ RBAC (role, permissions). Nhờ đó, phía frontend có thể hiện thị chức năng phù hợp tùy theo quyền.
- Việc cấu hình thời gian sống của token giúp tăng bảo mật khi làm việc với các phiên ngắn (ví dụ: thi trắc nghiệm hoặc điều khiển thời gian người dùng làm bài). Phản hồi cuối cùng của API login có cấu trúc rõ ràng, giúp frontend dễ dàng xử lý và hiển thị thông tin xác thực.

## Hàm logout(): Hàm đăng xuất và thu hồi token

```

-     async def logout(self, request: Request, logout_data:
Optional[LogoutRequest] = None, current_user: Optional[Dict[str, Any]] =
None) -> Dict[str, Any]:
-         """Enhanced logout with token revocation"""
-         try:
-             user_id = current_user.get("id") if current_user else
"unknown"
-
```

```
-         logger.info(f"Logout attempt for user: {user_id}")

-
# Get access token from request
auth_header = request.headers.get("authorization")
if auth_header and auth_header.startswith("Bearer "):
    access_token = auth_header.split(" ")[1]
    session_manager.blacklist_access_token(access_token)
    logger.info(f"Blacklisted access token for user:
{user_id}")

-
        logout_all = False

-
        if logout_data and logout_data.refresh_token:
            # Revoke specific refresh token
            logger.info(f"Revoking specific refresh token for user:
{user_id}")
            try:
                await
session_manager.revoke_refresh_token(logout_data.refresh_token)
            except Exception as e:
                logger.warning(f"Could not revoke specific refresh
token: {e}")

-
            elif logout_data and logout_data.logout_all and
current_user:
                # Revoke all user sessions
                logger.info(f"Revoking all sessions for user: {user_id}")
                logout_all = True
                try:
                    await
session_manager.revoke_all_user_tokens(current_user.get("id"))
                except Exception as e:
                    logger.warning(f"Could not revoke all user tokens:
{e}")

-
            elif current_user:
                # Revoke all sessions for current user (default behavior)
                logger.info(f"Revoking all sessions for current user:
{user_id}")
                logout_all = True
                try:
                    await
session_manager.revoke_all_user_tokens(current_user.get("id"))
                except Exception as e:
                    logger.warning(f"Could not revoke all user tokens:
{e}")
```

```

-         logger.info(f"Logout successful for user: {user_id}")
(logout_all: {logout_all}))"

-
    return {
        "success": True,
        "message": "Đăng xuất thành công! 🌟"
    }

-
except Exception as error:
    logger.error(f"Logout error for user {current_user.get('id')}")
if current_user else 'unknown'): {error}")
    # Return success anyway - logout should always appear to
work from user perspective
return {
    "success": True,
    "message": "Đăng xuất thành công! 🌟"
}
-

```

- Hàm logout xử lý việc thu hồi token để đảm bảo rằng sau khi người dùng đăng xuất, token không còn sử dụng được nữa. Đầu tiên, access token sẽ được lấy từ header Authorization, sau đó được đưa vào blacklist để ngăn truy cập tiếp theo.
- Nếu phía client gửi thêm thông tin về refresh token cụ thể, hàm sẽ thu hồi chính xác token đó. Ngoài ra, nếu người dùng chọn “đăng xuất tất cả”, hàm sẽ gọi revoke\_all\_user\_tokens để thu hồi toàn bộ các refresh token đang tồn tại của người đó.
- Trong trường hợp không có refresh token cụ thể và không bật cờ “logout all”, mặc định hệ thống vẫn sẽ thu hồi toàn bộ phiên để đảm bảo an toàn. Điều này phù hợp trong các tình huống như mất thiết bị hoặc nghi ngờ bị đánh cắp token.
- Ngoài Access Token được lưu ở localStorage thì refreshToken mặc định được lưu ở HTTP-only cookies khi user đăng nhập thành công, cụ thể như sau:

```

- response.set_cookie(
-     key="refresh_token",
-     value=result["refresh_token"],
-     max_age=max_age,
-     httponly=True, # HTTP-only cookie
-     secure=True,   # HTTPS only request
-     samesite="strict" # CSRF protection
- )

```

- key="refresh\_token": tên của cookie.
- value=result["refresh\_token"]: giá trị là refresh token vừa sinh ra.
- max\_age=max\_age: thời gian sống của cookie (7 hoặc 30 ngày).
- httponly=True: bảo vệ khỏi JavaScript, ngăn XSS tấn công đánh cắp cookie.
- secure=True: cookie chỉ gửi khi request qua HTTPS.
- samesite="strict": ngăn chặn cookie được gửi từ các website khác — bảo vệ chống CSRF (Cross-Site Request Forgery).

- Test Authentication trên Postman

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows collections like AIC2024, Bookstore api, Chess, LearnTwinChain, MediCareAI, and NT219. Under NT219, there is a folder named "Auth" which contains "POST Register" and "POST Login".
- Request URL:** https://api.voux-platform.shop/api/auth/register
- Method:** POST
- Body (JSON):**

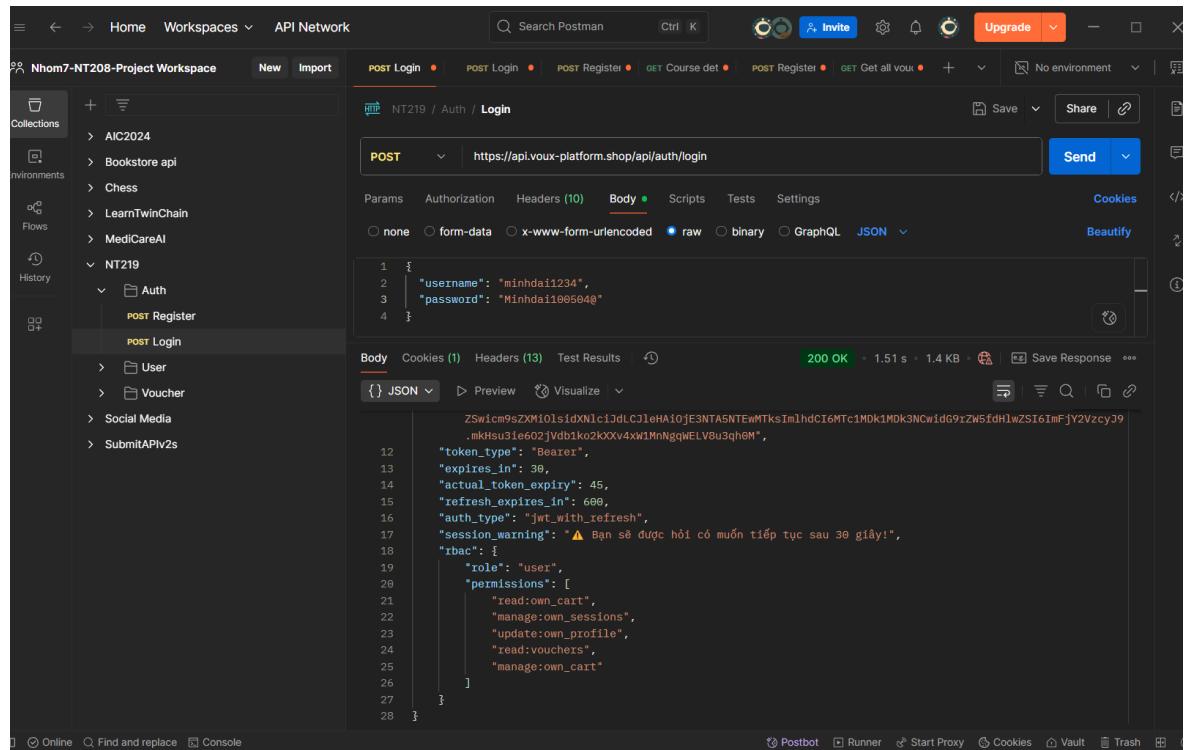
```

1 {
2   "username": "minhdai1234",
3   "email": "ttranduongminhdai@gmail.com",
4   "password": "Minhdai100504@"
5 }
```
- Response Status:** 200 OK
- Response Body (JSON):**

```

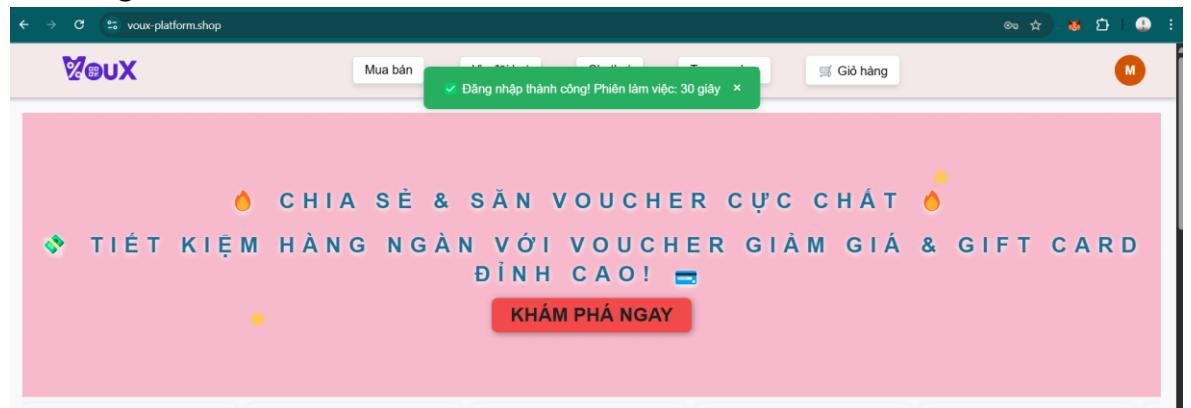
1 {
2   "success": true,
3   "message": "Đăng ký tài khoản thành công! 🎉"
4 }
```

Hình 4. Register thành công trên Postman



Hình 5. Login thành công trên Postman

- Test Login trên Web UI:



Hình 6. Đăng nhập thành công trên Web UI

### 3.2.2. Authorization với Role-based Access Control

- Hệ thống sử dụng hai Enum chính để xây dựng nền tảng cho RBAC:
- Permission: định nghĩa tất cả các quyền mà một người dùng có thể có trong hệ thống. Các quyền được nhóm theo chức năng như:

- Người dùng: READ\_USERS, CREATE\_USERS, UPDATE\_USERS, ...
- Voucher: CREATE\_VOUCHERS, UPDATE\_OWN\_VOUCHERS, ...
- Giỏ hàng: READ\_OWN\_CART, MANAGE\_OWN\_CART
- Quản trị: ADMIN\_ACCESS, MANAGE\_SYSTEM
- Phiên làm việc: MANAGE\_OWN\_SESSIONS, MANAGE\_ALL\_SESSIONS

```

- class Permission(str, Enum):
-     """System permissions"""
-     # User permissions
-     READ_USERS = "read:users"
-     CREATE_USERS = "create:users"
-     UPDATE_USERS = "update:users"
-     DELETE_USERS = "delete:users"
-     UPDATE_OWN_PROFILE = "update:own_profile"
-
-     # Voucher permissions
-     READ_VOUCHERS = "read:vouchers"
-     CREATE_VOUCHERS = "create:vouchers"
-     UPDATE_VOUCHERS = "update:vouchers"
-     DELETE_VOUCHERS = "delete:vouchers"
-     UPDATE_OWN_VOUCHERS = "update:own_vouchers"
-     DELETE_OWN_VOUCHERS = "delete:own_vouchers"
-
-     # Cart permissions
-     READ_OWN_CART = "read:own_cart"
-     MANAGE_OWN_CART = "manage:own_cart"
-
-     # Admin permissions
-     ADMIN_ACCESS = "admin:access"
-     MANAGE_SYSTEM = "manage:system"
-     VIEW_ANALYTICS = "view:analytics"
-
-     # Session management
-     MANAGE_OWN_SESSIONS = "manage:own_sessions"
-     MANAGE_ALL_SESSIONS = "manage:all_sessions"
-
```

- Việc sử dụng cú pháp ACTION\_RESOURCE giúp dễ đọc và mở rộng, đồng thời phân biệt rõ giữa quyền thao tác **tài nguyên của chính mình (OWN)** và **tất cả tài nguyên (ALL)**.
- Role: xác định các vai trò người dùng trong hệ thống. Ví dụ:

- GUEST: người dùng chưa đăng nhập.
- USER: người dùng đã đăng ký.
- VOUCHER\_CREATOR, MODERATOR, ADMIN, SUPER\_ADMIN: tăng dần theo mức độ quyền.

```
- class Role(Enum):
-     GUEST = "guest"
-     USER = "user"
-     VOUCHER_CREATOR = "voucher_creator"
-     MODERATOR = "moderator"
-     ADMIN = "admin"
-     SUPER_ADMIN = "super_admin"
```

- Cấu trúc này hỗ trợ việc phân tầng quyền linh hoạt, cho phép hệ thống mở rộng dễ dàng khi thêm vai trò mới.
- Hệ thống ánh xạ mỗi vai trò với một tập hợp quyền cụ thể trong ROLE\_PERMISSIONS. Ví dụ, GUEST chỉ có quyền đọc voucher (READ\_VOUCHERS), còn USER có thêm quyền quản lý giỏ hàng và cập nhật hồ sơ cá nhân. Vai trò VOUCHER\_CREATOR kế thừa quyền của USER và có thêm khả năng tạo, cập nhật và xóa voucher của chính mình. ADMIN được cấp toàn bộ quyền quản trị cao cấp như MANAGE\_SYSTEM, VIEW\_ANALYTICS.

```
- ROLE_PERMISSIONS = {
-     Role.GUEST: {
-         Permission.READ_VOUCHERS,
-     },
-     Role.USER: {
-         # Basic voucher permissions
-         Permission.READ_VOUCHERS,
-
-         # Cart permissions - THÊM VÀO ĐÂY
-         Permission.READ_OWN_CART,
-         Permission.MANAGE_OWN_CART,
-
-         # Profile permissions
-         Permission.UPDATE_OWN_PROFILE,
-
-         # Session permissions
-         Permission.MANAGE_OWN_SESSIONS,
-     },
-     Role.VOUCHER_CREATOR: {
-         # Inherit user permissions
-         Permission.READ_VOUCHERS,
```

```
-    Permission.READ_OWN_CART,
-    Permission.MANAGE_OWN_CART,
-    Permission.UPDATE_OWN_PROFILE,
-    Permission.MANAGE_OWN_SESSIONS,
-
-    # Voucher creation permissions
-    Permission.CREATE_VOUCHERS,
-    Permission.UPDATE_OWN_VOUCHERS,
-    Permission.DELETE_OWN_VOUCHERS,
-
- },
- Role.MODERATOR: {
-     # Inherit voucher creator permissions
-     Permission.READ_VOUCHERS,
-     Permission.CREATE_VOUCHERS,
-     Permission.UPDATE_OWN_VOUCHERS,
-     Permission.DELETE_OWN_VOUCHERS,
-     Permission.READ_OWN_CART,
-     Permission.MANAGE_OWN_CART,
-     Permission.UPDATE_OWN_PROFILE,
-     Permission.MANAGE_OWN_SESSIONS,
-
-     # Moderation permissions
-     Permission.UPDATE_VOUCHERS,
-     Permission.DELETE_VOUCHERS,
-     Permission.VIEW_ANALYTICS,
-
- },
- Role.ADMIN: {
-     # All permissions except super admin ones
-     Permission.READ_USERS,
-     Permission.CREATE_USERS,
-     Permission.UPDATE_USERS,
-     Permission.DELETE_USERS,
-     Permission.UPDATE_OWN_PROFILE,
-
-     Permission.READ_VOUCHERS,
-     Permission.CREATE_VOUCHERS,
-     Permission.UPDATE_VOUCHERS,
-     Permission.DELETE_VOUCHERS,
-     Permission.UPDATE_OWN_VOUCHERS,
-     Permission.DELETE_OWN_VOUCHERS,
-
-     Permission.READ_OWN_CART,
-     Permission.MANAGE_OWN_CART,
-
-     Permission.ADMIN_ACCESS,
-     Permission.VIEW_ANALYTICS,
```

```

-         Permission.MANAGE_OWN_SESSIONS,
-         Permission.MANAGE_ALL_SESSIONS,
-     },
-     Role.SUPER_ADMIN: {
-         # All permissions
-         *[perm for perm in Permission]
-     }
- }

