TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐOÀN NGỌC NAM - 52000084**

**TÌM HIỂU VỀ MICROSERVICE VÀ GRAPHQL XÂY DỰNG WEBSITE ĐÁNH GIÁ DOANH NGHIỆP VỀ LĨNH VỰC IT**

**DỰ ÁN CÔNG NGHỆ THÔNG TIN**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐOÀN NGỌC NAM - 52000084**

**TÌM HIỂU VỀ MICROSERVICE VÀ GRAPHQL XÂY DỰNG WEBSITE ĐÁNH GIÁ DOANH NGHIỆP VỀ LĨNH VỰC IT**

**DỰ ÁN CÔNG NGHỆ THÔNG TIN**

Người hướng dẫn

**Thạc Sĩ Doãn Xuân Thanh**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

**LỜI CẢM ƠN**

Để hoàn thành dự án môn học này, em xin gửi lời cảm ơn chân thành đến Thạc sĩ Doãn Xuân Thanh. Với sự định hướng rõ ràng và những kiến thức quý báu mà thầy đã truyền đạt, em đã có thể hoàn thành đề tài đúng tiến độ và đạt được những kết quả như mong đợi. Những hướng dẫn và góp ý của thầy không chỉ giúp em hiểu rõ hơn về nội dung chuyên môn mà còn giúp em phát triển kỹ năng tự nghiên cứu và tư duy hệ thống.

Do hạn chế về kiến thức và kinh nghiệm, bài báo cáo này chắc chắn vẫn còn nhiều thiếu sót. Em rất mong nhận được những góp ý quý báu từ thầy cô để có thể hoàn thiện hơn trong tương lai.

*TP. Hồ Chí Minh, ngày 15 tháng 7 năm 2024*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Nam*

*Đoàn Ngọc Nam*

**CÔNG TRÌNH ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của Thạc sĩ Doãn Xuân Thanh. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình**. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 15 tháng 7 năm 2024*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Nam*

*Đoàn Ngọc Nam*

**TÌM HIỂU VỀ MICROSERVICE VÀ GRAPHQL XÂY DỰNG WEBSITE ĐÁNH GIÁ DOANH NGHIỆP VỀ LĨNH VỰC IT**

**TÓM TẮT**

Bài báo cáo này trình bày quá trình nghiên cứu và phát triển một hệ thống quản lý đánh giá công ty dựa trên kiến trúc microservice, sử dụng GraphQL và NestJS. Mục tiêu chính của hệ thống là phân chia các chức năng thành các service độc lập, giúp dễ dàng quản lý và mở rộng. Hệ thống được chia thành ba microservice chính: User Service, Company Service và Review Service, mỗi service có cơ sở dữ liệu riêng. Báo cáo cung cấp cái nhìn tổng quan về quy trình thiết kế và triển khai hệ thống, từ việc phân tích yêu cầu, xây dựng kiến trúc, cho đến cấu hình và triển khai các microservice. Cuối cùng, những ưu và nhược điểm của việc sử dụng kiến trúc microservice trong bối cảnh dự án cũng được thảo luận.

**RESEARCH ON MICROSERVICES INTEGRATED WITH GRAPHQL FOR WEB DEVELOPMENT APPLICATION OF IT COMPANY RATING**

**ABSTRACT**

This report presents the research and development of a company review management system based on a microservice architecture, utilizing GraphQL and NestJS. The main objective of the system is to divide functionalities into independent services, allowing for easier management and scalability. The system is divided into three core microservices: User Service, Company Service, and Review Service, each with its own database. The report provides an overview of the design and implementation process, from requirement analysis, architecture development, to microservice configuration and deployment. Finally, the report discusses the pros and cons of adopting a microservice architecture in the context of the project.

**MỤC LỤC**

[DANH MỤC HÌNH VẼ vii](#_Toc175062934)

[DANH MỤC CÁC CHỮ VIẾT TẮT viii](#_Toc175062935)

[CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI 1](#_Toc175062936)

[1.1 Cơ sở thực tiễn 1](#_Toc175062937)

[1.2 Phạm vi nghiên cứu 2](#_Toc175062938)

[CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ 4](#_Toc175062939)

[2.1 Phân tích và thiết kế 4](#_Toc175062940)

[2.1.1 Sơ đồ usecase 4](#_Toc175062941)

[2.1.2 Thiết kế cơ sở dữ liệu 5](#_Toc175062942)

[2.1.3 Thiết kế các module 7](#_Toc175062943)

[CHƯƠNG 3. KIẾN TRÚC HỆ THỐNG 13](#_Toc175062944)

[3.1 Thiết kế microservice 13](#_Toc175062945)

[3.1.1 Lợi ích và thách thức 14](#_Toc175062946)

[3.1.2 Gateway 14](#_Toc175062947)

[3.2 GraphQL 16](#_Toc175062948)

[CHƯƠNG 4. TRIỂN KHAI THỰC HIỆN 18](#_Toc175062949)

[4.1 Microservices và gateway 18](#_Toc175062950)

[4.2 Database per service and outsource service 19](#_Toc175062951)

[4.3 Giao tiếp với Apollo 21](#_Toc175062952)

[4.3.1 Kiến trúc và cách hoạt động 21](#_Toc175062953)

[4.3.2 Áp dụng 21](#_Toc175062954)

[4.4 Mutation và Query 22](#_Toc175062955)

[4.5 Lưu trữ ảnh với Cloudinary 26](#_Toc175062956)

[CHƯƠNG 5. KẾT QUẢ VÀ HƯỚNG PHÁT TRIỂN 28](#_Toc175062957)

[5.1 Kết quả **Error! Bookmark not defined.**](#_Toc175062958)

[5.2 Hướng phát triển 28](#_Toc175062959)

# DANH MỤC HÌNH VẼ

[Hình 2.1: Sequence diagram đăng kí 9](#_Toc175062960)

[Hình 2.2: Đăng ký admin company 10](#_Toc175062961)

[Hình 2.3: Chức năng tạo công ty 11](#_Toc175062962)

[Hình 2.4: Chức năng viết một đánh giá 12](#_Toc175062963)

[Hình 3.1: Microservices thông thường 15](#_Toc175062964)

[Hình 3.2: Hệ thống với GraphQL 17](#_Toc175062965)

[Hình 4.1: company database 20](#_Toc175062966)

[Hình 4.2: User database 20](#_Toc175062967)

[Hình 4.3: Cloudinary console 27](#_Toc175062968)

[Hình 5.1: Playground testing 23](#_Toc175062969)

[Hình 5.2: Client call testing 23](#_Toc175062970)

[Hình 5.3: Login page 24](#_Toc175062971)

[Hình 5.4:Create company page 25](#_Toc175062972)

[Hình 5.5: Company overview 25](#_Toc175062973)

[Hình 5.6: Đánh giá công ty **Error! Bookmark not defined.**](#_Toc175062974)

# DANH MỤC CÁC CHỮ VIẾT TẮT

|  |  |
| --- | --- |
| CL | Client |
| DB | Database |
| API | Application Programming Interface |
| QL | Query Language |

# GIỚI THIỆU ĐỀ TÀI

## Cơ sở thực tiễn

Sau hàng chục năm phát triển, thế giới Internet ngày nay đã có những bước chuyển mình mạnh mẽ với vô vàn các công nghệ mới. Trong đó, các công nghệ để xây dựng và phát triển các ứng dụng web, tuy không phải là một cơn lốc nhanh và mạnh mẽ như những gì AI và Data Science đã làm được, nhưng lại là mảng có sự mở rộng nhiều nhất, nâng cấp nhiều nhất. Từ những ngày đầu thế kỉ khi chúng ta sử dụng HTML, CSS, JS cho việc xây dựng 1 web site tĩnh, sử dụng PHP để xây dựng hệ thống backend. Rồi đến sự ra đời của NodeJS, môi trường cho phép chạy ngôn ngữ JavaScript từ server thay vì chỉ có thể chạy trên trình duyệt như trước gần như đã mở ra 1 cuộc cách mạng về các framework và thư viện để xây dựng các ứng dụng web. Cùng thời điểm này, với nhu cầu mở rộng không ngừng của xã hội và nền kinh tế, các hệ thống công nghệ thông tin vốn đã trở thành một phần không thể thiếu trong mọi lĩnh vực cuộc sống, cũng đứng trước nhu cầu mở rộng không ngừng và cập nhật liên tục để trở nên phù hợp và đáp ứng các nhu cầu của người dùng. Khi này các hệ thống theo dạng Monolothic cũ đã lộ ra các khuyết điểm khi không thể đáp ứng các nhu cầu cập nhật liên tục và khó để bảo trì, mở rộng. Thời điểm ấy, Microservice được nhắc đến và ứng dụng rộng dãi hơn trong cộng đồng, mặc dù nó đã được các công ty công nghệ lớn như Amazon, Netflix tiên phong sử dụng từ những năm trước đó.

Cùng trong những năm này, Facebook, một trong những công ty công nghệ lớn nhất thế giới đang gặp phải một vấn đề đó là việc chia sẻ API giữa các nền tảng, hoặc các thành phần trên cùng 1 nền tảng. Ví dụ dễ hình dung nhất chính là việc nếu ở trên website cần 1 API trả về 1 lượng thông tin, thì ở phía mobile, họ chỉ cần 1 phần của lượng thông tin đó. Để giải quyết vấn đề này GraphQL ra đời. Và cũng trong khoảng thời gian này, Facebook đã đem đến 1 cuộc cách mạng cho việc lập trình giao diện người dùng phía client, đó là việc cho ra đời thư viện ReactJS.

Những thuận lợi về công nghệ đã có, vậy còn nhu cầu thực tiễn của đề tài thì sao?

Không thể phủ nhận rằng, hiện nay sự cạnh tranh trong thị trường việc làm ngành Công nghệ thông tin là vô cùng lớn. Sự “thiếu hụt nhân lực IT” vẫn luôn là 1 tiêu đề quen thuộc của các bài báo mạng, tuy nhiên, điều đó chỉ đúng với nhân lực chất lượng cao, với nhiều năm kinh nghiệm. Nhưng dù ở level nào, việc lựa chọn 1 công ty tốt và phù hợp cho việc phát triển bản thân luôn là một vấn đề trăn trở. Trên các trang mạng hiện nay, các trang web cho phép review các công ty thường mang các yếu tố hoặc là quá tiêu cực, hoặc là quá seeding. Điều này vô tình gây ra một vài sự lưỡng lự trong việc tin tưởng và lựa chọn tiếp nhận thông tin.

Với những yếu tố thuận lợi kể trên là điều kiện lý tưởng để em thực hiện đề tài: Tìm hiểu về Microservices và GraphQL ứng dụng vào xây dựng website đánh giá công ty.

## Phạm vi nghiên cứu

Đề tài này được thực hiện trong khuôn khổ môn học "Dự Án Công Nghệ Thông Tin" và là một phần của khóa luận tốt nghiệp. Mục tiêu của nghiên cứu là xây dựng và phát triển một hệ thống phần mềm sử dụng kiến trúc microservices và công nghệ GraphQL, giúp giải quyết các vấn đề về khả năng mở rộng, tối ưu hóa hiệu suất và quản lý dữ liệu trong các ứng dụng hiện đại.

