HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA ĐÀO TẠO SAU ĐẠI HỌC
----o0o----

BÁO CÁO HỌC PHẦN
CÁC HỆ THỐNG PHÂN TÁN

Chủ đề: MICROSERVICE

(Nhóm 10)

CHUYÊN NGÀNH: **HỆ THỐNG THÔNG TIN**
GIẢNG VIÊN: **TS. KIM NGỌC BÁCH**
NHÓM HỌC VIÊN THỰC HIỆN:

PHẠM QUANG MINH	**B25CHHT041**
ĐỖ HOÀNG LONG	**B25CHHT037**
NGUYỄN HOÀNG LONG	**B25CHHT035**

LỚP: M25CQIS02

HÀ NỘI, THÁNG 12 NĂM 2025

MỤC LỤC

CHƯƠNG I: TỔNG QUAN HỆ THỐNG 1

1.1. Giới thiệu chung về hệ thống 1

1.2. Mục tiêu xây dựng hệ thống 1

CHƯƠNG II: PHÂN TÍCH YÊU CẦU HỆ THỐNG 2

2.1. Mục tiêu nghiệp vụ 2

2.2. Các chức năng cụ thể sẽ được triển khai 3

CHƯƠNG III: THIẾT KẾ PHẦN MỀM 6

3.1. Nguyên tắc và định hướng thiết kế 6

3.2. Thiết kế kiến trúc tổng thể hệ thống 6

3.3. Thiết kế các Service nghiệp vụ (Microservices) 7

3.4. Thiết kế API Gateway 8

3.5. Thiết kế Eureka Server (Service Discovery) 8

3.6. Thiết kế giao diện người dùng (Frontend Service) 9

3.7. Thiết kế cơ sở dữ liệu 9

3.8. Thiết kế triển khai và hạ tầng 9

3.9. Tổng kết chương 10

CHƯƠNG IV: TRIỂN KHAI VÀ CHẠY DEMO HỆ THỐNG 11

4.1. Mục tiêu triển khai hệ thống 11

4.2. Môi trường và công cụ triển khai 11

4.3. Triển khai hạ tầng hệ thống 11

4.4. Kịch bản chạy demo hệ thống 14

CHƯƠNG V: PHÂN TÍCH SOURCE CODE 18

5.1. Tổng quan cấu trúc source code hệ thống 18

5.2. Phân tích source code Eureka Server 18

5.3. Phân tích source code API Gateway 18

5.4. Phân tích source code StudentService 19

5.5. Phân tích source code Room Service 20
5.6. Phân tích source code RentRoom Service 20
5.7. Phân tích source code Frontend Service 21
5.8. Phân tích file cấu hình và triển khai 21
5.9. Đánh giá chất lượng source code 21
5.10. Tổng kết chương 22
CHƯƠNG VI: ĐÁNH GIÁ VÀ KẾT LUẬN 23
6.1. Đánh giá hệ thống 23
6.2. Ưu điểm và hạn chế 23
6.3. Hướng phát triển 23
6.4. Kết luận 23

CHƯƠNG I: TỔNG QUAN HỆ THỐNG

1.1. Giới thiệu chung về hệ thống

Trong bối cảnh chuyển đổi số diễn ra mạnh mẽ, các hệ thống thông tin ngày càng có xu hướng mở rộng về quy mô, số lượng người dùng và mức độ phức tạp trong xử lý nghiệp vụ. Mô hình kiến trúc phần mềm truyền thống (Monolithic) dần bộc lộ nhiều hạn chế như khó mở rộng, khó bảo trì, phụ thuộc chặt chẽ giữa các thành phần và rủi ro cao khi nâng cấp hệ thống.

Để khắc phục những hạn chế trên, kiến trúc **Microservices** đã ra đời và trở thành xu hướng phát triển chủ đạo trong các hệ thống phần mềm hiện đại. Kiến trúc này cho phép chia hệ thống lớn thành nhiều dịch vụ nhỏ, độc lập, mỗi dịch vụ đảm nhiệm một chức năng nghiệp vụ riêng biệt và có khả năng triển khai, mở rộng độc lập.

Xuất phát từ yêu cầu đó, hệ thống **Quản lý thuê phòng - ThuePhongService** được xây dựng nhằm mô phỏng một hệ thống quản lý thuê phòng theo kiến trúc Microservices, ứng dụng các công nghệ phổ biến hiện nay như **Spring Boot, Spring Cloud, Eureka Server, API Gateway, OpenFeign và Docker**. Hệ thống không chỉ giải quyết bài toán nghiệp vụ quản lý thuê phòng mà còn đóng vai trò là mô hình minh họa cho việc thiết kế và triển khai kiến trúc Microservices trong thực tế.