```

- Lớp RBACManager trong rbac.py đóng vai trò trung tâm trong việc xử lý và kiểm tra quyền truy cập:
  - o get\_user\_role(user\_data): trích xuất vai trò từ thông tin người dùng. Nếu giá trị không hợp lệ, hệ thống mặc định là GUEST, giúp giảm lỗi và tăng tính an toàn.
  - o get\_user\_permissions(user\_data): dựa trên role, truy xuất tất cả các quyền liên quan và trả về dưới dạng Set, giúp tra cứu nhanh và tránh trùng lặp.
  - o has\_permission(user\_data, required\_permission): kiểm tra xem người dùng có quyền cụ thể hay không, và trả về kết quả True/False.

```

- class RBACManager:
-     """Role-Based Access Control Manager"""
-
-     @staticmethod
-     def get_user_role(user: Dict) -> Role:
-         """Determine user role based on user data"""
-         if user.get("admin", False):
-             return Role.ADMIN
-
-         # Check for specific role assignments
-         user_roles = user.get("roles", ["user"])
-
-         if "super_admin" in user_roles:
-             return Role.SUPER_ADMIN
-         elif "admin" in user_roles:
-             return Role.ADMIN
-         elif "moderator" in user_roles:
-             return Role.MODERATOR
-         elif "voucher_creator" in user_roles:
-             return Role.VOUCHER_CREATOR
-         else:
-             return Role.USER
-
-     @staticmethod

```

```

-     def get_user_permissions(user: Dict) -> Set[Permission]:
-         """Get all permissions for a user"""
-         role = RBACManager.get_user_role(user)
-         return ROLE_PERMISSIONS.get(role, set())
-
-     @staticmethod
-     def has_permission(user: Dict, permission: Permission) -> bool:
-         """Check if user has specific permission"""
-         user_permissions = RBACManager.get_user_permissions(user)
-         return permission in user_permissions

```

- Ta sẽ sử dụng một decorator để bảo vệ các route yêu cầu quyền cụ thể.

Decorator này sẽ:

- o Lấy thông tin user từ context request.
  - o Kiểm tra xem user có quyền được yêu cầu hay không.
  - o Nếu không có, trả về lỗi 403 Permission Denied với thông báo rõ ràng.
- Đây là cách tiếp cận hiệu quả để đánh kèm kiểm tra bảo mật vào bất kỳ endpoint nào mà không lặp lại logic kiểm tra quyền nhiều lần.

```

- # Permission decorators
- def require_permission(permission: Permission):
-     """Decorator to require specific permission"""
-     def decorator(func: Callable) -> Callable:
-         @async def wrapper(*args, **kwargs):
-             # Extract current_user from kwargs
-             current_user = kwargs.get("current_user")
-             if not current_user:
-                 raise HTTPException(
-                     status_code=status.HTTP_401_UNAUTHORIZED,
-                     detail={
-                         "success": False,
-                         "message": "Authentication required"
-                     }
-                 )
-
-             if not RBACManager.has_permission(current_user, permission):
-                 raise HTTPException(
-                     status_code=status.HTTP_403_FORBIDDEN,
-                     detail={
-                         "success": False,
-                         "message": "Insufficient permissions",
-                         "required_permission": permission.value
-                     }
-                 )
-
-             return await func(*args, **kwargs)

```

```
-         return wrapper
-     return decorator

-
-     def require_any_permission(permissions: List[Permission]):
-         """Decorator to require any of the specified permissions"""
-         def decorator(func: Callable) -> Callable:
-             async def wrapper(*args, **kwargs):
-                 current_user = kwargs.get("current_user")
-                 if not current_user:
-                     raise HTTPException(
-                         status_code=status.HTTP_401_UNAUTHORIZED,
-                         detail={
-                             "success": False,
-                             "message": "Authentication required"
-                         }
-                     )
- 
-                     if not RBACManager.has_any_permission(current_user, permissions):
-                         raise HTTPException(
-                             status_code=status.HTTP_403_FORBIDDEN,
-                             detail={
-                                 "success": False,
-                                 "message": "Insufficient permissions",
-                                 "required_permissions": [p.value for p in permissions]
-                             }
-                         )
- 
-                     return await func(*args, **kwargs)

-
-         return wrapper
-     return decorator

-
-     def require_ownership_or_permission(required_permission: Permission, ownership_permission: Permission):
-         """Decorator to require ownership or specific permission"""
-         def decorator(func: Callable) -> Callable:
-             async def wrapper(*args, **kwargs):
-                 current_user = kwargs.get("current_user")
-                 if not current_user:
-                     raise HTTPException(
-                         status_code=status.HTTP_401_UNAUTHORIZED,
-                         detail={
-                             "success": False,
```

```

-                         "message": "Authentication required"
-
-                     }
-
-                 )
-
-             # Extract resource owner ID from kwargs or args
-             resource_owner_id = kwargs.get("user_id") or
-             kwargs.get("resource_owner_id")
-
-             if not RBACManager.can_access_resource(
-                 current_user,
-                 resource_owner_id,
-                 required_permission,
-                 ownership_permission
-             ):
-                 raise HTTPException(
-                     status_code=status.HTTP_403_FORBIDDEN,
-                     detail={
-                         "success": False,
-                         "message": "Access denied - insufficient
permissions or not resource owner"
-                     }
-                 )
-
-             return await func(*args, **kwargs)
-
-         return wrapper
-     return decorator

```

- Cụ thể như decorator @require\_permission(permission: Permission)
- Decorator này dùng để yêu cầu người dùng phải có một quyền cụ thể để được phép truy cập vào một endpoint.
- Cơ chế:
  - o Lấy current\_user từ kwargs (thường inject sẵn từ FastAPI Depends).
  - o Nếu không có user (chưa đăng nhập) → trả lỗi 401 Unauthorized.
  - o Nếu user không có quyền permission được chỉ định → trả lỗi 403 Forbidden.
  - o Nếu hợp lệ → thực thi hàm gốc
- Ngoài decorator, ta implement thêm dependency injection theo chuẩn FastAPI thông qua require\_permission\_dep, giúp: Dễ dàng gắn vào danh sách dependencies của route. Kết hợp với get\_current\_user để đảm bảo người dùng đã đăng nhập và có quyền.

```

- # FastAPI dependency functions
- def require_admin():
-     """FastAPI dependency to require admin role"""

```

```
-     def check_admin(current_user: Dict = Depends(lambda: None)):
-         if not current_user:
-             raise HTTPException(
-                 status_code=status.HTTP_401_UNAUTHORIZED,
-                 detail={"success": False, "message": "Authentication
required"})
-         )
-
-         if not RBACManager.has_permission(current_user,
Permission.ADMIN_ACCESS):
-             raise HTTPException(
-                 status_code=status.HTTP_403_FORBIDDEN,
-                 detail={"success": False, "message": "Admin access
required"})
-         )
-
-     return current_user
-
-     return check_admin
-
- def require_role(required_role: Role):
-     """FastAPI dependency to require specific role"""
-     def check_role(current_user: Dict = Depends(lambda: None)):
-         if not current_user:
-             raise HTTPException(
-                 status_code=status.HTTP_401_UNAUTHORIZED,
-                 detail={"success": False, "message": "Authentication
required"})
-         )
-
-         user_role = RBACManager.get_user_role(current_user)
-         if user_role != required_role and user_role not in [Role.ADMIN,
Role.SUPER_ADMIN]:
-             raise HTTPException(
-                 status_code=status.HTTP_403_FORBIDDEN,
-                 detail={
-                     "success": False,
-                     "message": f"Role '{required_role.value}' required",
-                     "user_role": user_role.value
-                 }
-         )
-
-     return current_user
-
-     return check_role
```

- Cuối cùng, ta áp dụng các decorator hoặc dùng dependency injection để thêm vào các route api, ví dụ trong route của users ( user\_routes.py):

- ```
@router.get("/profile/me",
dependencies=[Depends(require_permission_dep(Permission.UPDATE_OWN_PROFILE))])
```
- Route này yêu cầu người dùng phải có quyền UPDATE\_OWN\_PROFILE để truy xuất thông tin cá nhân. Middleware sẽ tự động từ chối nếu quyền không hợp lệ.
- Hay ví dụ áp dụng vào route update profile của 1 user:

```

- 
- # Update user profile (admin or owner)
- @router.put("/{user_id}",      dependencies=[Depends(get_current_user),
- Dependes(normal_rate_limit)])
- async def update_user_profile(
-     user_id: str,
-     update_data: UserProfileUpdate,
-     current_user: dict = Depends(get_current_user)
- ):
-     """Update user profile (requires UPDATE_USERS permission or
- ownership + UPDATE_OWN_PROFILE)"""
-     # Check permissions: admin can update any profile, users can only
-     # update their own
-     can_update_any = RBACManager.has_permission(current_user,
- Permission.UPDATE_USERS)
-     can_update_own = (RBACManager.has_permission(current_user,
- Permission.UPDATE_OWN_PROFILE)
-                     and current_user.get("id") == user_id)
- 
-     if not (can_update_any or can_update_own):
-         raise HTTPException(
-             status_code=403,
-             detail={
-                 "success": False,
-                 "message": "Insufficient permissions to update this
- profile",
-                 "required": "UPDATE_USERS permission or ownership +
- UPDATE_OWN_PROFILE"
-             }
-         )
- 
-     return await user_controller.update_user_profile(user_id,
- update_data, current_user)

```

- Ở đây, logic kiểm tra quyền được viết thủ công:

- Nếu user có quyền UPDATE\_USERS → được phép cập nhật bất kỳ hồ sơ nào.
  - Nếu user chỉ có quyền UPDATE\_OWN\_PROFILE, thì chỉ được cập nhật hồ sơ của chính họ (user\_id == current\_user["id"]).
  - Nếu không có một trong hai quyền trên, hệ thống trả về lỗi 403 với thông báo chi tiết về quyền yêu cầu.
- Test RBAC qua Postman:
- Trước tiên ta cần đăng nhập 1 tài khoản chỉ có role User bình thường để lấy access token, trong role này chỉ có các quyền như UPDATE\_OWN\_PROFILE hay MANAGE\_OWN\_CART chứ không có quyền READ\_USERS để xem toàn bộ user:

The screenshot shows the Postman interface with the following details:

- Project Workspace:** NT219 / Auth
- Method:** POST
- URL:** https://api.voux-platform.shop/api/auth/login
- Body (raw JSON):**

```

1  {
2   "username": "minhdai1234",
3   "password": "Minhdai100504@"
4 }
```
- Response:**
  - Status: 200 OK
  - Time: 3.05 s
  - Size: 1.4 KB
  - Content: {"success": true, "message": "Đăng nhập thành công! (Phiên làm việc: 30 giây)", "user": {"id": "685d64191a15e577896eb127", "username": "minhdai1234", "admin": false, "avatar\_url": "/default-avatar.png", "theme": "light"}, "access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc2VxZ2lkIjoiNjg1ZDY0MTkxYTEzTU3Nzg5NmV1MTI3IiwidXNlcm5hbWUiOjTaw5oZGFpMTIzNCIsImFkbWluIjpmYWxzZSwicm9sZXMiolsidXNlcidldC3leHAiOjE3NTA5NTM2OkMsImlhCI6MTc1MDk1MzYwNCwidG9rZW5fdHlwZSI6ImFjY2VzcyJ9.e0JAS7N70jYc1kC18C16gwBw5HofJ7q0baHBHX8GHjc", "token\_type": "Bearer", "expires\_in": 30, "actual\_token\_expiration": 45, "refresh\_expires\_in": 600}, "refresh\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc2VxZ2lkIjoiNjg1ZDY0MTkxYTEzTU3Nzg5NmV1MTI3IiwidXNlcm5hbWUiOjTaw5oZGFpMTIzNCIsImFkbWluIjpmYWxzZSwicm9sZXMiolsidXNlcidldC3leHAiOjE3NTA5NTM2OkMsImlhCI6MTc1MDk1MzYwNCwidG9rZW5fdHlwZSI6ImFjY2VzcyJ9.e0JAS7N70jYc1kC18C16gwBw5HofJ7q0baHBHX8GHjc", "refresh\_expires\_in": 600}

Hình 7. Đăng nhập để lấy Access Token

- Gắn access token và Authorization header và test với 1 route như getAllUser để lấy về một user, vì mỗi tài khoản khi register qua endpoint /api/register sẽ được gán mặc định Role là User nên kết quả khi ta request để lấy sẽ bị unauthorized:

```

    ] _id: ObjectId('685d64191a15e577896eb127')
    username : "minhdai1234"
    email : "tranduongminhdai@gmail.com"
    password : "$2b$12$KB902afiAlkzKwfCUooTn0kTAjc00ps5tij8a4W.WCBM27G9XvUzm"
    created_at : 2025-06-26T15:15:37.249+00:00
    updated_at : 2025-06-26T15:15:37.249+00:00
    last_login : 2025-06-26T17:17:58.258+00:00
    login_count : 8
    rbac_role : "USER"

```

Hình 8. User minhdai1234 với role USER trên mongodb

- Kết quả bị báo status 403 và log báo không đủ quyền:

The screenshot shows the Postman interface with the following details:

- Project:** Nhom7-NT208-Project Workspace
- Request:** GET /User
- URL:** http://localhost:8060/api/users/
- Authorization:** Bearer Token (Token field is empty)
- Status:** 403 Forbidden
- Body:** JSON response (shown below)

```

1  {
2     "detail": {
3         "success": false,
4         "message": "Insufficient permissions",
5         "required_permission": "read:users",
6         "user_permissions": [
7             "read:own_cart",
8             "manage:own_sessions",
9             "update:own_profile",
10            "read:vouchers",
11            "manage:own_cart"
12        ]
13    }
14  }

```

Hình 9. Kết quả khi test với account có role là USER

- Nhưng khi test với role là SUPER\_ADMIN có quyền READ\_USERS:

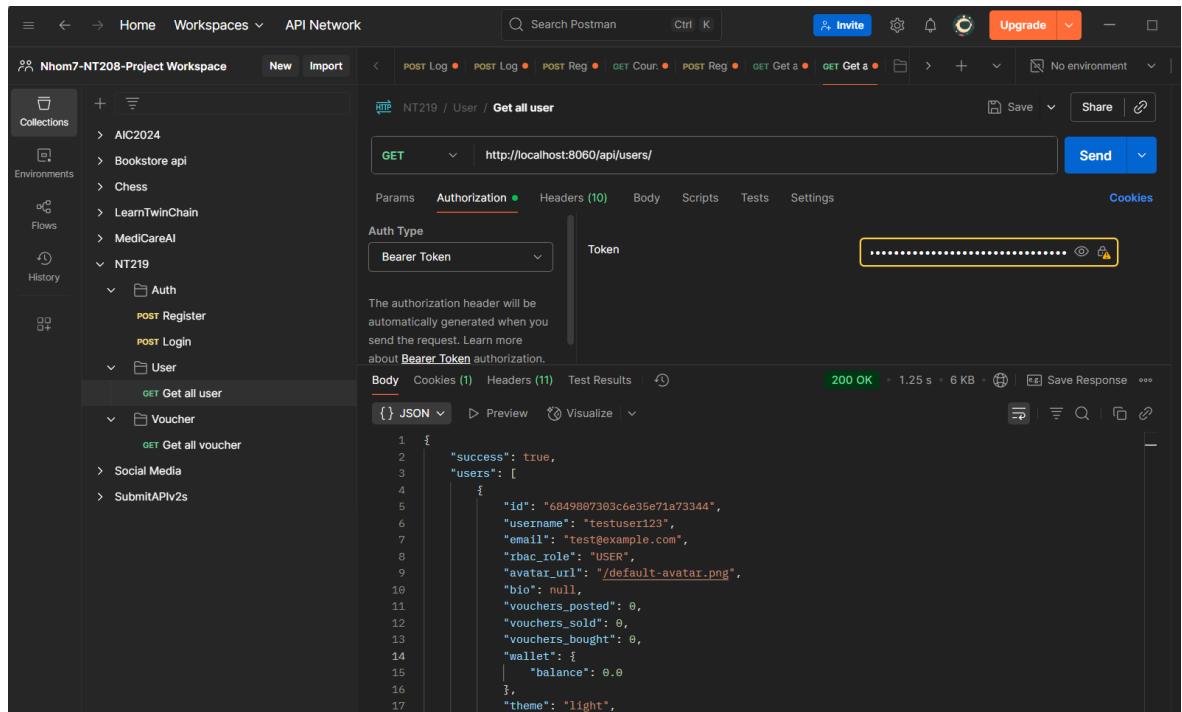
```

    ] _id: ObjectId('685d866375e1c75c32d51737')
    username : "superadmin"
    email : "superadmin@voux.com"
    password : "$2b$12$yKIhtQp8DVZVEyfXjg7UbOJPQHExXTaaovoCp8jnbe1XAi0tTpE2i"
    rbac_role : "SUPER_ADMIN"
    created_at : 2025-06-26T17:41:56.014+00:00
    updated_at : 2025-06-26T17:41:56.014+00:00

```

Hình 10. User superadmin với role là SUPER\_ADMIN trên mongodb

- Kết quả get thành công toàn bộ user:



Hình 11. Kết quả get toàn bộ user với account SUPER\_ADMIN

### 3.3. Quản lý phiên làm việc (Session Management)

- Ta có luồng hoạt động của session như sau login → hết hạn access token → hỏi user xem có gia hạn session không -> nếu có, dùng refresh token để làm mới access token → logout nếu không phản hồi.
  - Khi hết thời gian (ví dụ 30 giây của access token), SessionManager gọi showContinueModal() để hiện modal hỏi người dùng có muốn tiếp tục hay không. Nếu user không phản hồi trong vòng 10 giây, sẽ tự động logout.

```
- this.sessionExpiryTimer = setTimeout(() => {
-     console.log("⌚ Timer triggered - showing continue modal");
-     this.showContinueModal();
- }, this.sessionDuration * 1000);

-
- this.onSessionWarning({
-     timeLeft: 10,
-     message: "Phiên đăng nhập đã hết thời gian. Bạn có muốn tiếp tục không?",
-     isSessionExpired: true,
- });


```

- Nếu người dùng chọn tiếp tục, client gọi `extendSession()`, lúc này gửi request đến API `/auth/refresh`. Server đọc `refresh_token` từ cookie HTTP-only, xác thực và tạo mới `access token`.

```

-   const response = await authRequest({
-     method: "POST",
-     url: "/auth/refresh",
-     data: {}, // refresh_token nằm trong cookie
-   });

```

- Cụ thể ở Backend, trong auth-controller.py, trước tiên khi controller gọi session\_manager.validate\_refresh\_token(...) để kiểm tra refresh token có còn hợp lệ không. Token này có thể được lấy từ body hoặc cookie, tùy thiết kế.
  - o Nếu token không hợp lệ (hết hạn, đã bị thu hồi hoặc giả mạo), server trả về lỗi 401 Unauthorized kèm thông báo rõ ràng.
  - o Sau khi xác thực thành công, controller tạo access\_token mới bằng self.auth\_middleware.generate\_jwt\_token(...). Dữ liệu trong token gồm user\_id, username, admin flag và các role.
  - o Thay vì giữ nguyên refresh\_token cũ, hệ thống sẽ cấp mới một refresh token để tránh reuse (theo chuẩn bảo mật "refresh token rotation").
  - o Ngay sau khi tạo refresh token mới, backend sẽ thu hồi refresh token cũ, đánh dấu là revoked để tránh bị lạm dụng nếu bị lộ.
  - o Kết quả cuối cùng là một response JSON chứa cả access\_token mới, refresh\_token mới, thông tin thời gian hết hạn và cảnh báo session sắp hết.

```

-   async def refresh_token(self, refresh_request: RefreshTokenRequest,
request: Request) -> Dict[str, Any]:
    """Refresh access token using refresh token"""
    try:
        # Validate refresh token
        user_data = await
session_manager.validate_refresh_token(refresh_request.refresh_token)

        if not user_data:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail={
                    "success": False,
                    "message": "Refresh token không hợp lệ hoặc đã
hết hạn!"
                }
    )

        # Generate new access token
        token_data = {
            "user_id": user_data.get("user_id"),
            "username": user_data.get("username"),
            "admin": user_data.get("admin", False),
            "roles": user_data.get("roles", ["user"])
        }
    
```

```
-         new_access_token = self.auth_middleware.generate_jwt_token(
-             token_data,
-             expires_delta=timedelta(seconds=45)
-         )
-
-         # Optionally rotate refresh token (security best practice)
-         new_refresh_token = await
session_manager.create_refresh_token(
    user_data.get("user_id"),
    device_info=request.headers.get("user-agent"),
    ip_address=request.client.host
)
-
# Revoke old refresh token
await
session_manager.revoke_refresh_token(refresh_request.refresh_token)
-
return {
    "success": True,
    "message": "Token đã được refresh! (Phiên mới: 30
giây)",
    "access_token": new_access_token,
    "refresh_token": new_refresh_token,
    "token_type": "Bearer",
    "expires_in": 30, # Client sees 30 seconds, but token
valid for 45s
    "actual_token_expiry": 45, # Actual backend expiry
    "refresh_expires_in": 600, # 10 minutes (600 seconds)
    "session_warning": "⚠ Bạn sẽ được hỏi có muốn tiếp
tục sau 30 giây!"
}
-
except HTTPException:
    raise
except Exception as error:
    logger.error(f"Token refresh error: {error}")
    raise HTTPException(
        status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
        detail={
            "success": False,
            "message": "Lỗi khi refresh token!",
            "error": str(error)
        }
)
```

- Nếu người dùng không phản hồi trong 10 giây, client sẽ gọi handleSessionExpired(). Hàm này xóa access token trong localStorage, gọi /auth/logout để server thu hồi refresh\_token, rồi chuyển hướng về trang login.

```

-    await authRequest({
-      method: "POST",
-      url: "/auth/logout",
-      data: { logout_all: false },
-    });
-    localStorage.removeItem("accessToken");
-    window.location.href = "/login";

```

- Ở phía Backend, controller sẽ xử lý lần lượt theo logic bảo mật: access token hiện tại sẽ được đưa vào blacklist để vô hiệu hóa ngay lập tức.
  - o Nếu có refresh token đi kèm, hệ thống sẽ thu hồi token đó bằng cách đánh dấu là isRevoked. Trong trường hợp người dùng yêu cầu logout\_all = true hoặc không truyền refresh token cụ thể, hệ thống sẽ tiến hành thu hồi toàn bộ các phiên hoạt động của người dùng để đảm bảo an toàn.
  - o Sau khi hoàn tất các thao tác backend, phía frontend sẽ tiếp tục xóa access token khỏi localStorage, đảm bảo token không còn tồn tại ở phía client.
  - o Cuối cùng, người dùng sẽ được chuyển hướng về trang đăng nhập (/login).

```

- async def logout(self, request: Request, logout_data: Optional[LogoutRequest] = None, current_user: Optional[Dict[str, Any]] = None) -> Dict[str, Any]:
-     """Enhanced logout with token revocation"""
-     try:
-         user_id = current_user.get("id") if current_user else "unknown"
-         logger.info(f"Logout attempt for user: {user_id}")
-
-         # Get access token from request
-         auth_header = request.headers.get("authorization")
-         if auth_header and auth_header.startswith("Bearer "):
-             access_token = auth_header.split(" ")[1]
-             session_manager.blacklist_access_token(access_token)
-             logger.info(f"Blacklisted access token for user: {user_id}")
-
-         logout_all = False
-
```

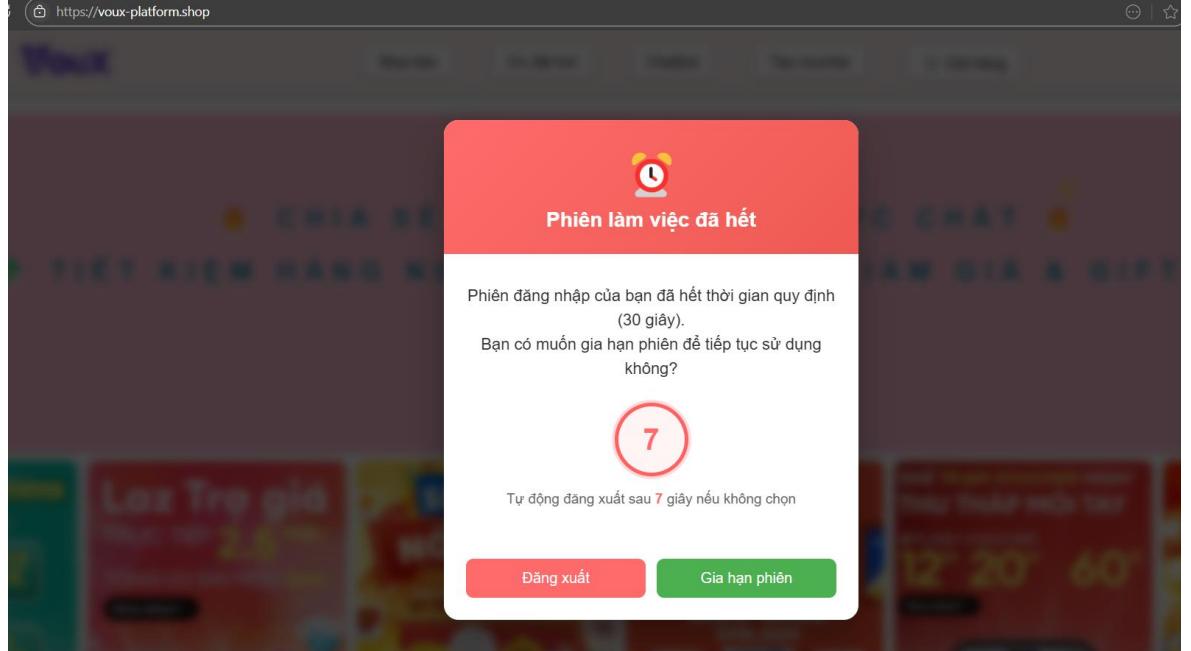
```
-         if logout_data and logout_data.refresh_token:
-             # Revoke specific refresh token
-             logger.info(f"Revoking specific refresh token for user:
{user_id}")
-             try:
-                 await
- session_manager.revoke_refresh_token(logout_data.refresh_token)
-             except Exception as e:
-                 logger.warning(f"Could not revoke specific refresh
token: {e}")
-
-             elif logout_data and logout_data.logout_all and
current_user:
-                 # Revoke all user sessions
-                 logger.info(f"Revoking all sessions for user: {user_id}")
-                 logout_all = True
-                 try:
-                     await
- session_manager.revoke_all_user_tokens(current_user.get("id"))
-                 except Exception as e:
-                     logger.warning(f"Could not revoke all user tokens:
{e}")
-
-             elif current_user:
-                 # Revoke all sessions for current user (default behavior)
-                 logger.info(f"Revoking all sessions for current user:
{user_id}")
-                 logout_all = True
-                 try:
-                     await
- session_manager.revoke_all_user_tokens(current_user.get("id"))
-                 except Exception as e:
-                     logger.warning(f"Could not revoke all user tokens:
{e}")
-
-             logger.info(f"Logout successful for user: {user_id}
(logout_all: {logout_all})")
-
-             return {
-                 "success": True,
-                 "message": "Đăng xuất thành công! 🙌"
-             }
-
-         except Exception as error:
-             logger.error(f"Logout error for user {current_user.get('id')}
if current_user else 'unknown': {error}")
```

```

-           # Return success anyway - logout should always appear to
-           work from user perspective
-           return {
-               "success": True,
-               "message": "Đăng xuất thành công! 🖐"
-           }

```

- Test quản lý phiên và gia hạn phiên:



Hình 12. Thông báo popup gia hạn session trên vox-platform.shop

### 3.4. Mã hóa truyền tải ( TLS Encryption)

Đầu tiên, ta sẽ tạo một private key sử dụng thuật toán ECDSA với đường cong P-256 (prime256v1) — đây là loại khóa được CloudFront hỗ trợ và cung cấp hiệu năng tốt lẫn bảo mật cao. Khóa này sẽ được dùng để ký CSR (Certificate Signing Request) ở bước kế tiếp.

```
openssl ecparam -genkey -name prime256v1 -out vox-platform.shop.key
```

Ta cần tạo public key như sau:

```
f:\NT219-Project\Voux\vox-platform.shop>openssl pkey -in vox-platform.shop.key -pubout -out vox-platform.shop.pubkey.pem
f:\NT219-Project\Voux\vox-platform.shop>cat vox-platform.shop.pubkey.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEEBwv49SRjdg3PPZ0idoLgRF44h0D
/TCIekJZCd31pdhVs4ue21e3BD2lK2gWXHhTAADwEIE5kfMY3jM3jDGaVA==
-----END PUBLIC KEY-----
```

Hình 13. Tạo public key từ private key vox-platform.shop.key

Tiếp theo, ta tạo một file cấu hình CSR chứa các thông tin định danh của tổ chức như quốc gia, tỉnh, tên tổ chức, email và các tên miền (SAN – Subject Alternative Names) cần chứng thực.

```
openssl req -new -sha256 \
-key vousx-platform.shop.key \
-out vousx-platform.shop.csr \
-config vousx-platform.shop.conf
```