Phạm vi nghiên cứu bao gồm:

* **Phát triển kiến trúc microservices**: Nghiên cứu các nguyên tắc thiết kế và áp dụng mô hình microservices cho hệ thống, bao gồm việc triển khai các dịch vụ độc lập với cơ sở dữ liệu riêng và giao tiếp thông qua các API.
* **Ứng dụng GraphQL trong quản lý dữ liệu**: Khảo sát cách GraphQL tối ưu hóa việc truy vấn và cập nhật dữ liệu trong môi trường đa dịch vụ, đồng thời tích hợp với các công nghệ hiện đại khác.
* **Xây dựng hệ thống quản lý người dùng và dữ liệu**: Hệ thống này sẽ tập trung vào việc quản lý người dùng, công ty và các đánh giá với cơ chế phân quyền rõ ràng, đảm bảo tính bảo mật và tiện dụng.

Phạm vi của đề tài chỉ tập trung vào các khía cạnh kỹ thuật và chức năng của hệ thống, không đi sâu vào các vấn đề kinh tế, xã hội hay các lĩnh vực ngoài công nghệ. Hệ thống được phát triển ở mức độ học thuật nhằm phục vụ cho mục đích học tập và nghiên cứu, với khả năng mở rộng và áp dụng trong thực tiễn nếu có nhu cầu.

# PHÂN TÍCH VÀ THIẾT KẾ

## Phân tích và thiết kế

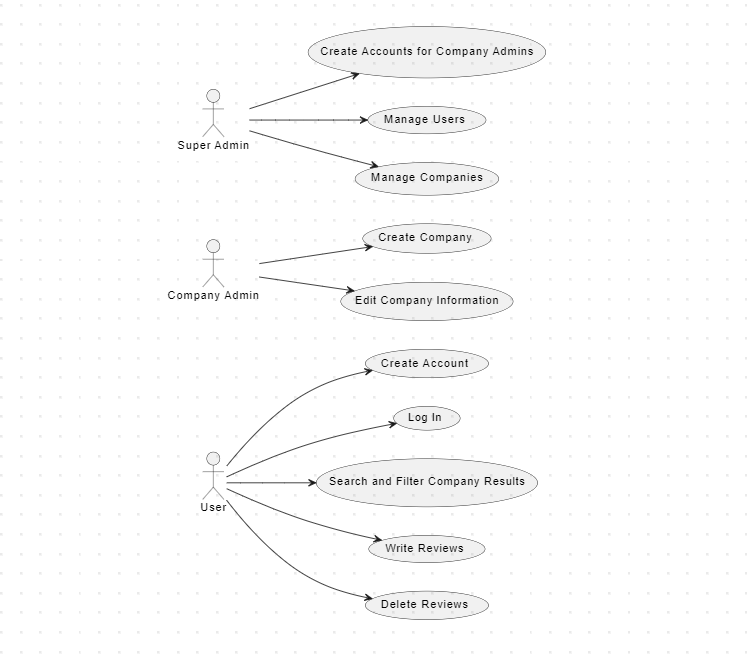
### Sơ đồ usecase

Với một hệ thống như đã được giới thiệu ở trên, ngoài việc phân chia các module sao cho phù hợp và mượt mà, chúng ta cũng cần quan tâm đến việc các bên giao tiếp với nhau để dễ dàng hơn trong việc phân chia services, database sau này.

Hệ thống sẽ bao gồm 3 tác nhân (actor) tương ứng với các vai trò và quyền hạn khác nhau bao gồm:

* User: đại diện cho người dùng thông thường, cũng là người dùng cuối cùng của hệ thống, tác nhân này có thể tạo tài khoản (account), đăng nhập (login), tìm kiếm và lọc kết quả các công ty (search) theo các tiêu chí và yêu cầu khác nhau, viết bài đánh giá (review) và xóa bài đánh giá
* Company admin: người quản lý cho 1 công ty, là người chịu trách nhiệm quản lý thông tin và hình ảnh của công ty mình trên hệ thống. Company admin có thể tạo một công ty, sửa thông tin cho công ty, đưa ra các phản hồi và thu thập thông tin đánh giá công ty của mình.
* Super admin: quản trị viên, là người duy trì toàn bộ hoạt động của hệ thống, đảm bảo rằng hệ thống hoạt động trơn chu quản trị viên sẽ là người tạo và cung cấp tài khoản cho Company admin, quản lý tất cả người dùng và các công ty trong hệ thống

Dưới đây là sơ đồ use case tổng quát của hệ thống, cho thấy các chức năng mà mỗi tác nhân có thể thực hiện.



Hình 2.1: Sơ đồ Use Case hệ thống

### Thiết kế cơ sở dữ liệu

Sau khi đã có được các service cần thiết, chung ta tiến hành thiết kế cơ sở dữ liệu cho hệ thống. Tuy rằng có các rang buộc về quan hệ cho mỗi bảng và mỗi thực thể, ở phần này, em vẫn sẽ thiết kế CSDL một các cơ bản nhất, phần rang buộc khóa ngoại và mối quan hệ sẽ được triển khai ở tầng Application.

**Bảng User**

Đây là bảng chứa các thông tin cơ bản của một người dùng (tên, email,…), bao gồm cả company admin. Ngoài các thông tin này, bảng sẽ có thuộc tính role/permission để xác định quyền thực hiện các hành động, và xác định người dùng là *thông thường* hay là *company admin.*

**Bảng Company**

Là bảng chứa các thông tin của 1 công ty (tên, email, địa chỉ, lĩnh vực,…). Một công ty sẽ được quản lý bởi một *company admin* duy nhất, và một *company admin* cũng chỉ được quản lý một công ty duy nhất.

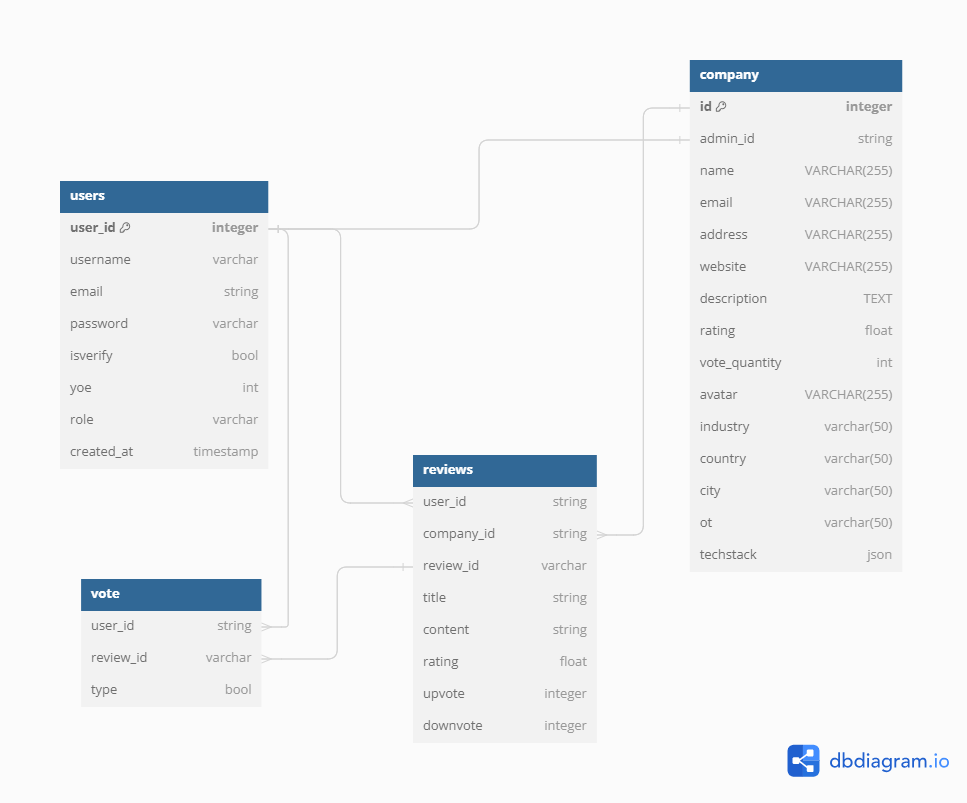
**Bảng Review**

Bảng chứa thông tin về mỗi nhận xét của người dùng cho một công ty, một người dùng có thể nhận xét một lần duy nhất về một công ty, và một công ty có thể nhận được nhiều lượt đánh giá từ nhiều nhân sự khác nhau. Mỗi đánh giá có thể nhận về lượt upvote/downvote từ người dùng khác.

**Bảng Vote**

Bảng xác định lượt upvote/downvote cho một đánh giá, đảm bảo rằng một người dùng chỉ có thể up/down vote 1 lần cho 1 đánh giá, và chỉ có thể là một trong hai giá trị này.

Dựa trên các thông tin đã có được từ những gì phân tích ở trên, chúng ta tiến hành khởi tạo cơ sở dữ liệu và vẽ lược đồ cơ sở dữ liệu:



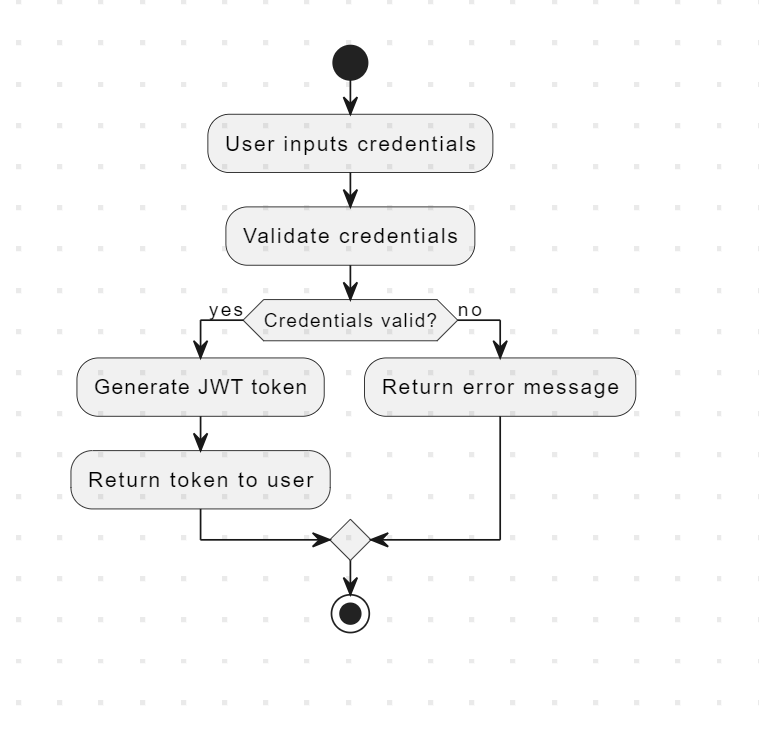
Hình 2.2: Lược đồ cơ sở dữ liệu

### Thiết kế các module

#### Login

Ở phần này, chúng ta sử dụng chiến lược xác thực cơ bản là Token Base Authentication với thư viện rất phổ biến JWT (Json Web Token). Quy trình xác thực diễn ra như sau:

* **Người dùng gửi yêu cầu đăng nhập:** Người dùng nhập thông tin đăng nhập (username và password) vào form đăng nhập và gửi yêu cầu tới server.
* **Xác thực thông tin đăng nhập**: Server kiểm tra thông tin đăng nhập của người dùng bằng cách đối chiếu với dữ liệu đã lưu trong cơ sở dữ liệu. Nếu thông tin đúng, server sẽ tạo ra một mã token (JWT - JSON Web Token).
* **Cung cấp Token cho người dùng:** Token được gửi lại cho người dùng cùng với phản hồi đăng nhập thành công. Khi người dùng thực hiện các yêu cầu đến server (ví dụ: truy cập trang dashboard), token sẽ được gửi kèm trong phần header của HTTP request