1.2. Mục tiêu xây dựng hệ thống

Mục tiêu tổng quát của hệ thống là xây dựng một ứng dụng quản lý thuê phòng theo kiến trúc Microservices, đảm bảo tính **tách biệt nghiệp vụ, khả năng mở rộng, dễ bảo trì** và phù hợp với các tiêu chuẩn phát triển phần mềm hiện đại.

Hệ thống cho phép quản lý thông tin sinh viên, thông tin phòng và hợp đồng thuê phòng thông qua các dịch vụ độc lập, có khả năng giao tiếp với nhau thông qua giao thức HTTP và cơ chế Service Discovery.

CHƯƠNG II: PHÂN TÍCH YÊU CẦU HỆ THỐNG

2.1. Mục tiêu nghiệp vụ

2.1.1. Bài toán nghiệp vụ đặt ra

Trong thực tế, hoạt động quản lý thuê phòng (đặc biệt trong ký túc xá, nhà trọ sinh viên hoặc khu lưu trú tập trung) thường liên quan đến nhiều nhóm thông tin khác nhau như: thông tin người thuê, thông tin phòng và thông tin hợp đồng.

Nếu các thông tin này được quản lý tập trung trong một hệ thống lớn, việc mở rộng, thay đổi hoặc bảo trì hệ thống sẽ gặp nhiều khó khăn.

Bài toán đặt ra là xây dựng một hệ thống quản lý thuê phòng có khả năng:

- Quản lý độc lập từng nhóm dữ liệu nghiệp vụ.
- Đảm bảo tính nhất quán khi các nghiệp vụ có sự liên kết với nhau.
- Dễ dàng mở rộng, nâng cấp mà không ảnh hưởng toàn bộ hệ thống.

2.1.2. Mục tiêu nghiệp vụ tổng quát

Hệ thống **ThuePhongService** được xây dựng nhằm đáp ứng các mục tiêu nghiệp vụ tổng quát sau:

- Tin học hóa quy trình quản lý thuê phòng.
- Phân tách rõ ràng các nghiệp vụ chính trong hệ thống.
- Đảm bảo các nghiệp vụ có thể hoạt động độc lập nhưng vẫn phối hợp chặt chẽ.
- Giảm sự phụ thuộc giữa các thành phần trong hệ thống.
- Nâng cao khả năng mở rộng và bảo trì trong tương lai.

2.1.3. Mục tiêu nghiệp vụ cụ thể

Trên cơ sở mục tiêu tổng quát, hệ thống hướng tới các mục tiêu nghiệp vụ cụ thể như sau:

a, Quản lý thông tin sinh viên:

Hệ thống cho phép lưu trữ và quản lý thông tin sinh viên thuê phòng, đồng thời hỗ trợ truy xuất thông tin sinh viên một cách nhanh chóng và chính xác nhằm phục vụ cho các nghiệp vụ liên quan.

Đặc điểm: Chức năng này được triển khai thông qua **StudentService**, là một microservice hoạt động độc lập, sử dụng cơ sở dữ liệu riêng (StudentDB) và không truy cập trực tiếp vào dữ liệu của các service khác.

b, Quản lý thông tin phòng:

Hệ thống cho phép quản lý danh sách các phòng cho thuê, theo dõi các thông tin cơ bản của phòng (mã phòng, loại phòng và tình trạng phòng).

Đặc điểm: Chức năng này được đảm nhiệm bởi **RoomService**, với cơ sở dữ liệu riêng (RoomDB). Dữ liệu phòng được quản lý độc lập với dữ liệu sinh

viên và được cung cấp thông qua các API để các service khác (đặc biệt là RentRoom Service) có thể truy vấn khi cần thiết.

c, Quản lý hợp đồng thuê phòng:

Hệ thống hỗ trợ lập và quản lý hợp đồng thuê phòng giữa sinh viên và phòng. Việc tạo hợp đồng chỉ được thực hiện khi sinh viên và phòng tồn tại hợp lệ trong hệ thống.