- Kết quả:

```
f:\NT219-Project\Voux\voux-platform.shop>openssl req -in vousx-platform.shop.csr -noout -text
Certificate Request:
Data:
    Version: 1 (0x0)
    Subject: C=VN, ST=Ho Chi Minh, L=Ho Chi Minh City, O=Voux Platform, OU=IT Department, CN=voux-platform.shop, emai
ilAddress=admin@voux-platform.shop
    Subject Public Key Info:
        Public Key Algorithm: id-ecPublicKey
            Public-Key: (256 bit)
            pub:
                04:10:1c:2f:e3:d4:91:8d:d8:37:3c:f6:4e:89:da:
                0b:81:11:78:e2:1d:03:fd:30:88:7a:42:59:09:dd:
                f5:a5:d8:55:b3:8b:9e:db:57:b7:04:3d:a5:2b:68:
                16:5c:78:53:00:00:f0:10:81:39:91:f3:18:de:33:
                37:8c:31:9a:54
            ASN1 OID: prime256v1
            NIST CURVE: P-256
    Attributes:
        Requested Extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:voux-platform.shop, DNS:*.voux-platform.shop, DNS:www.voux-platform.shop, DNS:api.voux-platform.
shop, DNS:app.voux-platform.shop
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
        Signature Algorithm: ecdsa-with-SHA256
```

Hình 14. Thông tin chi tiết vousx-platform.shop.csr

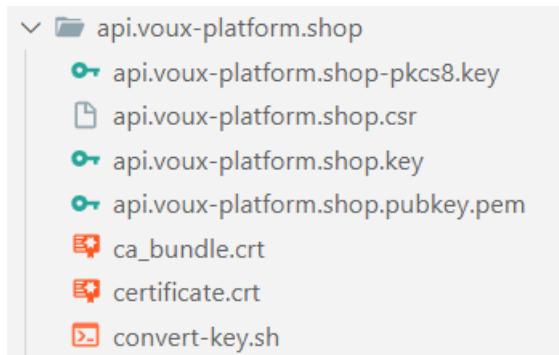
- Làm tương tự với api.voux-platform.shop, ta sẽ tiến hành ký với ZeroSSL như hình dưới:

The screenshot shows the ZeroSSL dashboard with the URL [app.zerossl.com/certificates](https://app.zerossl.com/certificates). The main area displays a table of certificates. The table has columns: TYPE, DOMAINS, STATUS, PROTECT, and EXPIRES. Two entries are listed:

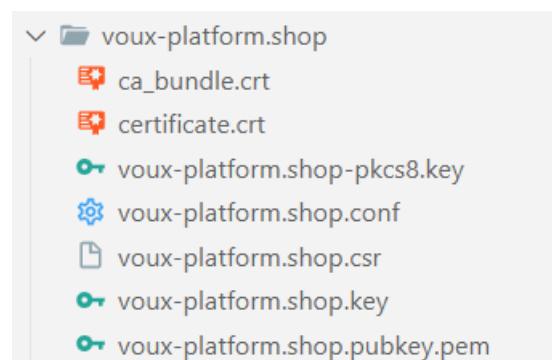
| TYPE       | DOMAINS                | STATUS | PROTECT     | EXPIRES      |
|------------|------------------------|--------|-------------|--------------|
| 90-Day SSL | api.voux-platform.shop | Issued | Try Feature | Sep 11, 2025 |
| 90-Day SSL | voux-platform.shop     | Issued | Try Feature | Sep 10, 2025 |

Hình 15. Kết quả ký 2 certificate trên ZeroSSL

- Ta được folder như sau:



*Hình 16. Folder của domain api.voux-platform.shop*



*Hình 17. Folder của domain vox-platform.shop*

- Vì AWS Certificate Manager cần PKCS#8 format nên ta sẽ convert với lệnh sau:

```
openssl pkcs8 -topk8 -nocrypt -in vox-platform.shop.key -out vox-
platform.shop-pkcs8.key
```

- Đẩy 2 certificate lên AWS ACM với lệnh:

```
- aws acm import-certificate \
-   --certificate file://certificate.crt \
-   --private-key file://vox-platform.shop-pkcs8.key \
-   --certificate-chain file://ca_bundle.crt \
-   --region us-west-1
```

- Kết quả:

Hình 18. Certificate lưu trên AWS Certificate Manager

- Private key sẽ được mã hóa với AWS KMS và lưu trên AWS Secret Manager, bằng cách trước tiên tạo custom key để AWS KMS mã hóa, cụ thể là AES-256-GCM (SYMMETRIC\_DEFAULT), như hình dưới:

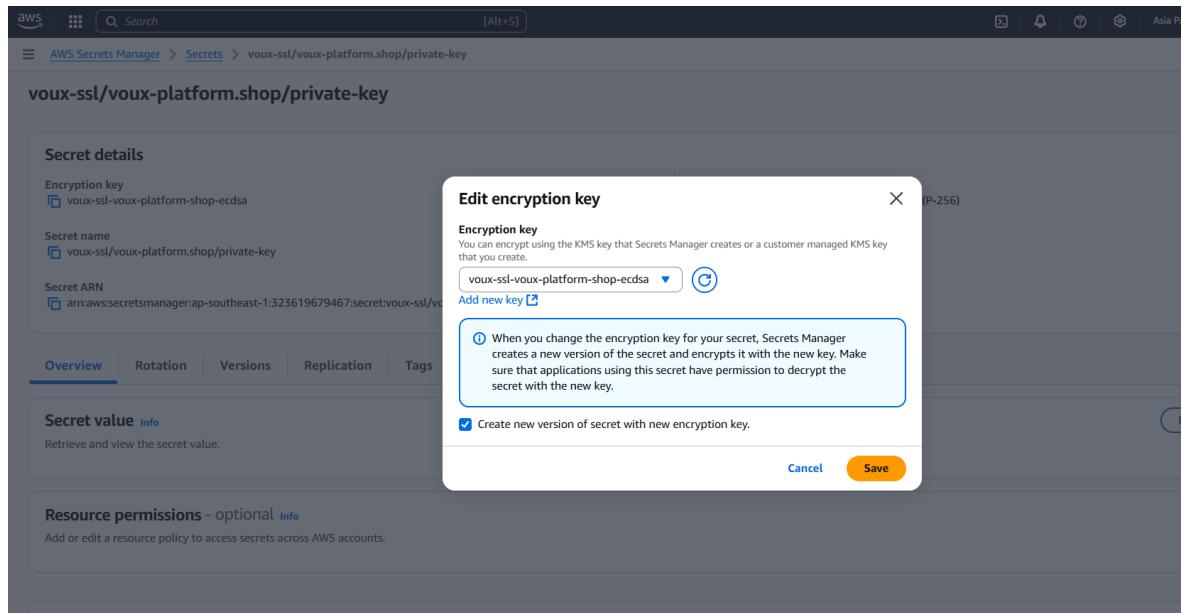
Hình 19. KMS Key tạo trên AWS KMS

- Tiếp theo ta cần tạo secret để lưu private key và dùng KMS key để mã hóa private key lưu trên Secret Manager:

```
root@MinhDaiLaptop:/mnt/f/NT219-Project/Voux/voux-platform.shop# aws secretsmanager create-secret --name vox-ssl/voux-platform.shop/private-key --description "ECDSA private key for vox-platform.shop (P-256)" --secret-string file://voux-platform.shop.key --kms-key-id alias/vox-ssl-voux-platform-shop-ecdsa
{
    "ARN": "arn:aws:secretsmanager:ap-southeast-1:323619679467:secret:vox-ssl/voux-platform.shop/private-key-NSiWpJ",
    "Name": "vox-ssl/voux-platform.shop/private-key",
    "VersionId": "c80ce6c4-4000-433e-a361-6e894abe21a5"
```

Hình 20. Kết quả tạo Secret trên Secret Manager

- Có thể thấy private key đã được lưu trên AWS Secret Manager với custom key mã hóa từ AWS KMS:

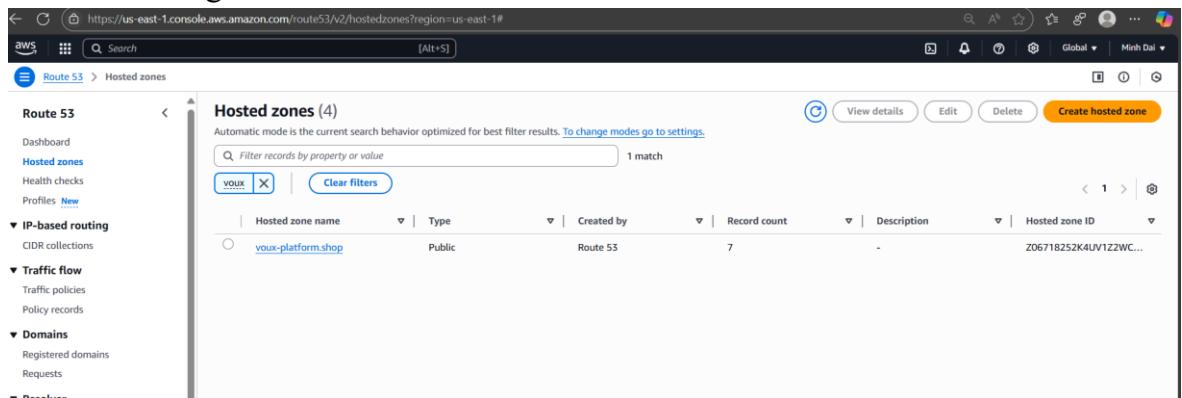


Hình 21. Lưu thành công key trên Secret Manager

### 3.5. Triển khai thử nghiệm

#### 3.5.1. Triển khai Frontend

- Đầu tiên, để route traffic từ phía người dùng ngoài Internet vào web app, ta cần config AWS Route 53, cụ thể là tạo hosted zone cho tên miền và tạo các record đúng để route traffic:



Hình 22. Kết quả tạo hosted zone với tên miền vox-platform.shop

- Trong bảng cấu hình DNS của tên miền `voux-platform.shop` như hình bên dưới, có tổng cộng 7 bản ghi (records) được thiết lập. Các bản ghi chính bao gồm:
  - o A record với alias trỏ đến CloudFront (`ds1zj23b5kdy97.cloudfront.net`), giúp route lưu lượng từ domain gốc `voux-platform.shop` đến frontend được triển khai qua CloudFront. Đây là bản ghi đóng vai trò chính trong việc phân phối nội dung frontend tĩnh đến người dùng cuối. Record `api.voux-platform.shop` là bản ghi CNAME, trỏ đến một endpoint dạng `dualstack.voux-api-alb-...elb.amazonaws.com`, định tuyến đến Load Balancer cho API Gateway của hệ thống.
  - o Hai bản ghi CNAME cuối cùng là: `_8b8b70f34355..._b7ef6151b0b...` Đây là các bản ghi dùng để xác minh quyền quản lý DNS với dịch vụ bên ngoài, cụ thể là để thay thế quyền quản lý từ Hostinger sang AWS Route 53. Các bản ghi này thường được dùng trong quá trình xác thực tên miền (domain verification) khi cấu hình dịch vụ email hoặc hosting mới.
  - o Ngoài ra, các bản ghi mặc định như NS (Name Server) và SOA (Start of Authority) cũng được cấu hình để định danh hệ thống DNS đang được sử dụng, đảm bảo Route 53 là nhà quản lý chính thức cho tên miền.

The screenshot shows the AWS Route 53 Hosted Zone Details page for the domain 'voux-platform.shop'. The 'Records (7)' tab is selected. The table lists the following records:

| Record name         | Type  | Alias | Value/Route traffic to                                                                               | TTL (s) |
|---------------------|-------|-------|------------------------------------------------------------------------------------------------------|---------|
| voux-platform.s...  | A     | Yes   | d3izj23b5kdy97.cloudfront.n...                                                                       | -       |
| voux-platform.s...  | NS    | No    | ns-1699.awsdns-20.co.uk.<br>ns-139.awsdns-17.com.<br>ns-990.awsdns-59.net.<br>ns-1125.awsdns-12.org. | 17280   |
| voux-platform.s...  | SOA   | No    | ns-1699.awsdns-20.co.uk. a...                                                                        | 900     |
| _64a8e9d3179f...    | CNAME | No    | 082A5CFC6809E35EE5DC0C...                                                                            | 3600    |
| api.voux-platfor... | A     | Yes   | dualstack.voux-api-alb-1734...                                                                       | -       |
| _8b8b7034355...     | CNAME | No    | A9CB1439C13A9194F278D4...                                                                            | 3600    |
| _b7e1fc511b0...     | CNAME | No    | _8c7dcfeefa6adf48a9048d9...                                                                          | 300     |

Hình 23. Chi tiết các Record trong vox-platform.shop

- Traffic sẽ được route qua CloudFront, nơi sẽ cấu hình custom SSL Certificate từ zeroSSL như hình dưới:

The screenshot shows the AWS CloudFront Distribution settings page for the distribution 'EE1TM260898PO'. The 'General' tab is selected. The distribution details are as follows:

- Name:** EE1TM260898PO
- Distribution domain name:** d3izj23b5kdy97.cloudfront.net
- ARN:** arn:aws:cloudfront:323619679467:distribution/EE1TM260898PO
- Last modified:** June 13, 2025 at 4:10:52 AM UTC

**Settings:**

- Description:** -
- Price class:** Use all edge locations (best performance)
- Supported HTTP versions:** HTTP/2, HTTP/1.1, HTTP/1.0
- Standard logging:** Off
- Cookie logging:** Off
- Default root object:** -

Hình 24. Chi tiết setting của Cloudfront

The screenshot shows the AWS CloudFront Origins page for distribution EE1TM26O898PO. The 'Origins' tab is selected. A single origin entry is listed: 'voux-platform-frontend.s3-we' with 'Origin path' set to 'voux-platform-frontend....'. The 'Origin type' is 'S3 static website'. There are buttons for 'Edit', 'Delete', and 'Create origin'.

Hình 25. Origin path của Cloudfront

- Frontend sau khi build bằng lệnh npm run build thì sẽ được đẩy lên s3 để lưu như Hình 26 và được bật S3 Website Endpoint để Cloudfront tham chiếu origin như Hình 27 :

The screenshot shows the AWS S3 Objects page for the 'voux-platform-frontend' bucket. The 'Objects' tab is selected, displaying four objects: '\_redirects' (Folder), 'assets/' (Folder), 'index.html' (html), and 'vite.svg' (svg). The table includes columns for Name, Type, Last modified, Size, and Storage class. A sidebar on the left shows general purpose buckets and storage lens settings.

Hình 26. Các object đã build của Frontend trong S3

The screenshot shows the AWS S3 Static website hosting configuration page. It indicates that the bucket is enabled for static website hosting. The hosting type is 'Bucket hosting'. The bucket website endpoint is listed as <http://voux-platform-frontend.s3-website-us-east-1.amazonaws.com>. A note recommends using AWS Amplify Hosting for static website hosting.