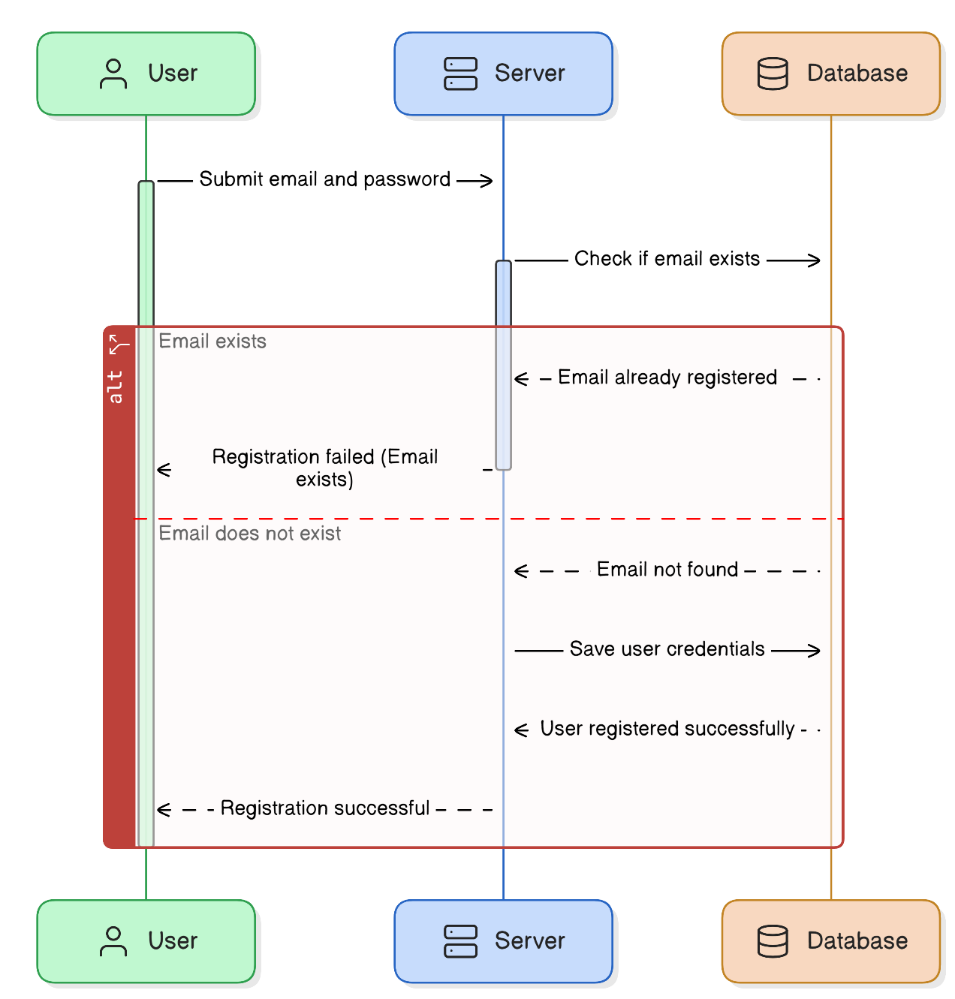


Hình 2.3: Activity Diagram chức năng Login

#### Đăng kí người dùng mới

Đối với đăng kí người dùng mới, chúng ta cũng sẽ có 3 bước cơ bản như trên, nhưng thay vì là sử dụng JWT để xác thực, chúng ta cần thao tác với database để tạo và lưu người dùng mới:

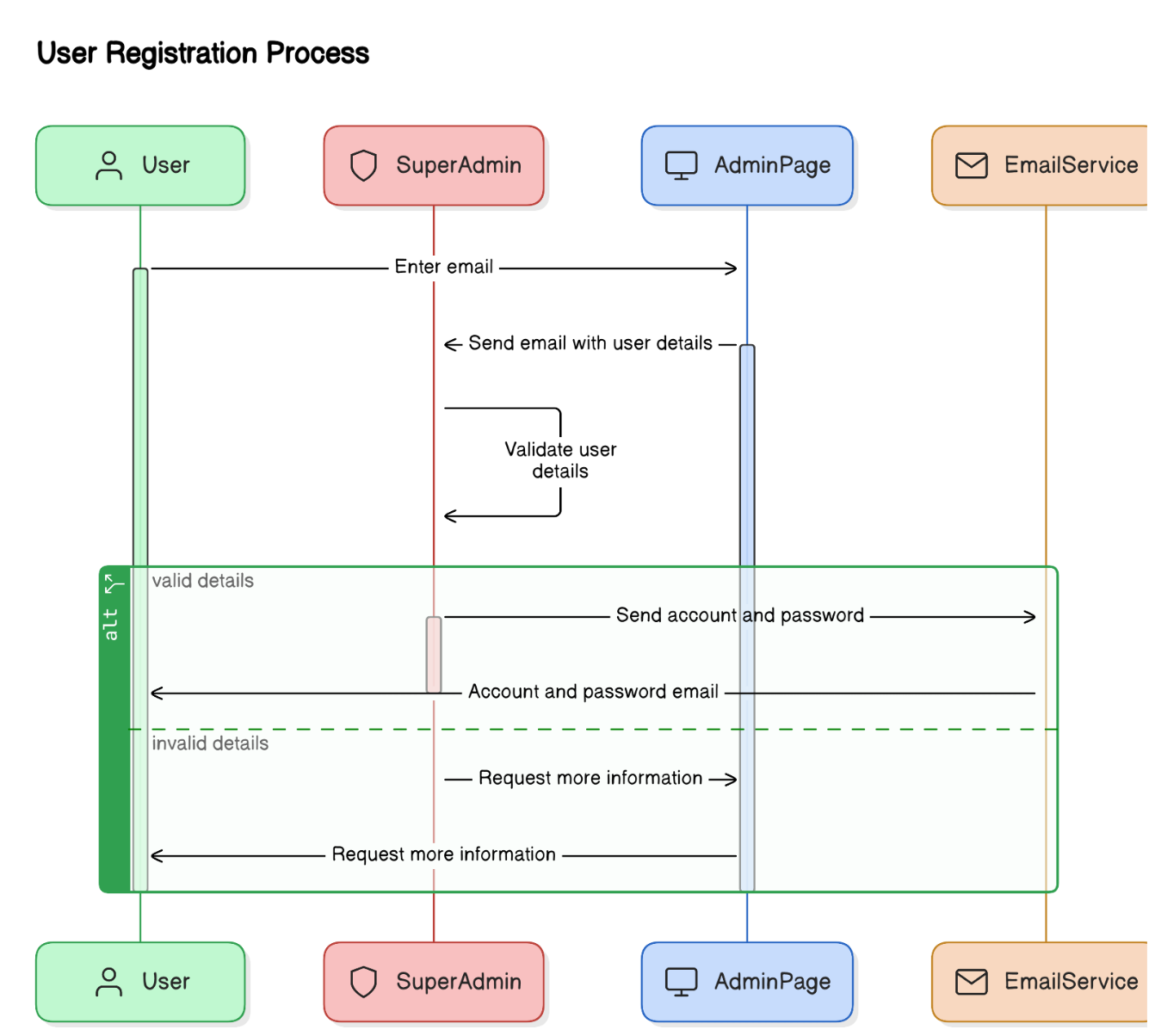
* **Người dùng gửi yêu cầu đăng nhập:** Người dùng nhập thông tin đăng kí (email và password) vào form đăng nhập và gửi yêu cầu tới server.
* **Server kiểm tra:** Server sẽ kiểm tra thông tin đăng kí của người dùng có hợp hệ và kiểm tra dữ liệu với database thông qua ORM. Nếu email chưa tồn tại, dữ liệu sẽ được lưu và ngược lại.
* **Trả về:** Nếu đăng kí thành công, tiến hành đăng nhập cùng lúc và trả về JWT token, ngược lại, gửi thông báo đăng kí thất bại cho người dùng:

****

Hình 2.1: Sequence diagram đăng kí

#### Đăng kí Admin

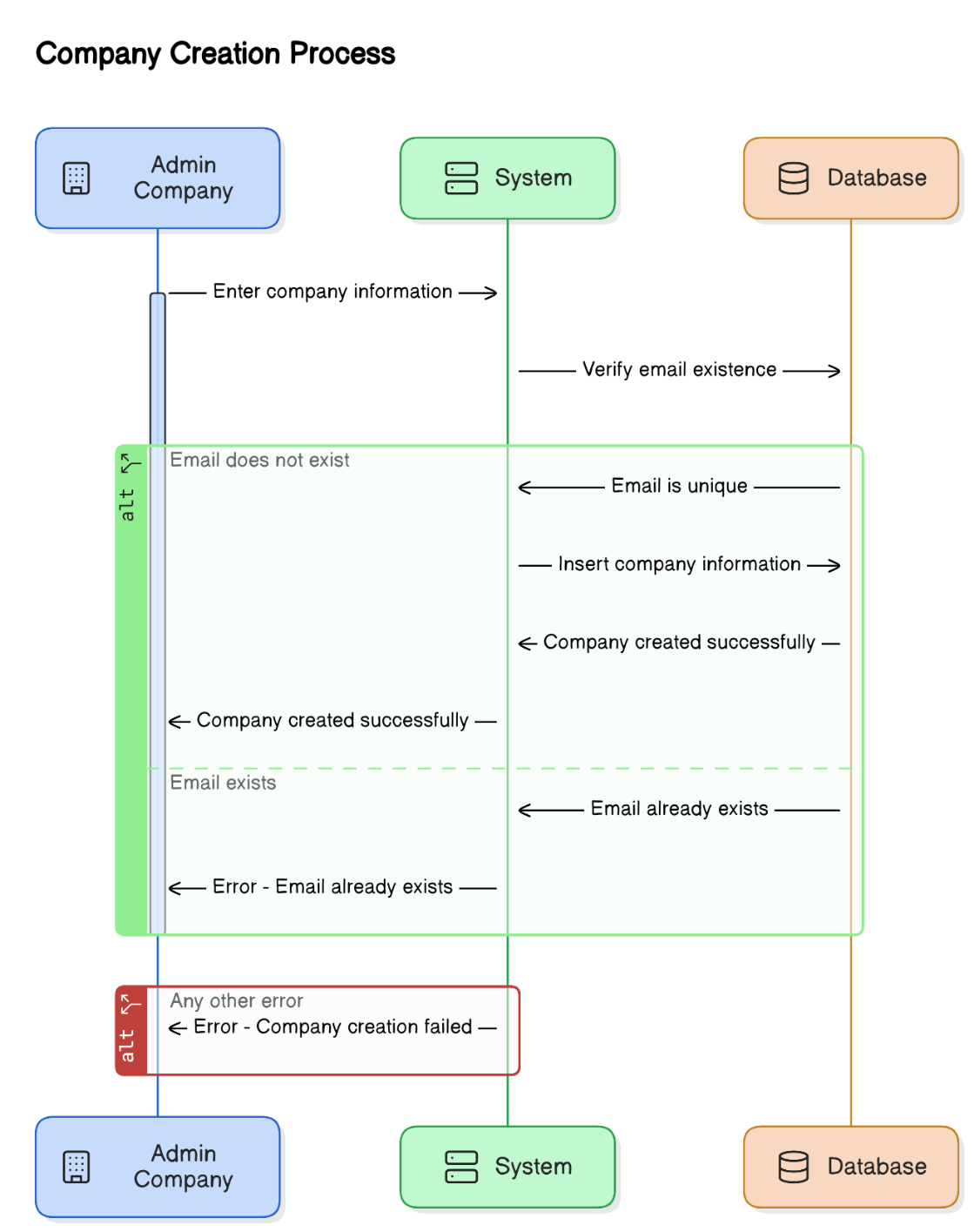
Đối với Admin company, người dùng phải liên hệ với super admin để sử dụng chức năng. Chúng ta sẽ tạo 1 trang để người dùng nhập email và gửi đến quản trị viên. Sau đó tài khoản và mật khẩu sẽ được gửi qua email. Điều này giúp đảm bảo tính thống nhất trông quản lý của hệ thống. Sau khi đăng kí thành công trở thành admin company, người dùng có thể bắt đầu đăng ký công ty của mình trên hệ thống để sử dụng và quản lý:



Hình 2.2: Đăng ký admin company

#### Tạo công ty

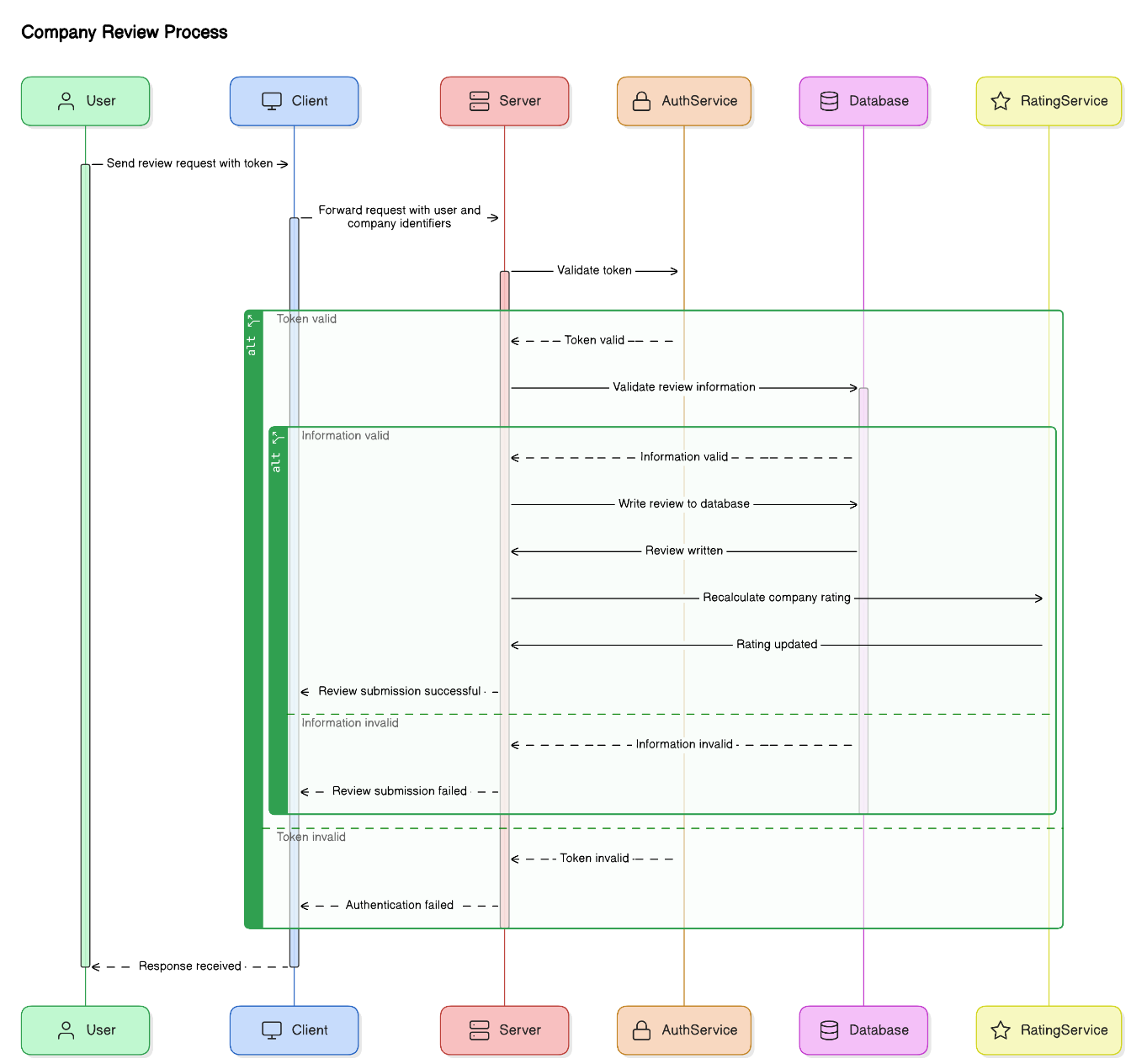
Đây là chức năng được phân quyền giành riêng cho admin company. Để tạo 1 công ty trên hệ thống, admin company cần nhập đầy đủ các thông tin đã được định nghĩa trong bảng ***company*** trong database. Email liên hệ của công ty bắt buộc phải cùng tên miền với email của admin company. Địa chỉ phải được xác thực trên Google. Sau khi hệ thống kiểm tra rằng email (index key của mỗi công ty) chưa tồn tại trên hệ thống, công ty sẽ được khởi tạo, ngược lại nếu có bất kì lỗi nào sẽ trả về giao diện lỗi:



Hình 2.3: Chức năng tạo công ty

#### Viết đánh giá

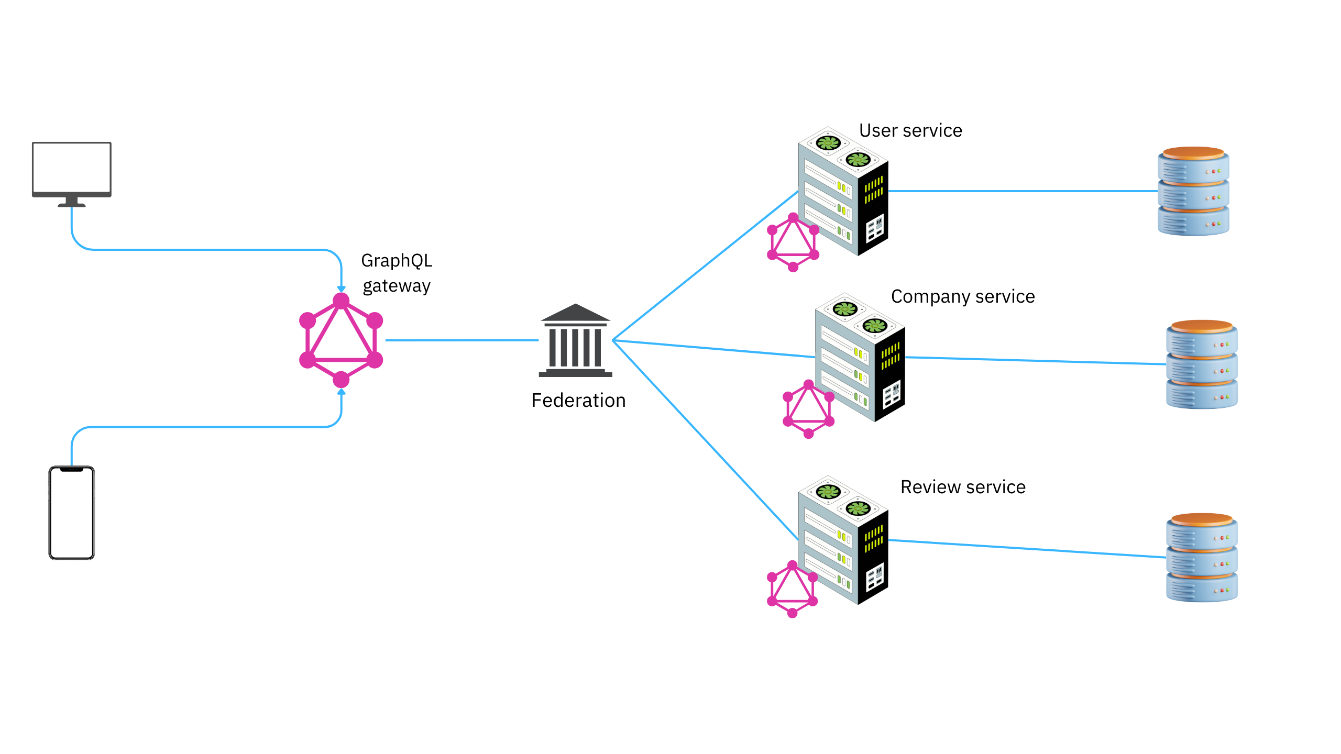
Đây là một chức năng phức tạp và có nhiều hành động đi kèm một lúc. Để có thể đánh giá một công ty, trước tiên người dùng cần gửi kèm Token thông qua Header của request. Đồng thời trước đó, người dùng cần nhập thông tin các điều mình muốn nhận xét về công ty bao gồm: Tiêu đề, nội dung chi tiết, số điểm mà họ đánh giá, có gợi ý công ty với người quen và bạn bè hay không. Khi gửi request, client sẽ gửi kèm một thông tin bao gồm định danh của người dùng và định danh công ty. Sau khi nhận request, server sẽ kiểm tra các thông tin, ghi vào database đồng thời tính toán lại điểm đánh giá của công ty:



Hình 2.4: Chức năng viết một đánh giá

# KIẾN TRÚC HỆ THỐNG

Dựa trên những thiết kế đã phân tích ở trên, chúng ta đi đến xây dựng kiến trúc tổng quát hơn với việc triển khai các module thành các service nhỏ lẻ, tách cơ sở dữ liệu ban đầu thành các cơ sở dư liệu nhỏ hơn và triển khai microservice cùng với GraphQL.



Hình 3.1: System Diagram

## Thiết kế microservice

Microservices (kiến trúc vi dịch vụ) là một phương pháp thiết kế hệ thống phần mềm dựa trên việc chia nhỏ ứng dụng thành các dịch vụ độc lập. Thay vì xây dựng một hệ thống đơn khối (monolithic) với tất cả các tính năng gói gọn trong một ứng dụng lớn, microservices chia hệ thống thành nhiều dịch vụ nhỏ hơn, mỗi dịch vụ chịu trách nhiệm cho một chức năng cụ thể. Các dịch vụ này hoạt động riêng biệt, giao tiếp với nhau qua các giao thức như HTTP, gRPC, hoặc qua message broker.

Về cơ bản, chúng ta không có 1 quy chuẩn nào toàn diện cho một hệ thống microservice. Mỗi một hệ thống với các nhu cầu khác nhau sẽ có các biến thể khác nhau và được điều chỉnh sao cho phù hợp. Ở đây, trong dự án với quy mô không quá lớn của chúng ta, hệ thống có thể được chia thành 3 service nhỏ, mỗi service trong đó tương ứng với một module trong hệ thống monolithic cũ:

* User service: Đảm nhiệm toàn bộ các tác vụ liên quan đến quản lý tài khoản người dùng, xác thực và phân quyền.
* Company service: Xử lý việc quản lý thông tin công ty, bao gồm tạo, cập nhật và tìm kiếm các thông tin về công ty.
* Review Service: Quản lý các bài đánh giá, bao gồm việc tạo mới, chỉnh sửa và xóa bài đánh giá.