Hình 27. S3 Website Endpoint tính năng chi tiết

- Như đã đề cập ở các phần trước, kiến trúc của nhóm sẽ sử dụng 2 Certificate chính là `voux-platform.shop` và `api.voux-platform.shop` đều đã được Issued và đẩy lên AWS ACM như hình dưới, nhóm không sử dụng Wildcard Certificate vì trả phí trên ZeroSSL:

| Certificate ID                                       | Domain name            | Type     | Status | In use | Rewind eligible |
|------------------------------------------------------|------------------------|----------|--------|--------|-----------------|
| <a href="#">30f54934-202a-40c2-a974-a584bfc3ba6f</a> | api.voux-platform.shop | Imported | Issued | Yes    | Ineligible      |
| <a href="#">a4af6b76-e19a-4e60-855e-278ab8dd6d0c</a> | vox-platform.shop      | Imported | Issued | Yes    | Ineligible      |

Hình 28. Hai Certificate lưu ở AWS Certificate Manager

```

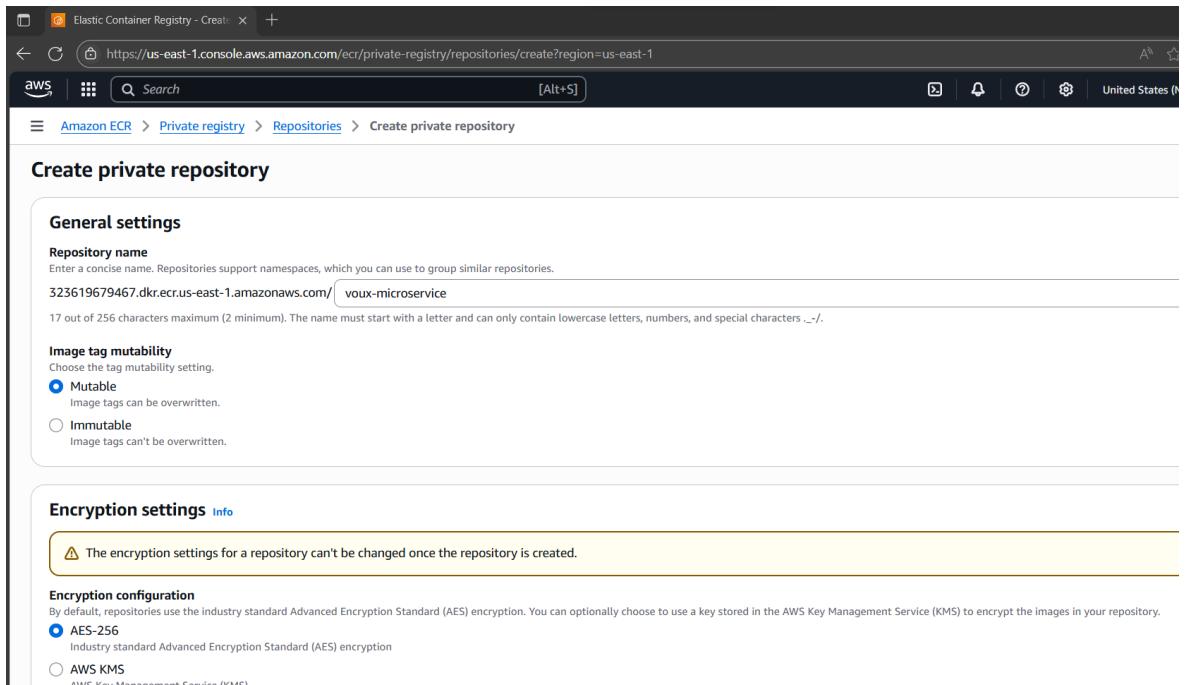
1  {
2    "Version": "2012-10-17",
3    "Id": "voux-ssl-ecdsa-key-policy",
4    "Statement": [
5      {
6        "Sid": "Enable TAM User Permissions"
7      }
8    ]
9  }

```

Hình 29. Tham chiếu private key trên AWS Key Management Store

### 3.5.2. Triển khai Backend Microservices

- Đầu tiên, để chuẩn bị cho quá trình triển khai các dịch vụ dưới dạng container, ta cần tạo một repository trên Amazon ECR (Elastic Container Registry) như Hình 4 để lưu trữ các Docker image tương ứng.



Hình 30. Giao diện tạo repository trong AWS ECR

- Tại đây, ta khai báo tên repository là `voux-microservice` và cấu hình một số tùy chọn bảo mật và tag như sau:
  - o Tag Mutability: được chọn là `Mutable`, cho phép ghi đè image có cùng tag. Tùy chọn này phù hợp trong quá trình phát triển, khi cần cập nhật thường xuyên.
  - o Encryption: sử dụng chuẩn mã hóa AES-256 mặc định của AWS để đảm bảo an toàn cho các image lưu trữ.
- Kết quả ta tạo được repo như Hình 5:

| Private repositories (1) |  |                   |                                                                |                                  |                  |
|--------------------------|--|-------------------|----------------------------------------------------------------|----------------------------------|------------------|
|                          |  | Repository name   | URI                                                            | Created at                       | Tag immutability |
|                          |  | voux-microservice | 323619679467.dkr.ecr.us-east-1.amazonaws.com/voux-microservice | June 12, 2025, 23:00:27 (UTC+07) | Mutable          |

Hình 31. Repository `Voux-microservice` sau khi tạo trên ECR

- Tiếp theo, ta cần đóng gói các services thành docker image và đẩy lên Container Registry để lưu là ECR, ta có script để đẩy lên như sau:

```

-#!/bin/bash

-
- # Build and Push Docker Images to Single ECR Registry
- # Usage: ./build-and-push.sh [ECR_REGISTRY] [AWS_REGION] [IMAGE_TAG]
-
- set -e
-
- # Configuration
- ECR_REGISTRY=${1:-"323619679467.dkr.ecr.us-east-1.amazonaws.com/voux-
microservice"}
- AWS_REGION=${2:-"us-east-1"}
- IMAGE_TAG=${3:-"latest"}
-
- echo "🔧 Building and pushing Docker images to single ECR registry..."
- echo "Registry: $ECR_REGISTRY"
- echo "Region: $AWS_REGION"
- echo "Tag: $IMAGE_TAG"
-
- # Microservices to build
- SERVICES=(
-     "api-gateway"
-     "auth-service"
-     "user-service"
-     "voucher-service"
-     "cart-service"
- )
-
- # Login to ECR
- echo "🔐 Logging in to ECR..."
- aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --password-stdin $ECR_REGISTRY
-
- # Function to build and push a service
- build_and_push_service() {
-     local service=$1
-     local service_dir="../microservice-python"
-     local dockerfile_path="$service_dir/$service/Dockerfile"
-     local local_image_name="voux-$service"
-     local service_tag="$service-$IMAGE_TAG"
-     local full_image_name="$ECR_REGISTRY:$service_tag"
-
-     echo ""
-     echo "📦 Building $service..."
-     -
-     # Check if service directory exists

```

```

-     if [[ ! -d "$service_dir/$service" ]]; then
-         echo "✗ Service directory not found: $service_dir/$service"
-         return 1
-     fi

-
-     # Check if Dockerfile exists
-     if [[ ! -f "$dockerfile_path" ]]; then
-         echo "✗ Dockerfile not found: $dockerfile_path"
-         return 1
-     fi

-
-     # Build Docker image from microservice-python directory with
-     # service-specific dockerfile
-     echo "Building: docker build -f $service/Dockerfile -t
$local_image_name $service_dir"
-     cd $service_dir
-     docker build -f $service/Dockerfile -t $local_image_name .
-     cd - > /dev/null

-
-     # Tag for ECR with service-specific tag
-     echo "Tagging: docker tag $local_image_name $full_image_name"
-     docker tag $local_image_name $full_image_name

-
-     # Push to ECR
-     echo "Pushing: docker push $full_image_name"
-     docker push $full_image_name

-
-     echo "☑ Successfully pushed $full_image_name"
}

-
# Create single ECR repository if it doesn't exist
echo ""

echo "📝 Creating ECR repository..."
REPO_NAME=$(echo $ECR_REGISTRY | sed 's|.*|||')
echo "Creating repository: $REPO_NAME"

aws ecr create-repository \
    --repository-name $REPO_NAME \
    --region $AWS_REGION \
    --image-scanning-configuration scanOnPush=true \
    --encryption-configuration encryptionType=AES256 \
    2>/dev/null || echo "Repository $REPO_NAME already exists"

#
# Build and push all services
echo ""

```

```

- echo "🏗 Building and pushing all services to single registry..."
- for service in "${SERVICES[@]}"; do
    build_and_push_service $service
done

-
echo ""
echo "🎉 All images successfully built and pushed to single registry!"
echo ""
echo "📋 Images pushed to registry: $ECR_REGISTRY"
for service in "${SERVICES[@]}"; do
    echo "  - $ECR_REGISTRY:$service-$IMAGE_TAG"
Done

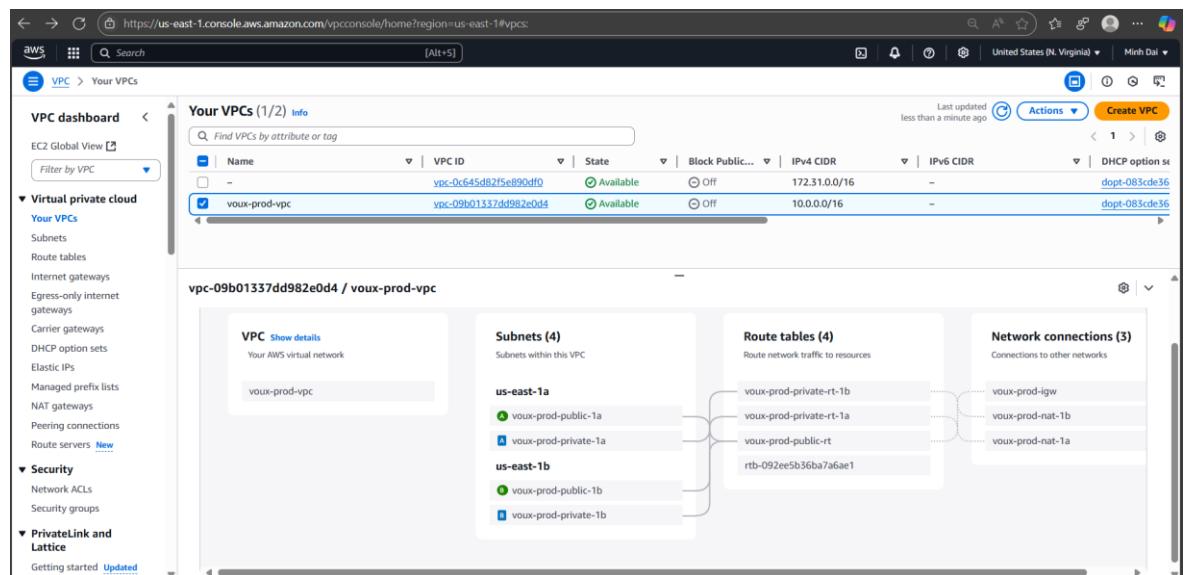
```

- Script này dùng để build và đẩy (push) Docker image của nhiều microservice lên một ECR registry duy nhất trên AWS. Cụ thể:
- Tham số đầu vào gồm:
  - o ECR\_REGISTRY: địa chỉ ECR registry.
  - o AWS\_REGION: vùng AWS.
  - o IMAGE\_TAG: tag của image (mặc định là latest).
- Chức năng chính:
  - o Đăng nhập vào ECR bằng aws ecr get-login-password.
  - o Duyệt qua từng service trong danh sách SERVICES, build Docker image từ thư mục ../microservice-python/[service].
  - o Tag mỗi image theo định dạng: [ECR\_REGISTRY]:[service]-[tag].
  - o Push image lên ECR.
  - o Tự động tạo ECR repo nếu chưa tồn tại.
- Kết quả, ta thấy các image đã được đẩy lên ECR với tag latest:

| Image tag              | Artifact type | Pushed at                        | Size (MB) | Image URI                | Digest                                     | Last recorded pull time          |
|------------------------|---------------|----------------------------------|-----------|--------------------------|--------------------------------------------|----------------------------------|
| cart-service-latest    | Image Index   | June 25, 2025, 13:49:13 (UTC+07) | 83.59     | <a href="#">Copy URI</a> | <a href="#">sha256:19de17c7db0b61...</a>   | June 25, 2025, 13:50:52 (UTC+07) |
| -                      | Image         | June 25, 2025, 13:49:12 (UTC+07) | 83.59     | <a href="#">Copy URI</a> | <a href="#">sha256:af574a199cd675...</a>   | June 25, 2025, 13:49:13 (UTC+07) |
| -                      | Image         | June 25, 2025, 13:49:12 (UTC+07) | 0.00      | <a href="#">Copy URI</a> | <a href="#">sha256:edd9f67fd2ed5eb...</a>  | -                                |
| voucher-service-latest | Image Index   | June 25, 2025, 13:49:02 (UTC+07) | 83.61     | <a href="#">Copy URI</a> | <a href="#">sha256:c28d3efef192a9bc...</a> | June 25, 2025, 13:50:36 (UTC+07) |
| -                      | Image         | June 25, 2025, 13:49:01 (UTC+07) | 83.61     | <a href="#">Copy URI</a> | <a href="#">sha256:2d68ae44612975...</a>   | June 25, 2025, 13:49:02 (UTC+07) |
| -                      | Image         | June 25, 2025, 13:49:01 (UTC+07) | 0.00      | <a href="#">Copy URI</a> | <a href="#">sha256:5e5b463a01bdb3...</a>   | -                                |
| user-service-latest    | Image Index   | June 25, 2025, 13:48:52 (UTC+07) | 83.60     | <a href="#">Copy URI</a> | <a href="#">sha256:9267074990c110...</a>   | June 25, 2025, 13:50:53 (UTC+07) |

Hình 32. Các images sau khi đã được đẩy lên ECR

- Để cho các Service hoạt động một cách an toàn và bảo mật, ta cần tạo một mạng VPC với các subnet riêng biệt (public và private), nhóm bảo mật (Security Groups), bảng định tuyến (Route Tables), Internet Gateway sẽ forward các traffic của services ra ngoài internet, cụ thể là đến Load Balancer và NAT Gateway sẽ route traffic từ private subnet đến public subnet đảm bảo các tài nguyên nội bộ không bị phơi bày trực tiếp ra Internet, cụ thể kết quả như các hình sau:



Hình 33. Kết quả tạo VPC

- Ta có 4 subnet chính với 2 private subnet và 2 public subnet để phân chia lưu lượng đồng đều cho web app:

The screenshot shows the AWS VPC Subnets page. It lists four subnets under the 'Info' tab:

| Name                  | Subnet ID                | State     | VPC                             | Block Public... | IPv4 CIDR    | IPv6 |
|-----------------------|--------------------------|-----------|---------------------------------|-----------------|--------------|------|
| vroux-prod-private-1a | subnet-019ee1817d6f73d02 | Available | vpc-09b01337dd982e0d4   vrou... | Off             | 10.0.11.0/24 | -    |
| vroux-prod-public-1b  | subnet-0f6189e627a3bd9e8 | Available | vpc-09b01337dd982e0d4   vrou... | Off             | 10.0.2.0/24  | -    |
| vroux-prod-public-1a  | subnet-0dc8a2185710a3032 | Available | vpc-09b01337dd982e0d4   vrou... | Off             | 10.0.1.0/24  | -    |
| vroux-prod-private-1b | subnet-0202a01b3ba0f2e7c | Available | vpc-09b01337dd982e0d4   vrou... | Off             | 10.0.12.0/24 | -    |

Hình 34. Các subnet có trong mạng VPC

- Route table được cấu hình để định tuyến phù hợp giữa các subnet trong VPC: các public subnet có route mặc định (0.0.0.0/0) trả về Internet Gateway để truy cập Internet 2 chiều, trong khi các private subnet có route mặc định trả về NAT Gateway để truy cập Internet một chiều (chỉ outbound).

The screenshot shows the AWS VPC Route tables page. It lists three route tables under the 'Info' tab:

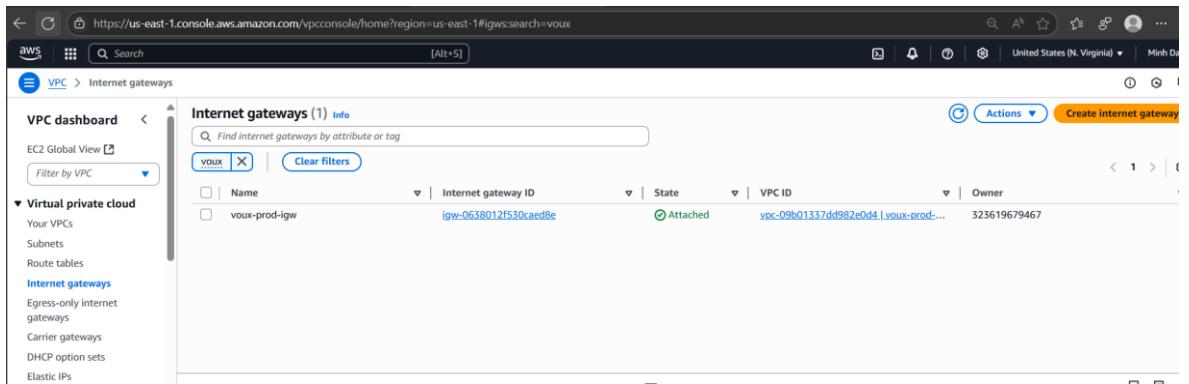
| Name                     | Route table ID        | Explicit subnet assoc... | Edge associations | Main | VPC                             | Owner ID |
|--------------------------|-----------------------|--------------------------|-------------------|------|---------------------------------|----------|
| vroux-prod-private-rt-1b | rtb-0c53176baaf60f51b | subnet-0202a01b3ba0f2... | -                 | No   | vpc-09b01337dd982e0d4   vrou... | 3236196  |
| vroux-prod-private-rt-1a | rtb-034ba76f34755386e | subnet-019ee1817d6f73... | -                 | No   | vpc-09b01337dd982e0d4   vrou... | 3236196  |
| vroux-prod-public-rt     | rtb-07dff4192795dcf3c | 2 subnets                | -                 | No   | vpc-09b01337dd982e0d4   vrou... | 3236196  |
| -                        | rtb-092ee5b36ba7a6ae1 | -                        | -                 | Yes  | vpc-09b01337dd982e0d4   vrou... | 3236196  |

Hình 35. Route table để route traffic

The screenshot shows the AWS VPC NAT gateways page. It lists two NAT gateways under the 'Info' tab:

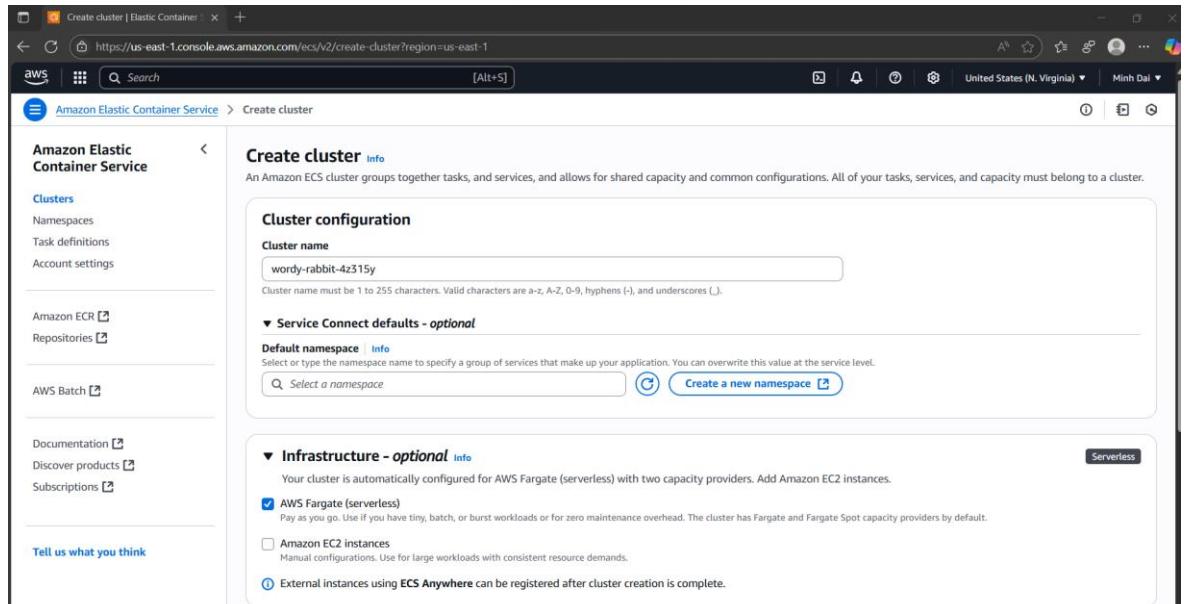
| Name              | NAT gateway ID        | Connectivity... | State     | State message | Primary public I... | Primary private I... | Primary netwo... |
|-------------------|-----------------------|-----------------|-----------|---------------|---------------------|----------------------|------------------|
| vroux-prod-nat-1b | nat-062673d8c825a972c | Public          | Available | -             | 3.221.221.69        | 10.0.2.32            | eni-0bd79159c... |
| vroux-prod-nat-1a | nat-04bcd0e9c851c5fdb | Public          | Available | -             | 3.208.76.253        | 10.0.1.43            | eni-011c6a65d... |

Hình 36. Kết quả tạo NAT Gatewat



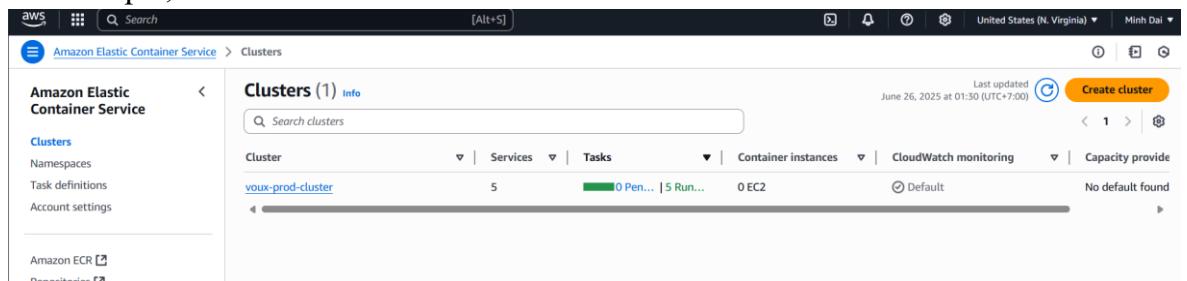
Hình 37. Kết quả tạo Internet Gateway

- Sau khi đã có các images trên ECR, ta cần tạo và setting một AWS ECS cluster để sử dụng các image này để backend có thể hoạt động. Đầu tiên, ta tiến hành tạo cluster với cluster type là Fargate để tiết kiệm chi phí vì Fargate chạy serverless:



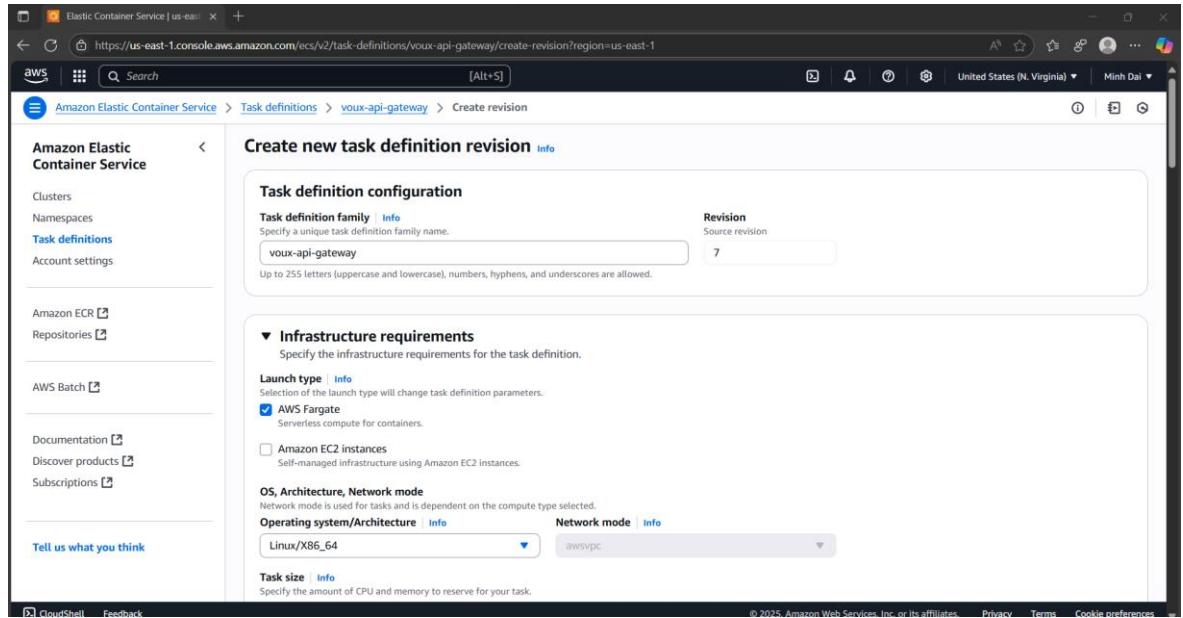
Hình 38. Giao diện tạo ECS Cluster

- Kết quả, ta có 1 cluster ECS như Hình 8:



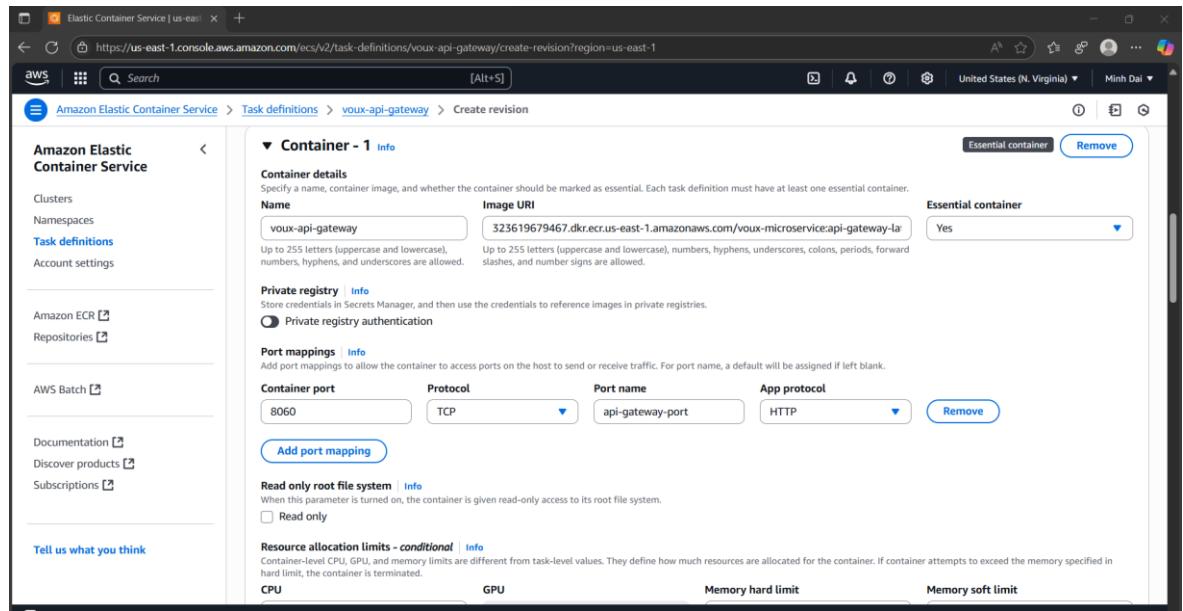
Hình 39. ECS Cluster sau khi tạo

- Tiếp theo, ta cần tạo 1 Task definition, Task Definition là nơi 1 Task để các service chạy và dùng các image trên ECR, nó không giống như EC2 sẽ chạy liên tục mà chỉ chạy khi được nêu gọi là Task (Có thể thấy như Hình 8 có hiển 0 EC2 Instance đang chạy):



*Hình 40. Giao diện tạo Task Definition*

- Trong Task Definition, ta sẽ tiến hành cấu hình Launch type là AWS Fargate, operating system để chạy container là Linux/x86\_64 như trong Hình 9, sau đó ta sẽ thêm URI của Docker Image lưu ECR để nó chạy và port 8060, ở đây là port của API Gateway, protocol HTTP như Hình 10:



Hình 41. Cấu hình Task Definition ECS

- Vì khi code có dùng các biến môi trường .env nên ta sẽ setting để thêm các biến môi trường đó như Hình 11, để đảm bảo các service lấy các biến đó và chạy bình thường:

#### ▼ Environment variables - optional

**Environment variables** | [Info](#)

Add individually

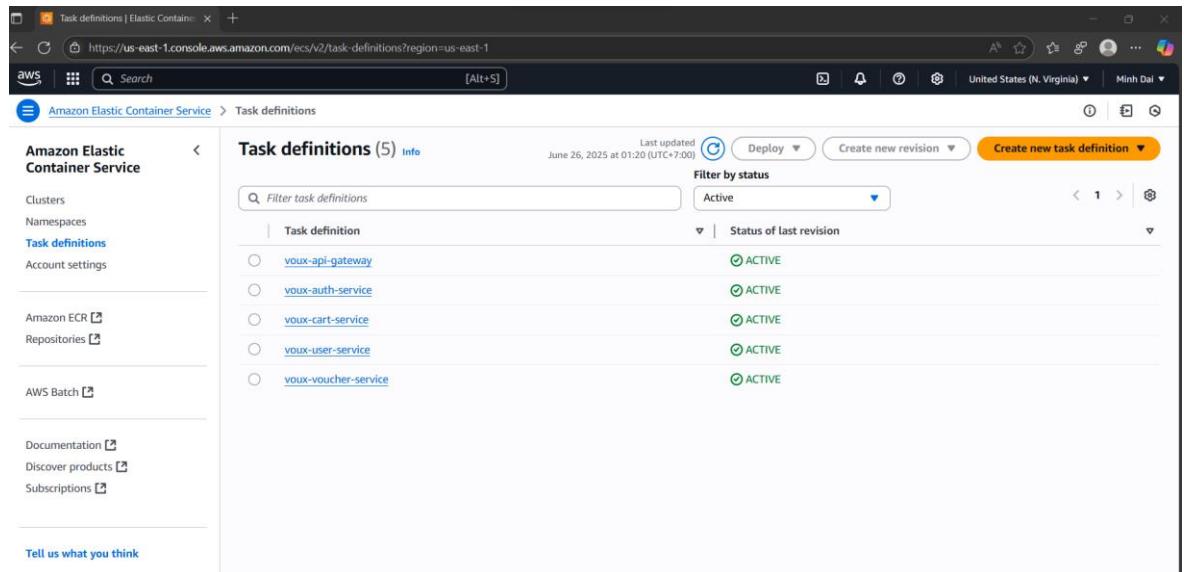
Add a key-value pair to specify an environment variable.