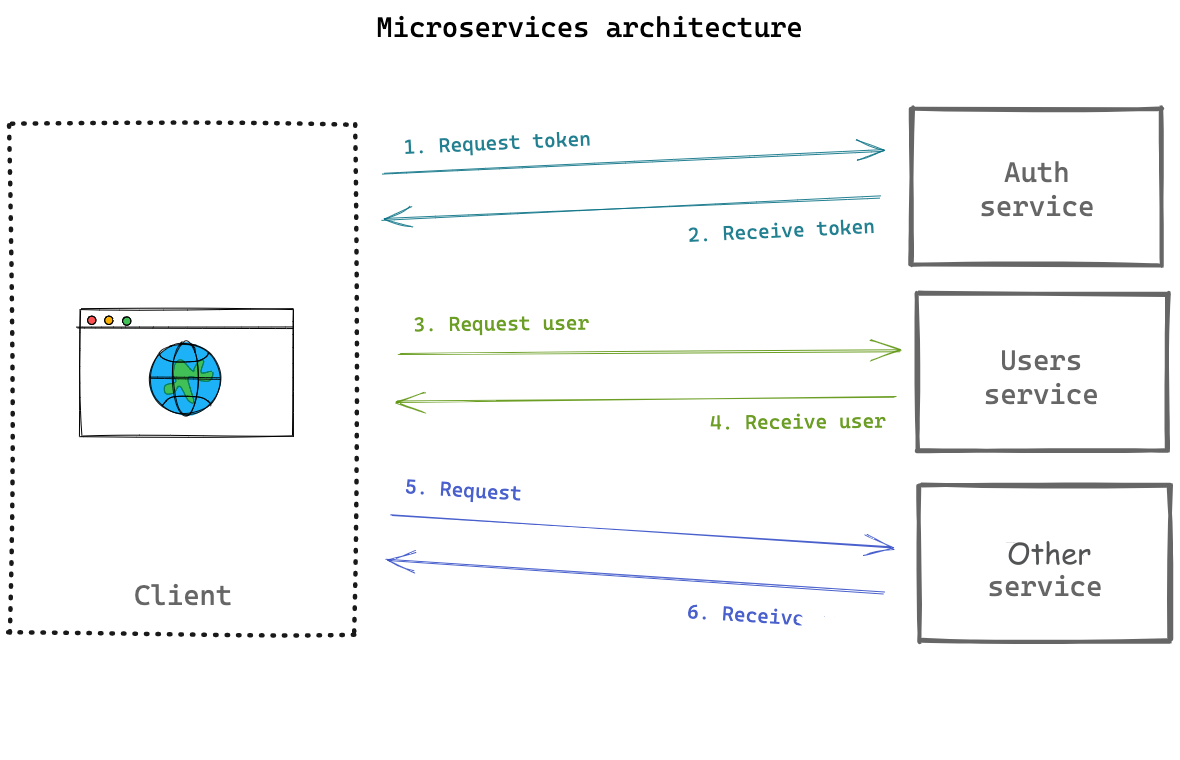
Về cả mặt logic và kỹ thuật, chúng ta cần đảm bảo rằng các microservice là riêng biệt về hoạt động và tách biệt nhau hoàn toàn về mặt hệ thống. Thông thường, việc áp dụng kiến trúc này chỉ được hiệu quả ở các dự án lớn và cực lớn, nơi có nhiều team đảm nhiệm các service khác nhau. Mỗi team sẽ phát triển service theo tốc độ độc lặp, với ngôn ngữ và nền tảng độc lập, điều này cũng khiến cho tốc độ phát triển dự án nâng lên rất nhiều. Tuy nhiên, với mục tiêu và phạm vi có hạn của dự án, các service vẫn sẽ được tách riêng với cơ chế xây dựng và hoạt động độc lập, những sẽ chỉ sử dụng chung 1 ngôn ngữ nền tảng duy nhất là NodeJS (NestJS).

### Lợi ích và thách thức

Một vấn đề phổ biến trong kiến trúc microservices là việc làm thế nào để client có thể tương tác với nhiều service mà không cần gửi nhiều yêu cầu riêng lẻ tới từng service. Điều này không chỉ gây tốn kém về mặt hiệu năng mà còn làm phức tạp quá trình phát triển và bảo trì ứng dụng. Để giải quyết vấn đề này, API Gateway đóng vai trò quan trọng như một lớp trung gian, điều phối toàn bộ các yêu cầu từ client và phân phối chúng tới các service tương ứng.

### Gateway

Với việc có nhiều service nằm trên các server (máy chủ) khác nhau, chúng ta xuất hiện một vấn đề đó là client không thể gửi lần lượt các request đến các domain hoặc server khác nhau để lấy hoặc gửi dữ liệu cho mỗi api hoặc service.



Hình 3.1: Microservices thông thường

Như ở sơ đồ bên trên, nếu theo đúng thiết kế bảo mật và Role Base Access Controll thông thường, người dùng phía client sẽ phải trải qua 2 services trước khi đến được với service cuối cùng. Client sẽ phải gửi 2 request trước khi có thể gửi request thứ cuối cùng. Như vậy tổng cộng sẽ là 3 requests và 6 network request/responses cho 1 mục đích sử dụng. Hơn nữa chúng ta sẽ phải điều chỉnh các API ở mỗi service sao cho chỉ nhận được lượng data vừa đủ để sử dụng cho các request tiếp theo.

Và, một vấn đề nữa ở đây là khi mỗi service được tách riêng biệt và độc lập, làm thế nào khi 1 service cần sử dụng đến tài nguyên và dịch vụ của một service khác, và làm thế nào khi chúng cần thiết có sự giao tiếp với nhau đồng thời cũng đảm bảo tính nhất quán của dữ liệu. Khi này, chúng ta cần đến 1 gateway, thứ sẽ giúp chúng ta phân phối, điều phối và che giấu cấu trúc của dự án và gateway này sẽ đồng thời được thiết kế như 1 datalayer giữa client và server.

## GraphQL

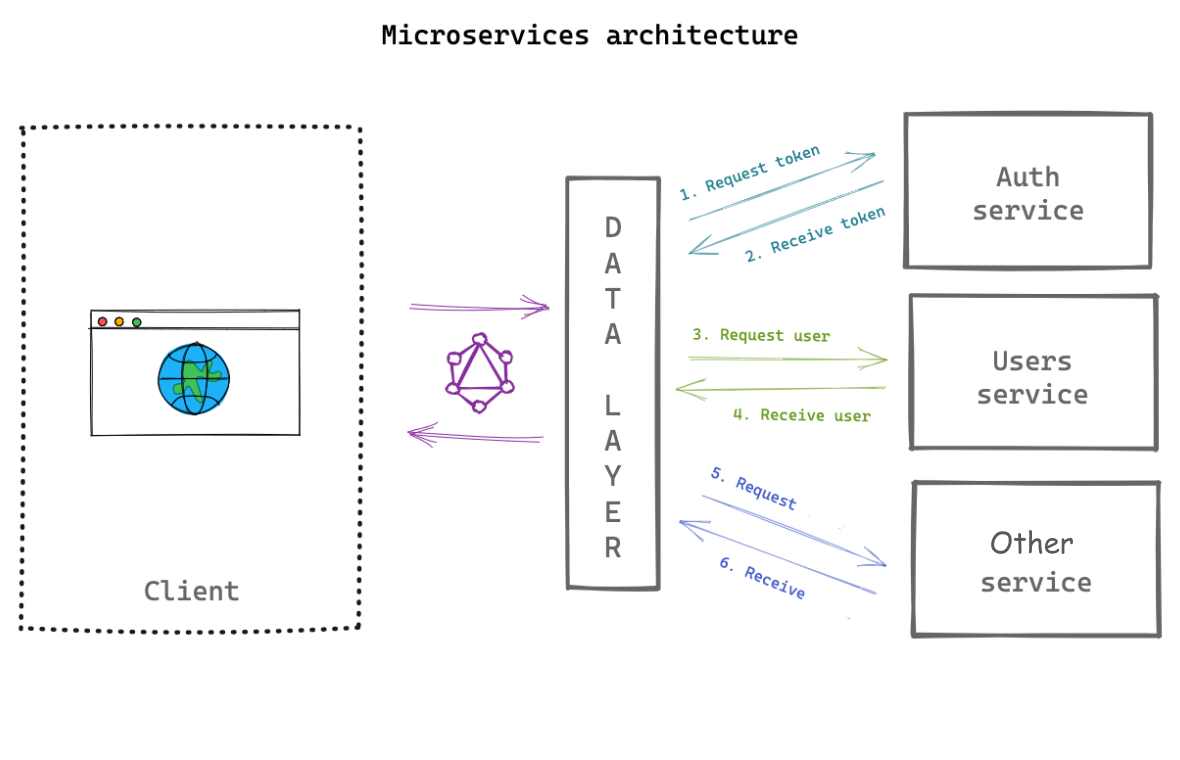
GraphQL là một ngôn ngữ truy vấn API và môi trường runtime được phát triển bởi Facebook vào năm 2015. Mục tiêu của GraphQL là cung cấp một cách linh hoạt và hiệu quả để tương tác với dữ liệu. Thay vì phải làm việc với các endpoint cố định như trong REST, GraphQL cho phép client yêu cầu chính xác những gì họ cần và không hơn thế.

* **Truy vấn có chọn lọc**: Client có thể chỉ định chính xác các trường dữ liệu cần lấy, giúp giảm thiểu lượng dữ liệu không cần thiết được truyền tải và tối ưu hóa hiệu suất.
* **Đột biến dữ liệu (Mutations)**: Bên cạnh truy vấn, GraphQL cũng hỗ trợ các thao tác cập nhật, thêm, và xóa dữ liệu một cách linh hoạt thông qua các đột biến.
* **Schema mạnh mẽ và tự mô tả**: GraphQL sử dụng schema để mô tả cấu trúc dữ liệu, giúp client biết được những truy vấn nào có sẵn và những kiểu dữ liệu nào có thể truy xuất. Điều này làm cho quá trình phát triển trở nên trực quan hơn.
* **Subscription**: GraphQL hỗ trợ việc cập nhật dữ liệu theo thời gian thực, cho phép client lắng nghe và nhận dữ liệu mới khi có sự thay đổi.

Như chúng ta đã đề cập, một hệ thống microservice sẽ chia các module thành các service nhỏ, được xây dựng, quản lý và phát triển độc lập. Điều này dẫn đến một yêu cầu đó là đồng thời chia nhỏ database (cơ sở dữ liệu) cho mỗi service (*Database per service*) . Tuy nhiên, đây là một vấn đề không dễ giải quyết, nó phức tạp và phổ biến đến nỗi có hẳn 1 mẫu thiết kế dành riêng cho nó, người ta còn viết ra hẳn các thư viện nhằm mục đích giải quyết vấn đề này (một cách dễ dàng hơn). Như đã nói bên trên, ở phía server, kiến trúc *share nothing ,*hoặc *Database per service* nhìn có vẻ rất phù hợp khi chúng ta phát triển từng service, những khi triển khai hệ thống và kết nối, đó sẽ là một cơn ác mộng.

Lúc này GraphQL sẽ cung cấp 1 phương pháp giải quyết vô cùng đơn giản và phù hợp. GraphQL cho phép chúng ta linh hoạt thay đổi và kiểm soát dữ liệu trả về. Đồng thời nó cũng cung cấp 1 phương thức xây dựng gateway vô cùng dễ dàng (tất nhiên chỉ dễ dàng cho các microservice con sử dụng GraphQL).

GraphQL và Microservice lúc này là sự kết hợp hoàn hảo. GraphQL cho phép chúng ta che dấu kiến trúc hệ thống microservice với client. Khi client gọi đến các services, nó chỉ cần thông qua một API duy nhất được cung cấp bởi GraphQL:



Hình 3.2: Hệ thống với GraphQL

# TRIỂN KHAI THỰC HIỆN

## Microservices và gateway

Trước tiên, chúng ta cần tạo ra 3 microservice đã được đề cập trước đó. Sau khi chạy lệnh khởi tạo một mono-repo, tương ứng một app nhỏ trong toàn dự án mà NestJS cung cấp, ta cần thực hiện cấu hình riêng tương ứng cho mỗi microservice và xây dựng ORM cho mỗi services.