| Key                | Value type | Value                   | Remove                 |
|--------------------|------------|-------------------------|------------------------|
| AUTH_SERVICE_URL   | Value      | http://auth-service:300 | <a href="#">Remove</a> |
| CART_SERVICE_URL   | Value      | http://cart-service:300 | <a href="#">Remove</a> |
| USER_SERVICE_URL   | Value      | http://user-service:300 | <a href="#">Remove</a> |
| VOUCHER_SERVICE_UF | Value      | http://voucher-service: | <a href="#">Remove</a> |

[Add environment variable](#)

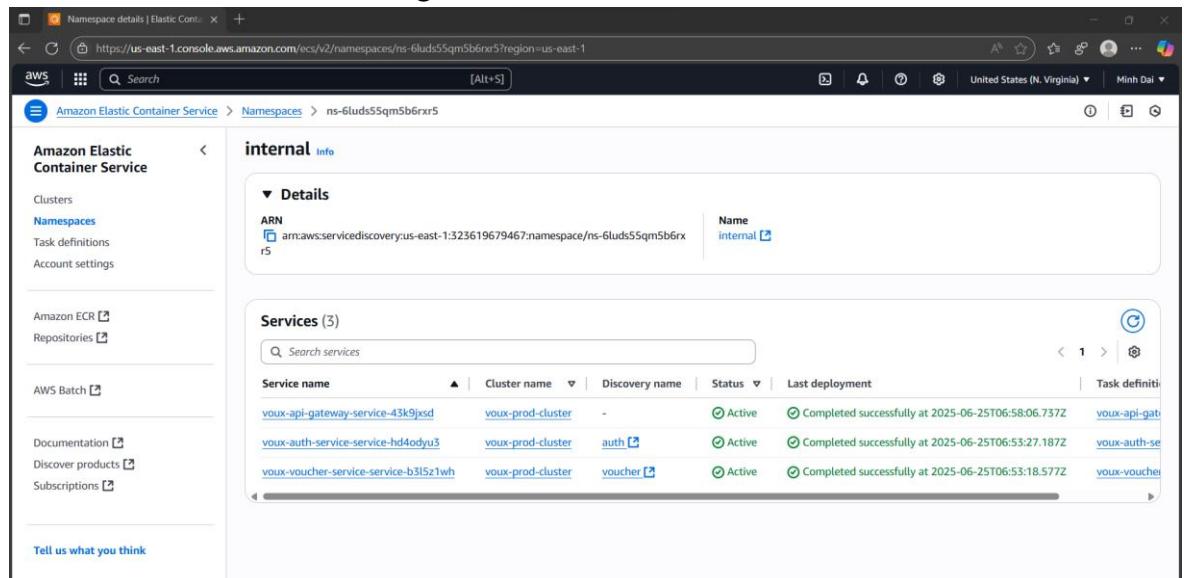
Hình 42. Các biến Enviroment trong Task Definition của ECS

- Tiến hành setting tương tự với các services khác, ta được 5 task definition là vox-api-gateway, vox-auth-service, vox-user-service, vox-cart-service và vox-voucher-service như Hình 12:

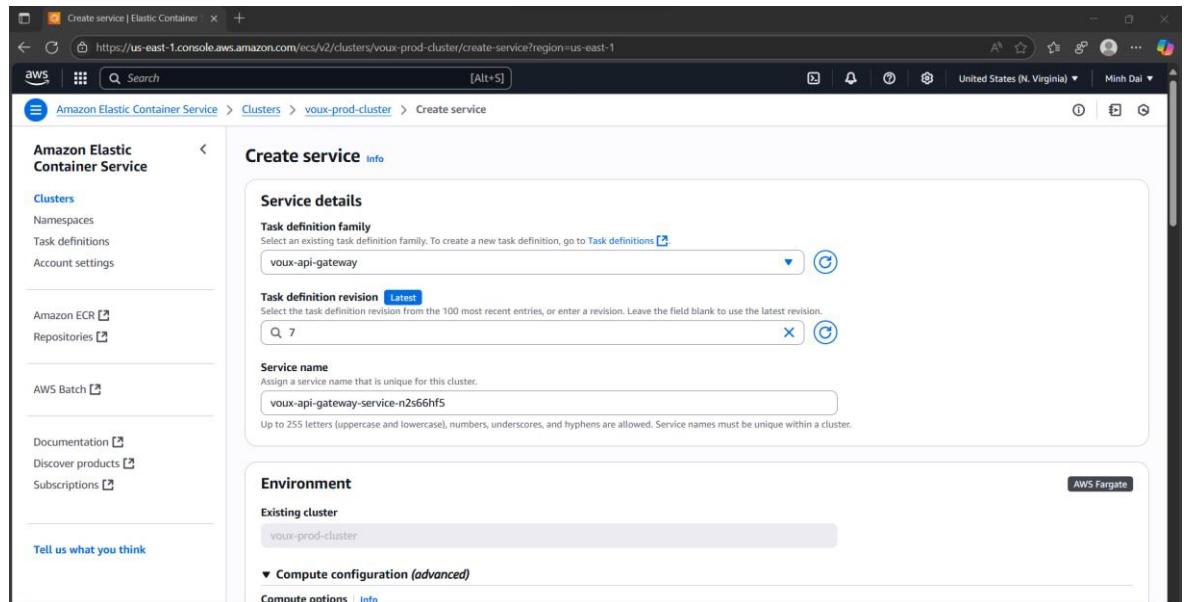


Hình 43. Kết quả tạo thành công các Task definitions

- Để cho các Services có thể giao tiếp với nhau, ta cần tạo 1 namespace, namespace sẽ là các endpoint nội bộ chỉ các service mới kết nối được với nhau, bên ngoài sẽ ko truy cập được, Namespace này sau đó sẽ được dùng trong Service Connect, tính năng của ECS Services.



Hình 44. Các services đang sử dụng Namespace là Internal



Hình 45. Giao diện tạo 1 Service trên ECS

### ▼ Networking

**VPC** | [Info](#)  
Select a VPC to use for your Amazon ECS resources.

vpc-09b01337dd982e0d4
▼
[Create a new VPC](#)

**Subnets**  
Choose the subnets within the VPC that the task scheduler should consider for placement.

Choose subnets
▼
[Clear current selection](#)

|                          |            |              |
|--------------------------|------------|--------------|
| subnet-019ee1817d6f73d02 | public     | x            |
| voux-prod-private-1a     | us-east-1a | 10.0.11.0/24 |
| subnet-0f6189e627a3bd9e8 | public     | x            |
| voux-prod-public-1b      | us-east-1b | 10.0.2.0/24  |
| subnet-0dc8a2185710a3032 | public     | x            |
| voux-prod-public-1a      | us-east-1a | 10.0.1.0/24  |
| subnet-0202a01b3ba0f2e7c | public     | x            |
| voux-prod-private-1b     | us-east-1b | 10.0.12.0/24 |

**Security group** | [Info](#)  
Choose an existing security group or create a new security group.

Use an existing security group  
 Create a new security group

Hình 46. Cấu hình VPC cho ECS Services

Hình 47. Setting Service Connect trong Api-gateway-service

- Nhưng vì auth-service là nơi để nhận traffic từ API-gateway và cũng là nơi giao tiếp với các service khác nên ta sẽ cấu hình Service Connect là Client and Server với các namespace tương tự là Internal, Discovery name là auth và DNS để gắn vào biến môi trường của API Gateway như Hình 11:

Hình 48. Setting Service Connect trong Auth-service

- Kết quả ta được 5 service chính được deploy thành công và hoạt động bình thường:

The screenshot shows the AWS Elastic Container Service (ECS) Cluster Services page. On the left, there's a sidebar with 'Amazon Elastic Container Service' and 'Clusters' selected. The main area has tabs for 'Services', 'Tasks', 'Infrastructure', 'Metrics', 'Scheduled tasks', 'Configuration', and 'Tags'. Under 'Services', it says 'Draining' and 'Active 5'. Below this, there's a table for 'Services (5)' with columns for 'Service name', 'ARN', 'Status', 'Service...', 'Created at', and 'Deployments and tasks'. The table lists five services, all in 'Active' status with 'REPLICA' type, created 13 days ago, and each having 1/1 task running.

| Service name                          | ARN                                                                              | Status | Service... | Created at  | Deployments and tasks                               |
|---------------------------------------|----------------------------------------------------------------------------------|--------|------------|-------------|-----------------------------------------------------|
| voux-api-gateway-service-43k9jxsd     | arn:aws:ecs:us-east-1:123456789012:service/voux-api-gateway-service-43k9jxsd     | Active | REPLICA    | 13 days ago | <span style="width: 100%;">1/1 Tasks running</span> |
| voux-auth-service-service-hd4od0yu5   | arn:aws:ecs:us-east-1:123456789012:service/voux-auth-service-service-hd4od0yu5   | Active | REPLICA    | 13 days ago | <span style="width: 100%;">1/1 Tasks running</span> |
| voux-cart-service-service-ef7ukfe8    | arn:aws:ecs:us-east-1:123456789012:service/voux-cart-service-service-ef7ukfe8    | Active | REPLICA    | 13 days ago | <span style="width: 100%;">1/1 Tasks running</span> |
| voux-user-service-service-131mlt9a    | arn:aws:ecs:us-east-1:123456789012:service/voux-user-service-service-131mlt9a    | Active | REPLICA    | 13 days ago | <span style="width: 100%;">1/1 Tasks running</span> |
| voux-voucher-service-service-b3l5z1wh | arn:aws:ecs:us-east-1:123456789012:service/voux-voucher-service-service-b3l5z1wh | Active | REPLICA    | 13 days ago | <span style="width: 100%;">1/1 Tasks running</span> |

Hình 49. Kết quả deploy các Services trên cụm ECS

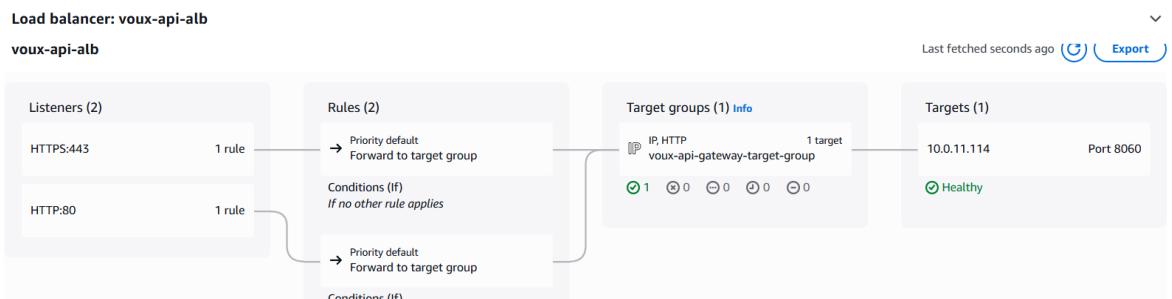
- Tiếp đến, để đảm bảo khả năng phân phối và xử lý lượng truy cập lớn từ phía người dùng một cách hiệu quả, ta cần tạo một Load Balancer để cân bằng tải cho traffic đi từ Frontend đến API Gateway. Load Balancer này sẽ đóng vai trò làm tầng trung gian, định tuyến các yêu cầu đến các dịch vụ tương ứng dựa trên loại giao thức và cổng được cấu hình. Cụ thể, ta sẽ tạo hai Target Group: một sử dụng giao thức HTTP port 80 và một sử dụng HTTPS port 443. Đối với target group dùng HTTPS, ta sẽ cấu hình với một chứng chỉ SSL tùy chỉnh đã được

cấp phát và ký bởi ZeroSSL, chứng chỉ này có tên miền định danh là api.voux-platform.shop. Kết quả cấu hình được thể hiện ở Hình 20 bên dưới:

The screenshot shows the AWS EC2 Load Balancers console. On the left sidebar, under the 'Instances' section, there is a 'Load balancers' link. The main pane displays a table titled 'Load balancers (1/1)'. A single row is listed: 'voux-api-alb' with a status of 'Active', VPC ID 'vpc-09b01337dd982e0d4', 2 Availability Zones, Type 'application', and Date created 'June 12, 2021'. Below this, a detailed view for 'Load balancer: vox-api-alb' is shown. It lists two listeners: 'HTTP:80' and 'HTTPS:443'. Both listeners have a 'Default action' of 'Forward to target group' and point to the same target group 'voux-api-gateway-target-group'. This target group has one target, '10.0.11.114' on port 8060, and is marked as 'Healthy'.

Hình 50. Kết quả tạo Load Balancer

- Load Balancer này định tuyến lưu lượng HTTP và HTTPS từ người dùng đến cổng 8060 của API Gateway thông qua một target group duy nhất như resource map của Hình 21:



Hình 51. Chi tiết Resource Map của Load Balancer

- Thêm Custom Certificate từ ZeroSSL trong load balancer vào HTTPS port 443 (đã upload lên AWS Certificate Manager) đảm bảo traffic từ Frontend đến API Gateway và ngược lại được mã hóa :

**Secure listener settings** [Info](#)

**Security policy** [Info](#)  
Your load balancer uses a Secure Socket Layer (SSL) negotiation configuration called a security policy to manage SSL connections with clients. [Compare security policies](#)

**Security category**  
[All security policies](#) ▾ [ELBSecurityPolicy-TLS13-1-2-Res-2021-06 \(recommended\)](#) ▾

**Default SSL/TLS server certificate**  
The certificate used if a client connects without SNI protocol, or if there are no matching certificates. You can source this certificate from AWS Certificate Manager (ACM), Amazon Identity and Access Management (IAM), or import a certificate from your listener certificate list.

**Certificate source**  
 From ACM  From IAM  Import certificate

**Certificate (from ACM)**  
The selected certificate will be applied as the default SSL/TLS server certificate for this load balancer's secure listeners.  
 api.voux-platform.shop ▾ 

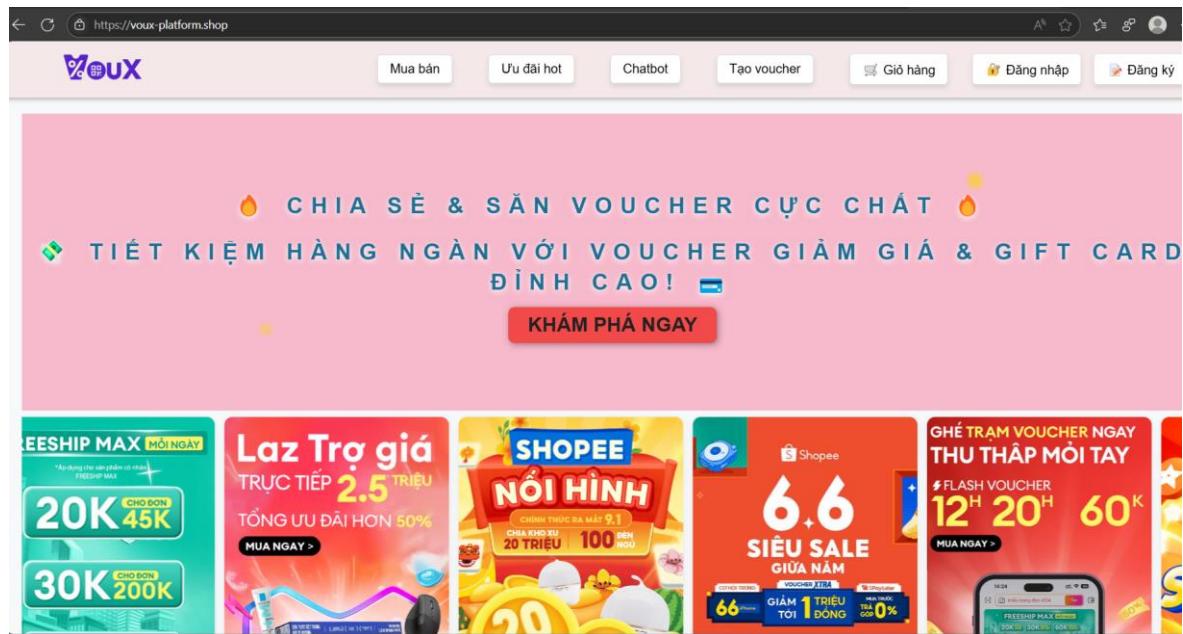
[Request new ACM certificate](#)

**Client certificate handling** [Info](#)  
Client certificates are used to make authenticated requests to remote servers. [Learn more](#)

**Mutual authentication (mTLS)**  
Mutual TLS (Transport Layer Security) authentication offers two-way peer authentication. It adds a layer of security over TLS and allows your services to verify the client that's making the connection.

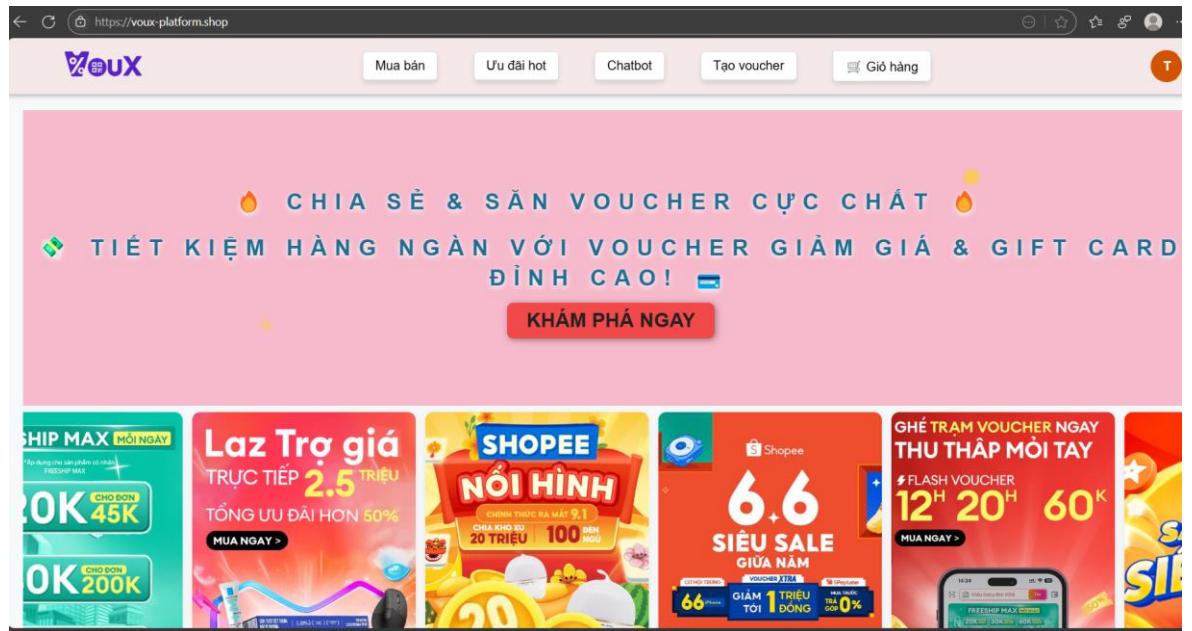
Hình 52. Thêm Custom Certificate từ ZeroSSL vào Load Balancer Target Group

- Web UI sau khi đã triển khai:



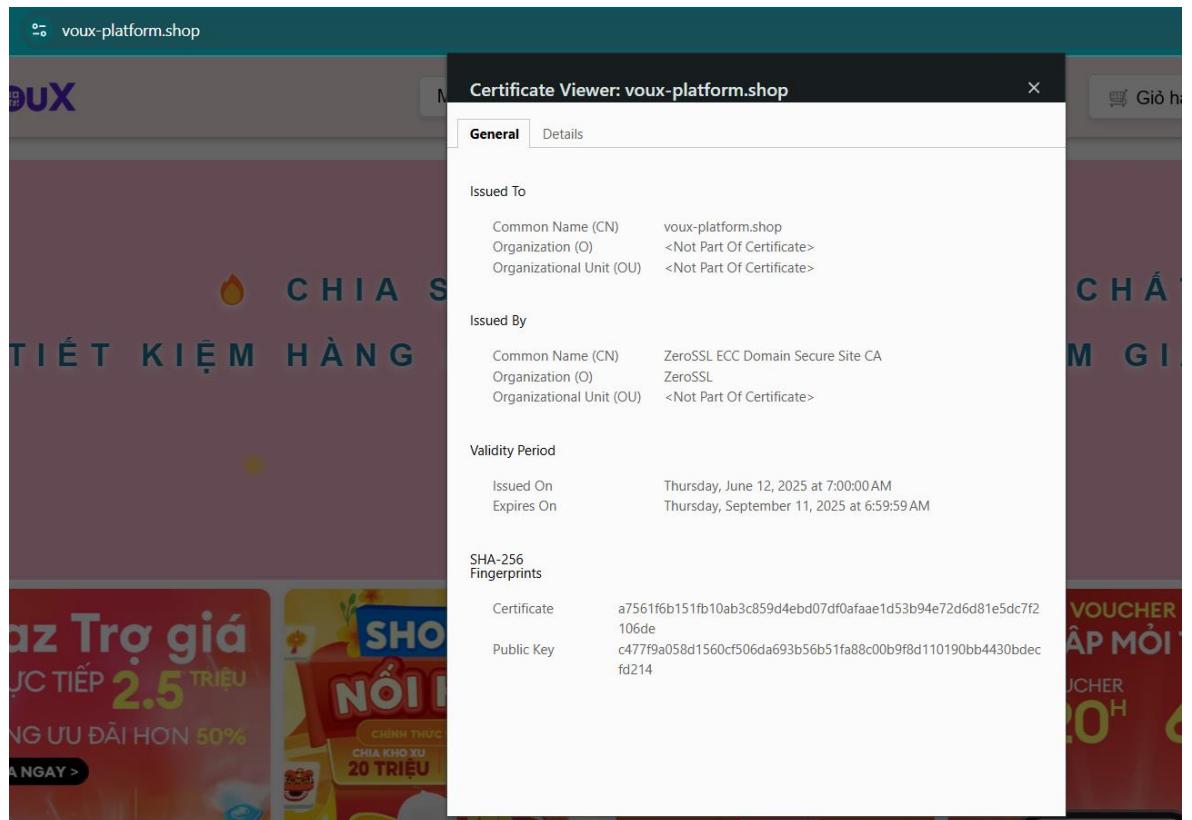
Hình 53. Web UI Chính của vox-platform.shop

- Test chức năng đăng ký và đăng nhập:



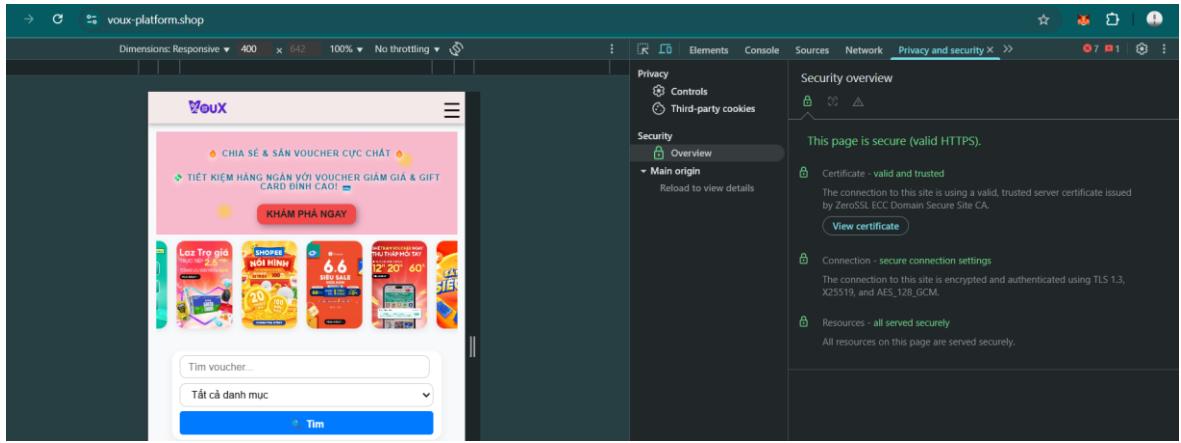
Hình 54. UI khi user đã đăng nhập

- Check certificate từ User đến Frontend:

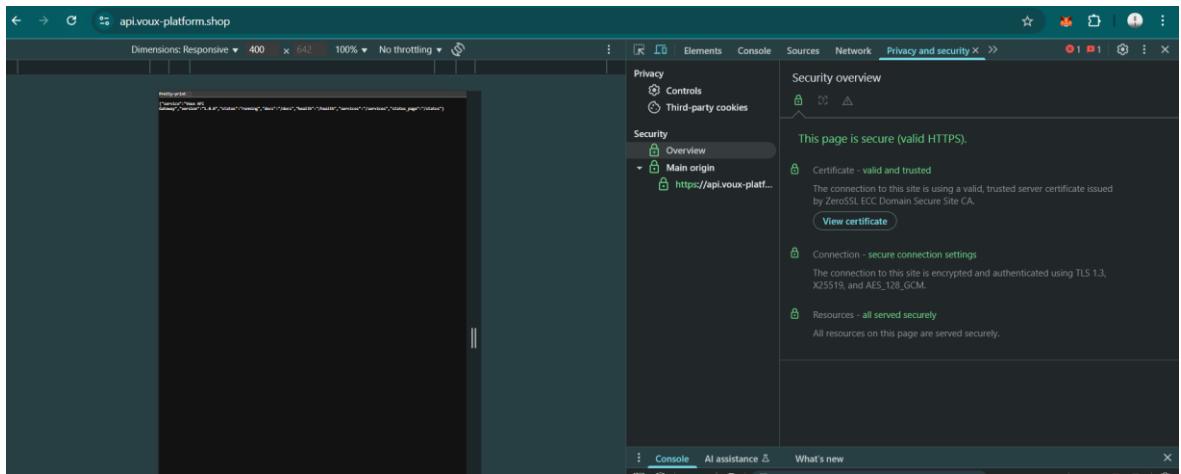


Hình 55. Certificate trên domain voux-platform.shop

- TLS trên domain `voux-platform.shop` là TLS 1.3 cũng như `api.voux-platform.shop`, xác định qua tab Privacy and Security trên browser:

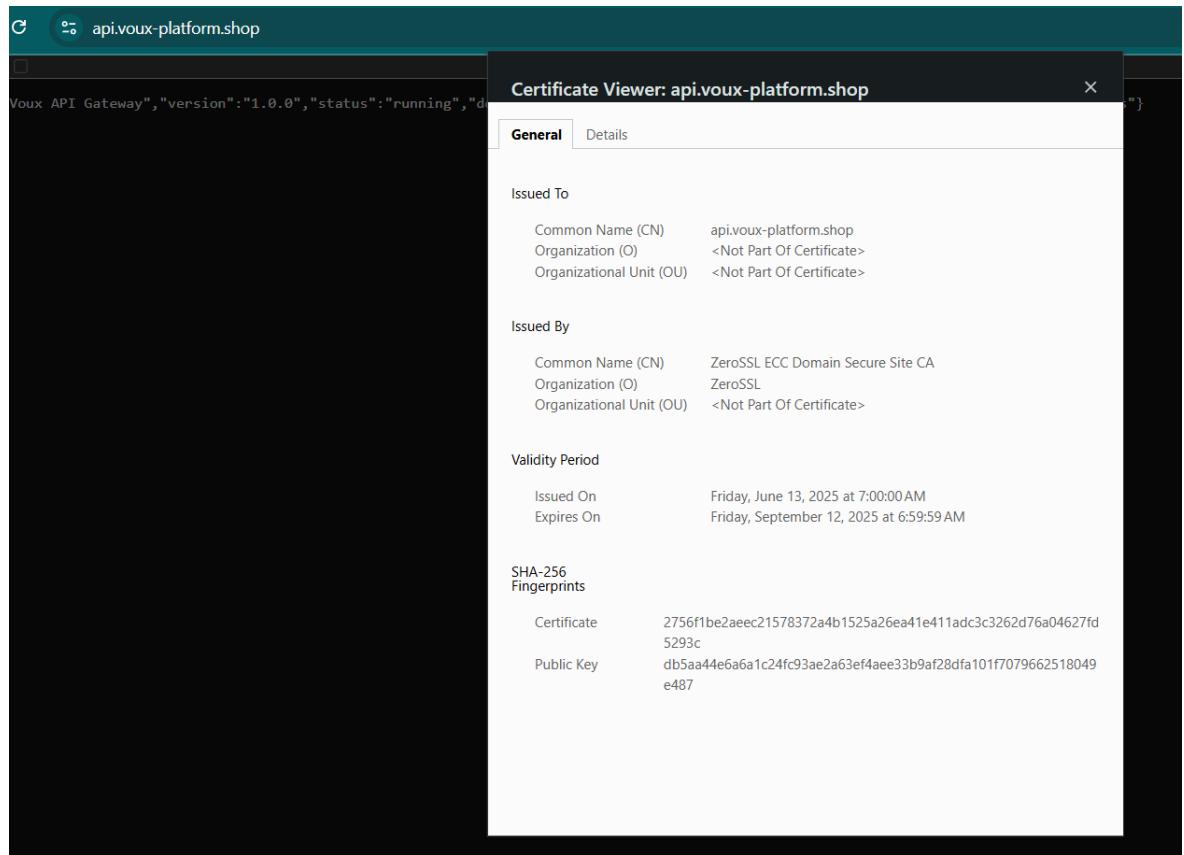


Hình 56. Phiên bản TLS sử dụng là TLS 1.3 cho domain `voux-platform.shop`



Hình 57. Phiên bản TLS sử dụng là TLS 1.3 cho domain `api.voux-platform.shop`

- Check certificate từ Frontend đến API:



*Hình 58. Certificate trên api.voux-platform.shop*