Ở đây, chúng ta cần phân định rõ ràng giữa ORM để giao tiếp giữa database và application so với GraphQL để xây dựng mô hình cho các đối tượng. Vậy nên, định nghĩa model cho một service sẽ gồm có các decorator của cả ORM và GraphQL:

Sau đó là phần xây dựng riêng lẻ từng service một, đảm bảo việc mỗi service đều hoạt động trơn chu và độc lập.

Khi mỗi service, hoặc một service đã có thể hoạt động, chúng ta tiến hành xây dựng gateway cho hệ thống. Nếu như so với việc phải sử dụng REST api thông thường, việc xây dựng gateway, chia sẻ và kiểm soát tài nguyên là một bài toán vô cùng lớn, thì ở GraphQL, chúng ta có thể thở phào với Federation.

Federation trong GraphQL là một phương pháp cho phép bạn chia tách một GraphQL schema lớn thành nhiều schema nhỏ hơn và phân tán chúng giữa các microservice khác nhau, nhưng vẫn giữ được một API tổng hợp. Với GraphQL Federation, mỗi microservice quản lý một phần của schema tổng thể và các microservice này có thể hợp tác với nhau để cung cấp dữ liệu một cách liền mạch qua một cổng (gateway) duy nhất.

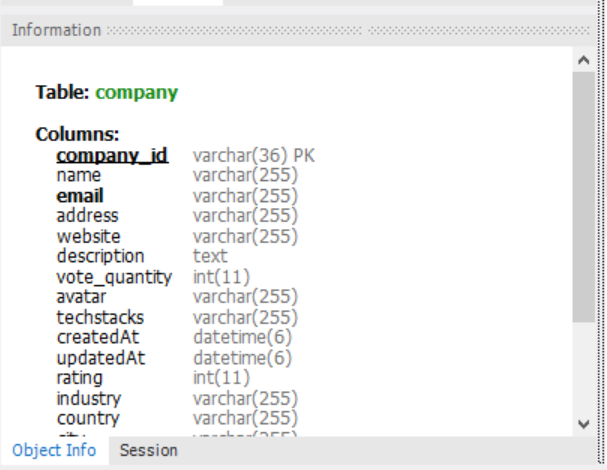


Và rồi, chỉ từ 1 API duy nhất, chúng ta có thể truy cập đến tất cả các services. Các hoạt động truy vấn cũng sẽ đồng thời thông qua datalayer của GraphQL.

## Database per service and outsource service

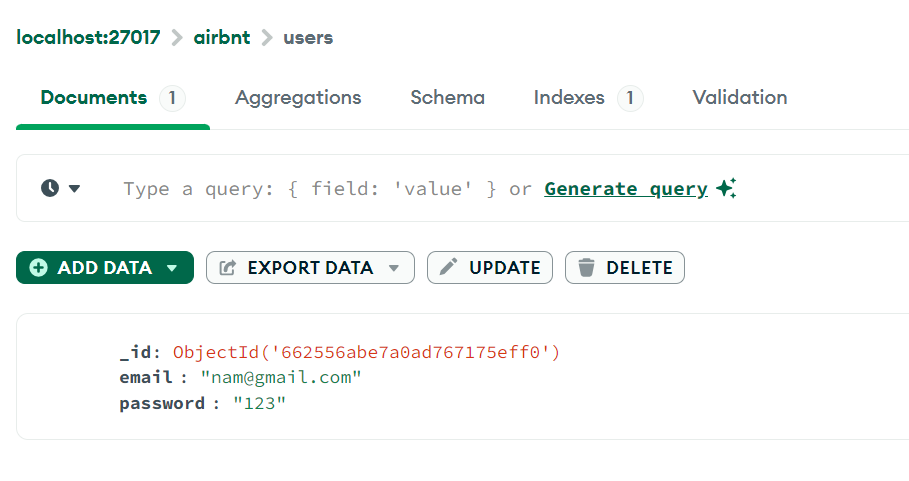
Để đảm bảo tính độc lập, riêng lẻ cho mỗi microservices, chúng ta tiến hành phân tách database thành các database nhỏ hơn và độc lập. Để dễ hình dung và minh chứng hơn, chúng ta sẽ tiến hành chọn database khác nhau. Ở đây sẽ là 3 loại:

* MySQL cho company service và review service nhằm mục đích đảm bảo sự nhất quán và toàn vẹn dữ liệu, điều này đảm bảo được chất lượng content hiển thị của trang web, cũng như việc đảm bảo dễ dàng cho việc quản lý mối quan hệ giữa công ty và các đánh giá sau này



Hình 4.1: company database

* MongoDB cho user service vì sự nhanh chóng và tiện lợi



Hình 4.2: User database

## Giao tiếp với Apollo

Apollo là một bộ công cụ mạnh mẽ cho phép phát triển các ứng dụng dựa trên GraphQL một cách hiệu quả. Với vai trò là một thư viện mã nguồn mở, Apollo giúp kết nối ứng dụng của bạn với API GraphQL, hỗ trợ việc quản lý trạng thái, truy vấn dữ liệu, và tối ưu hóa hiệu suất thông qua các tính năng như caching.

### Kiến trúc và cách hoạt động

Apollo có hai thành phần chính: Apollo Client và Apollo Server.

* **Apollo Client**: Được sử dụng trên frontend, Apollo Client hỗ trợ thực hiện các truy vấn, đột biến và đăng ký dữ liệu một cách dễ dàng. Nó tích hợp tốt với các framework frontend như React, Vue, và Angular.
* **Apollo Server**: Được sử dụng trên backend, Apollo Server cung cấp môi trường để xây dựng API GraphQL một cách đơn giản và linh hoạt. Nó tương thích với nhiều framework như Express, NestJS, v.v.

### Áp dụng

Chúng ta sẽ phải cấu hình Apollo trước tiên ở cả 2 phía client và server,chúng ta cần phải chú ý đến việc cấu hình đúng đường dẫn API.

* **Apollo Client**: Ở phía frontend (ReactJS), Apollo Client được cấu hình để kết nối với Apollo Server thông qua URL của endpoint GraphQL. Việc cấu hình bao gồm cài đặt Apollo Client, thiết lập ApolloProvider để bọc toàn bộ ứng dụng và cung cấp khả năng truy cập GraphQL cho các thành phần con.
* import { ApolloClient, InMemoryCache, ApolloProvider } *from* '@apollo/client';
* const client = new ApolloClient({
* uri: 'http://localhost:3009/graphql',
* cache: new InMemoryCache(),
* });
* **Apollo Server**: Ở backend (NestJS), Apollo Server được tích hợp để quản lý các truy vấn và đột biến. Server được cấu hình sử dụng các schema GraphQL và resolver để xử lý logic.



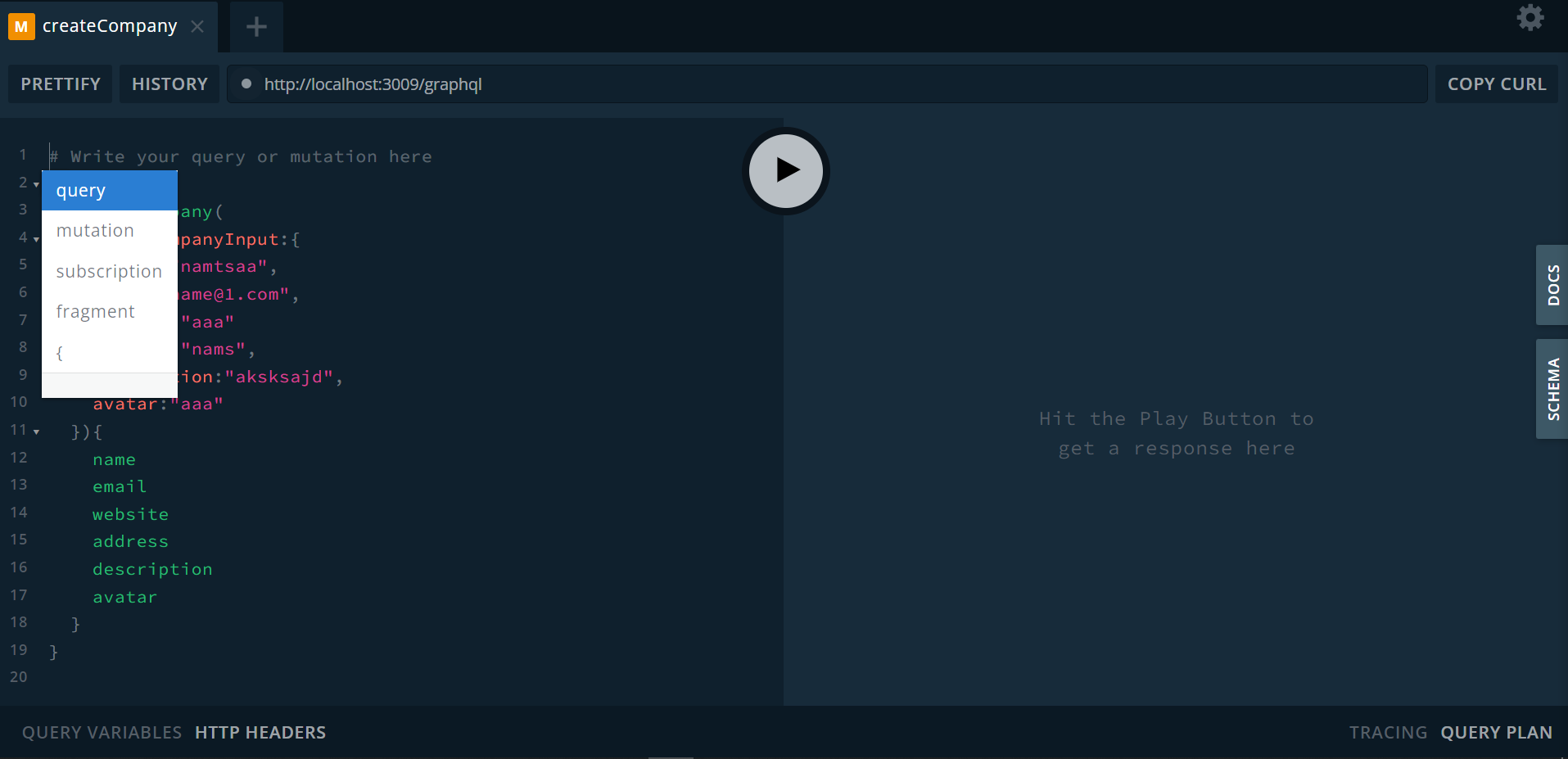
## Mutation và Query

Trong GraphQL, hai khái niệm chính cần hiểu rõ là Query và Mutation. Chúng là cách thức tương tác với hệ thống thông qua GraphQL API:

* Query được sử dụng để lấy dữ liệu từ server. Nó giống như các lệnh GET trong REST API, giúp truy vấn dữ liệu mà không làm thay đổi trạng thái của hệ thống. Ví dụ, một query có thể được sử dụng để lấy danh sách các công ty hoặc thông tin chi tiết của một công ty cụ thể:

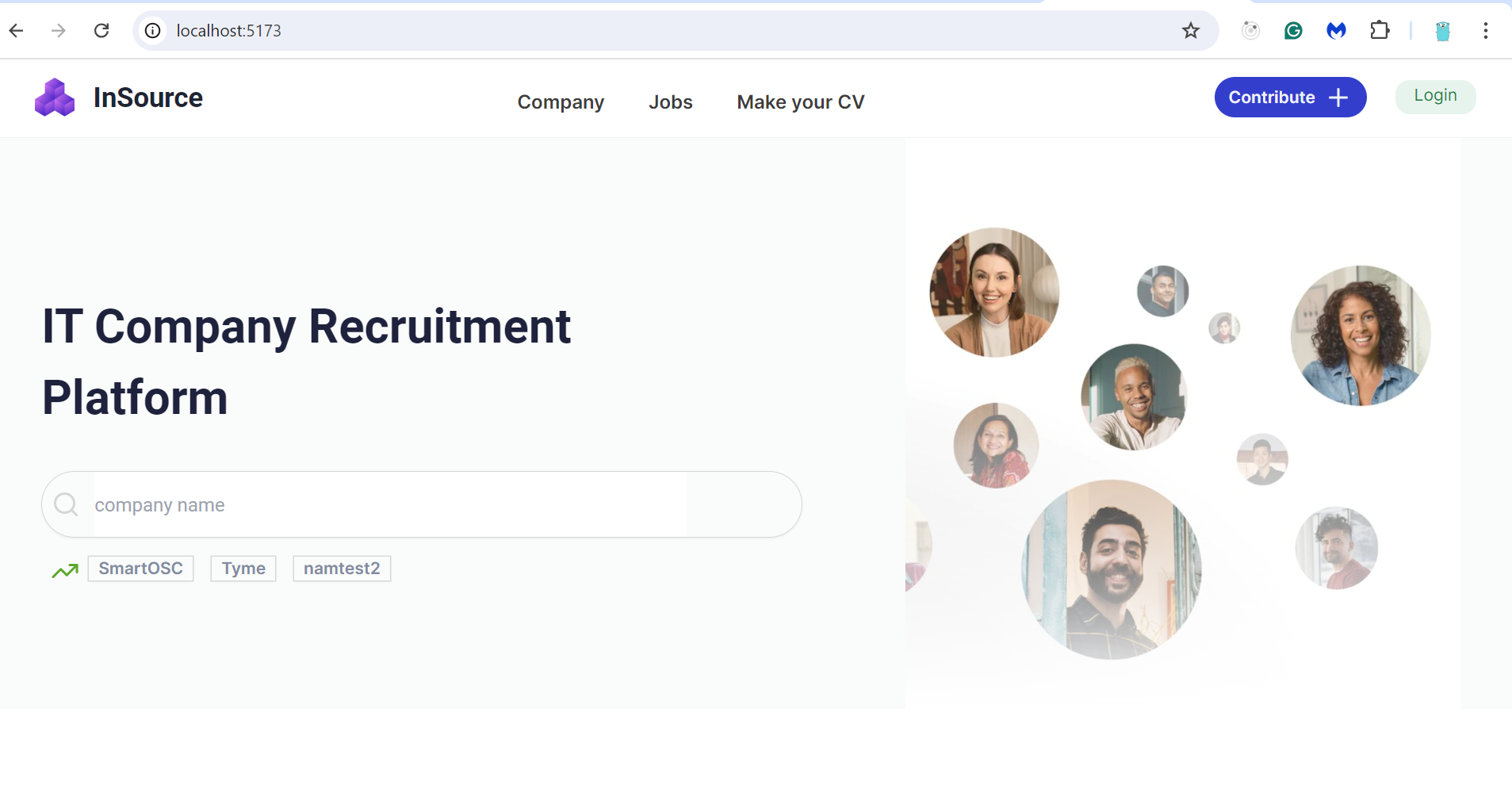


Sau khi hiện thực các bước ở trên, chúng ta sẽ có được một ứng dụng có sự kết hợp cơ bản của Microservice và GraphQL. Chúng ta thực hiện testing trước tiên trên Playground của GraphQL và được các kết quả như mong muốn.



Hình 4.3: Playground testing

Việc gọi API từ phía client thông qua Apollo cũng cho thấy kết quả hoàn toàn khả quan. Đây là trang landing page của hệ thống:



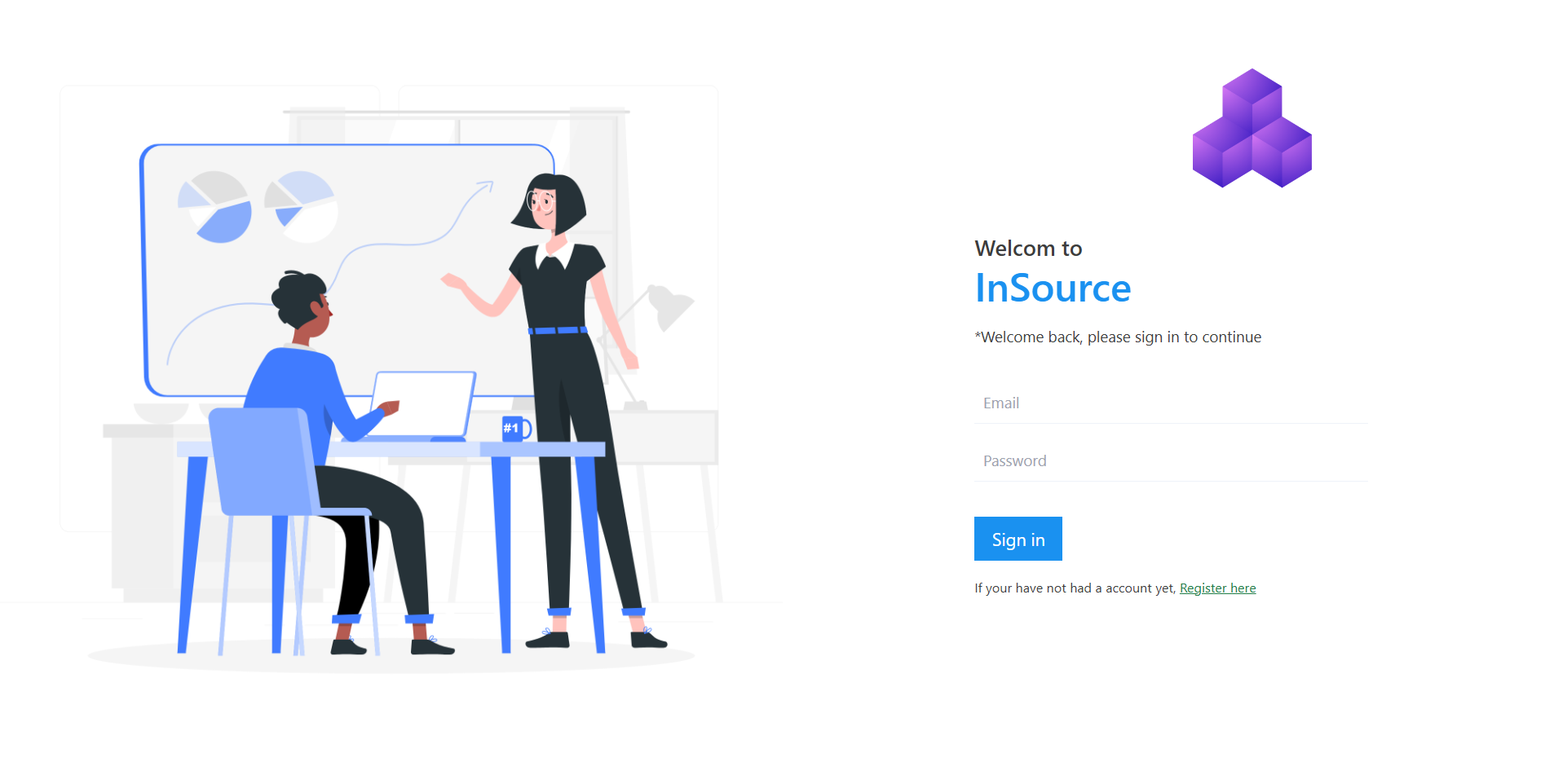
Hình 4.4: Client call testing

### Mutation

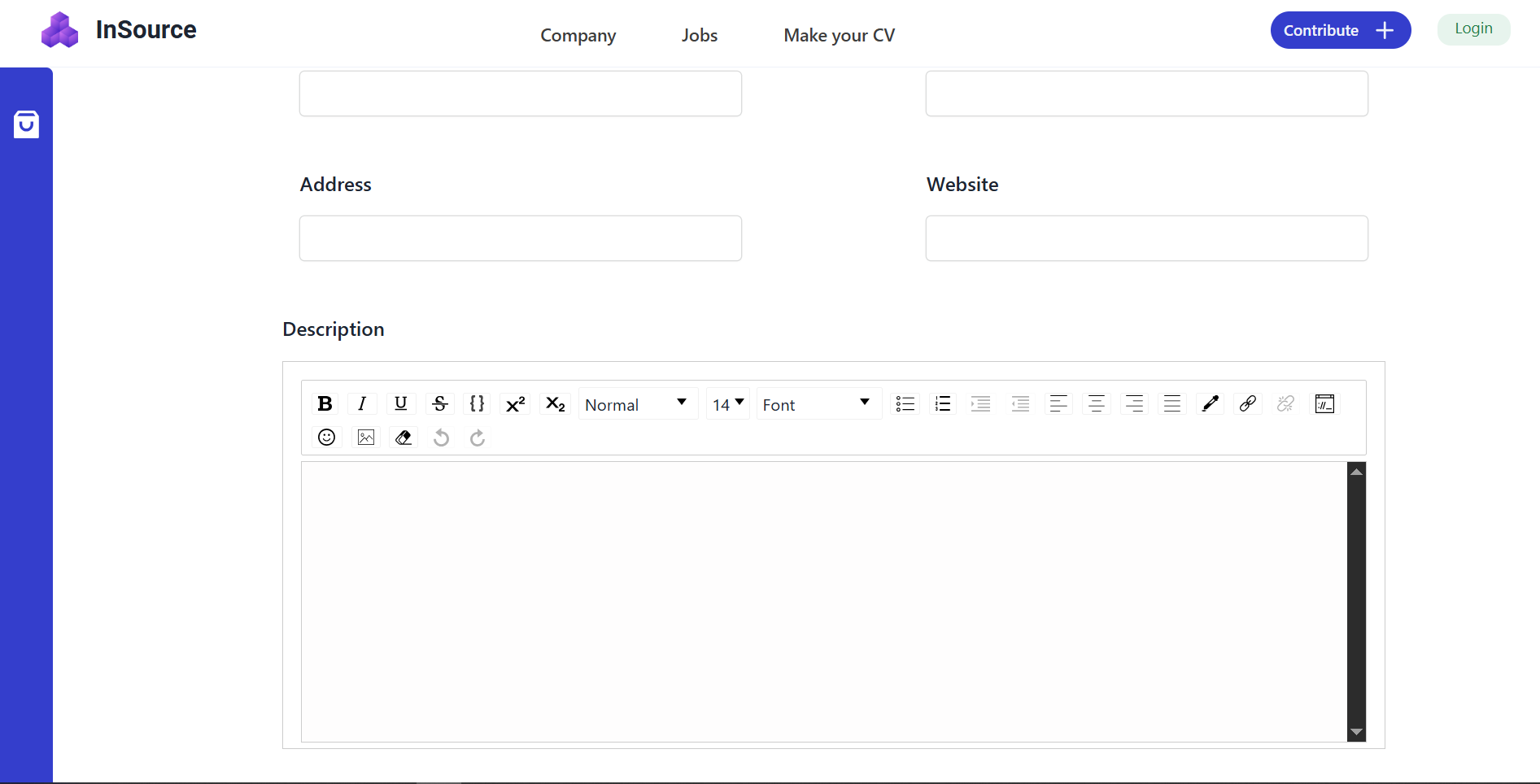
Mutation được dùng để thực hiện các thao tác thay đổi dữ liệu, như thêm mới, cập nhật, hoặc xóa dữ liệu. Mutation tương đương với các lệnh POST, PUT, và DELETE trong REST API.

Lấy ví dụ với trang login và tạo company:



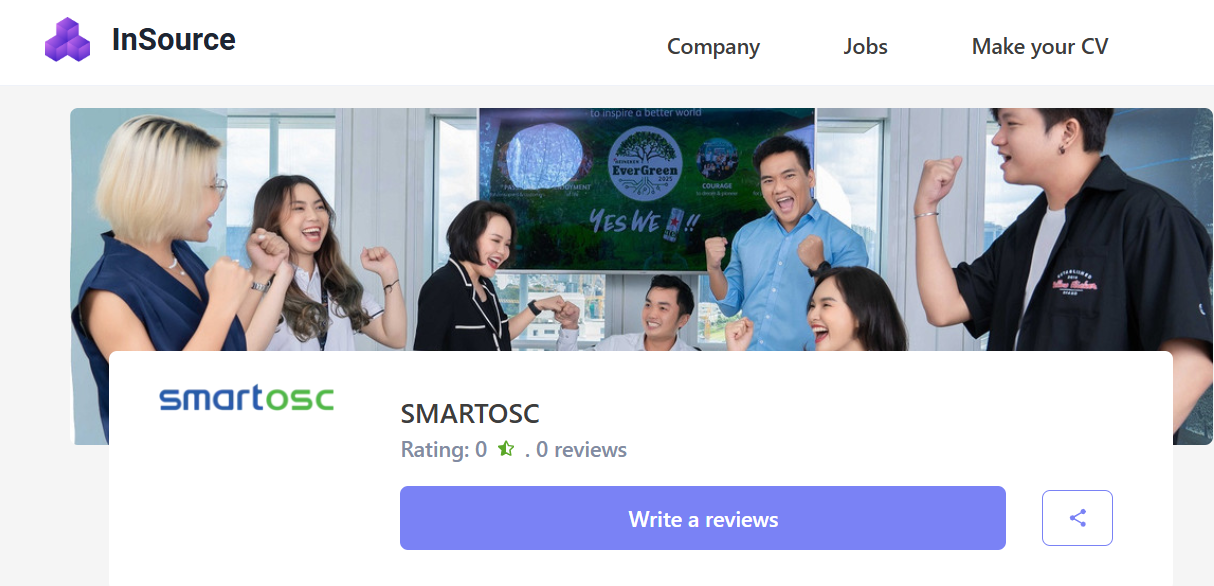


Hình 4.5: Login page



Hình 4.6:Create company page

Sau khi tạo mới một công ty, người dùng có thể truy cạp vào trang web của công ty để xem các thông tin:



Hình 4.7: Company overview

Sau đó, người dùng có thể viết 1 đánh giá cho công ty dựa trên trải nghiệm khách quan của mình:



Hình 4.8: Đánh giá công ty

## Lưu trữ ảnh với Cloudinary

Cloudinary là một dịch vụ quản lý hình ảnh và video trên đám mây, cung cấp các tính năng như lưu trữ, quản lý, tối ưu hóa và phân phối nội dung đa phương tiện. Cloudinary hỗ trợ nhiều phương thức tích hợp cho cả frontend và backend, giúp đơn giản hóa quy trình xử lý hình ảnh trong các ứng dụng web.

Trong dự án này, quy trình lưu trữ ảnh được thực hiện như sau:

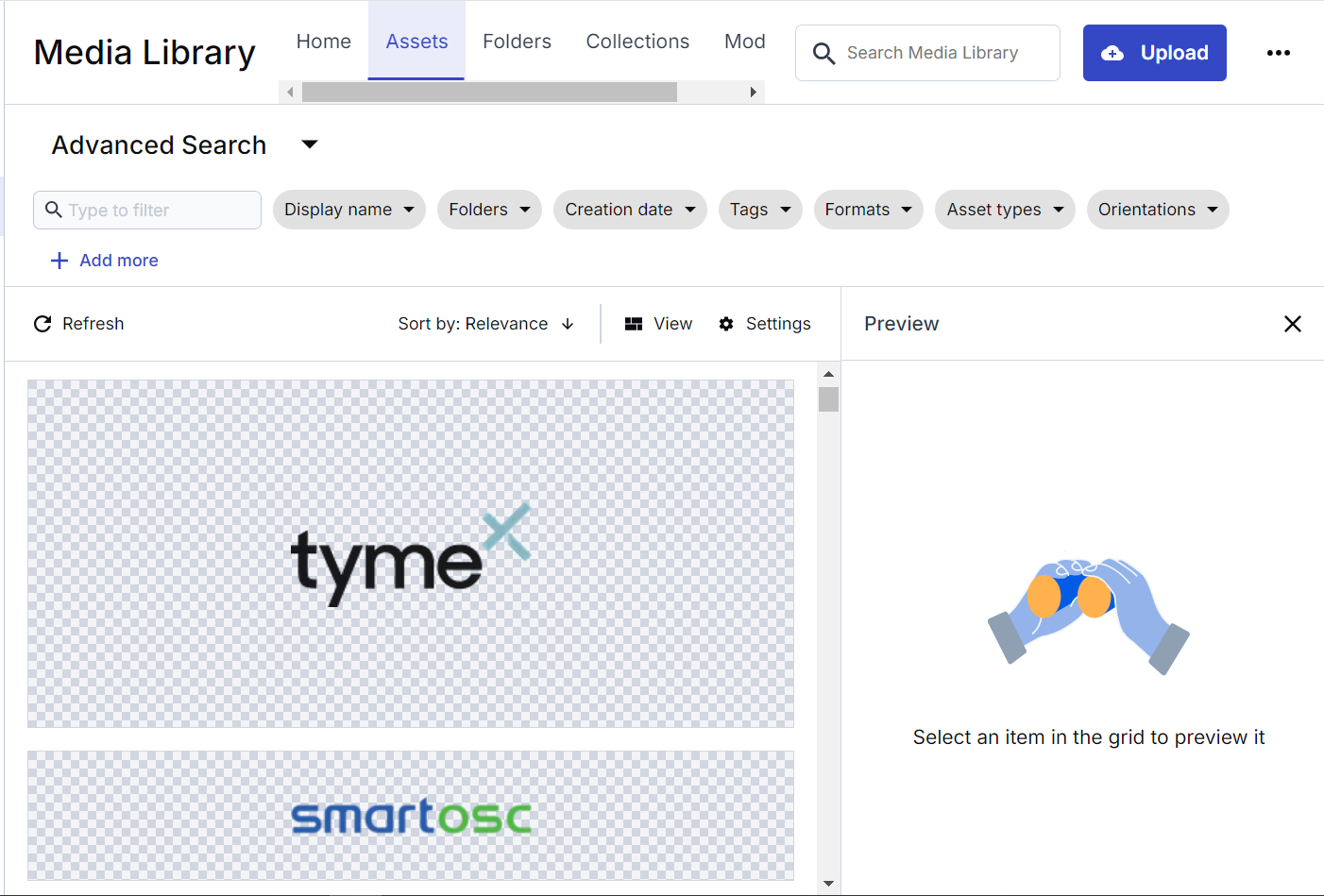
* **Upload ảnh từ phía client lên Cloudinary:** Người dùng tải ảnh trực tiếp từ giao diện frontend lên Cloudinary.
* **Nhận về URL của ảnh từ Cloudinary:** Sau khi ảnh được tải lên thành công, Cloudinary trả về một URL duy nhất cho ảnh đã được lưu trữ.
* **Lưu URL vào cơ sở dữ liệu:** URL nhận được sẽ được lưu trữ vào cơ sở dữ liệu của hệ thống, để sử dụng cho việc hiển thị ảnh sau này.

Phương pháp này giúp giảm tải cho server backend, vì toàn bộ quá trình upload và xử lý ảnh đều diễn ra trên Cloudinary.

Qúa trình tích hợp có nhiều bước, nhung chúng ta có thể tóm gọn trong đoạn code sau:



Sau khi tích hợp thành công, chúng ta có thể xem được các ảnh đã tải lên trên cloudinary:



Hình 4.9: Cloudinary console

# KẾT QUẢ VÀ HƯỚNG PHÁT TRIỂN

## Tổng kết

Qua quá trình thực hiện dự án, chúng ta đã triển khai được một hệ thống microservices với kiến trúc GraphQL, kết hợp các công nghệ hiện đại như NestJS và ReactJS. Hệ thống không chỉ giải quyết được các bài toán liên quan đến quản lý dữ liệu mà còn tạo nền tảng cho những tính năng mở rộng trong tương lai, như tích hợp AI,..

### Ưu điểm

* Kiến trúc linh hoạt: Việc áp dụng microservices giúp hệ thống dễ dàng mở rộng và bảo trì, mỗi dịch vụ có thể được phát triển và triển khai độc lập.
* Tối ưu hóa hiệu suất với GraphQL: GraphQL cho phép truy vấn dữ liệu một cách tối ưu, chỉ lấy những thông tin cần thiết, giúp giảm tải cho hệ thống.
* Khả năng mở rộng trong tương lai: Với nền tảng microservices, hệ thống có khả năng tích hợp thêm nhiều tính năng mới mà không ảnh hưởng đến các thành phần hiện tại.

### Nhược điểm

* Độ phức tạp trong triển khai: Việc quản lý nhiều dịch vụ nhỏ và kết nối giữa chúng đòi hỏi công tác triển khai và bảo trì phức tạp hơn so với kiến trúc đơn thể.
* Khả năng đồng bộ dữ liệu: Việc chia nhỏ cơ sở dữ liệu theo từng dịch vụ có thể dẫn đến các vấn đề đồng bộ và nhất quán dữ liệu.

## Hướng phát triển

Như đã đề cập ở phần trên, trong phạm vi môn học, dự án hiện tại vẫn chỉ là một phần của khóa luận tốt nghiệp. Trong tương lai, chúng tôi sẽ tiếp tục hoàn thiện các dịch vụ của hệ thống, đồng thời mở rộng và triển khai những tính năng nâng cao như trí tuệ nhân tạo (AI) và tạo CV theo đúng mục tiêu ban đầu.

Cụ thể, kế hoạch phát triển sắp tới bao gồm:

1. **Hoàn thiện các dịch vụ của hệ thống**: Tối ưu hóa và mở rộng các microservices hiện có, đảm bảo tính ổn định và khả năng mở rộng khi hệ thống đối mặt với lượng người dùng lớn hơn.
2. **Triển khai AI**: Áp dụng các mô hình AI nhằm phân tích và dự đoán dựa trên dữ liệu người dùng, từ đó cung cấp các gợi ý thông minh và tối ưu hóa trải nghiệm người dùng.
3. **CI/CD**: Triển khai quy trình tích hợp liên tục và phân phối liên tục (CI/CD) nhằm tự động hóa việc xây dựng, kiểm thử và triển khai ứng dụng. Điều này sẽ giúp tăng tốc độ phát triển, đồng thời đảm bảo chất lượng mã nguồn và hiệu suất của hệ thống.

Những định hướng phát triển này sẽ giúp hệ thống trở nên hoàn thiện và mạnh mẽ hơn, đáp ứng nhu cầu sử dụng trong môi trường thực tế.

Hết.

**TÀI LIỆU THAM KHẢO**