Đặc điểm: Chức năng này được triển khai thông qua **RentRoomService**, sử dụng cơ chế giao tiếp liên dịch vụ (OpenFeign) để xác minh thông tin sinh viên và phòng. **RentRoomService** không lưu trữ thông tin chi tiết của sinh viên và phòng, mà chỉ lưu các mã tham chiếu (ID), đồng thời sử dụng cơ sở dữ liệu riêng (RentRoomDB) để quản lý dữ liệu hợp đồng.

d, Đảm bảo tính phân tán của hệ thống:

Hệ thống được thiết kế theo kiến trúc Microservices, trong đó mỗi nghiệp vụ được triển khai dưới dạng một dịch vụ độc lập, có khả năng triển khai, mở rộng và bảo trì riêng biệt. Dữ liệu của mỗi dịch vụ được quản lý độc lập, góp phần giảm sự phụ thuộc giữa các thành phần và nâng cao tính linh hoạt của hệ thống.

2.2. Các chức năng cụ thể sẽ được triển khai

Dựa trên phân tích mục tiêu nghiệp vụ, hệ thống **ThuePhongService** được thiết kế với các chức năng cụ thể, tương ứng với từng microservice trong hệ thống.

2.2.1. Nhóm chức năng quản lý hợp đồng thuê phòng

Nhóm chức năng quản lý hợp đồng thuê phòng là nhóm chức năng nghiệp vụ trung tâm của hệ thống **ThuePhongService**, phản ánh trực tiếp mục tiêu chính của bài toán quản lý thuê phòng. Nhóm chức năng này được triển khai thông qua sự phối hợp giữa **Frontend Service**, **API Gateway** và các microservice nghiệp vụ phía backend, trong đó **RentRoomService** giữ vai trò xử lý nghiệp vụ chính. Các chức năng cụ thể của nhóm chức năng này bao gồm:

a, Hiển thị danh sách các hợp đồng đã tạo.

Hệ thống cho phép người dùng xem danh sách toàn bộ các hợp đồng thuê phòng đã được lập trước đó. Dữ liệu hợp đồng được truy xuất từ cơ sở dữ liệu riêng của RentRoom Service và hiển thị thông qua giao diện người dùng. Việc hiển thị này giúp người dùng theo dõi, quản lý và kiểm soát tình trạng các hợp đồng thuê phòng trong hệ thống.

b, Cho phép tạo hợp đồng thuê phòng mới.

Hệ thống cung cấp chức năng tạo mới hợp đồng thuê phòng thông qua giao diện người dùng. Người dùng nhập các thông tin cần thiết như mã sinh viên, mã phòng và thời gian thuê. Yêu cầu tạo hợp đồng được gửi từ Frontend qua API Gateway đến **RentRoomService** để xử lý nghiệp vụ.

c, Kiểm tra tính hợp lệ của sinh viên và phòng trước khi tạo hợp đồng.

Để đảm bảo tính chính xác và nhất quán của dữ liệu, hệ thống chỉ cho phép tạo hợp đồng khi sinh viên và phòng thuê thực sự tồn tại trong hệ thống. **RentRoomService** sử dụng cơ chế giao tiếp liên dịch vụ (OpenFeign) để:

- Gọi sang **StudentService** kiểm tra sự tồn tại của sinh viên.
- Gọi sang **RoomService** kiểm tra sự tồn tại của phòng.

Chỉ khi cả hai điều kiện trên đều được thỏa mãn, hợp đồng thuê phòng mới được tạo và lưu trữ vào cơ sở dữ liệu riêng của **RentRoomService**. Trường hợp một trong hai thông tin không hợp lệ, hệ thống sẽ từ chối tạo hợp đồng và trả thông báo lỗi phù hợp cho người dùng.

Đặc điểm của nhóm chức năng này là thể hiện rõ nguyên lý kiến trúc Microservices: mỗi service quản lý dữ liệu của riêng mình, không truy cập trực tiếp cơ sở dữ liệu của service khác, mà chỉ trao đổi thông tin thông qua các API. Cách thiết kế này giúp hệ thống đảm bảo tính độc lập, dễ mở rộng và dễ bảo trì trong tương lai.

2.2.2. Nhóm chức năng định tuyến và điều phối (API Gateway)

API Gateway đóng vai trò là cổng giao tiếp duy nhất giữa hệ thống và người dùng.

Các chức năng chính gồm:

- Nhận tất cả các request từ client.
- Định tuyến request đến đúng microservice.
- Tra cứu địa chỉ service thông qua Eureka Server.
- Đóng vai trò trung gian trong việc kiểm soát truy cập hệ thống.

2.2.3. Nhóm chức năng đăng ký và phát hiện dịch vụ (Eureka Server)

Eureka Server đảm nhiệm chức năng quản lý các microservice trong hệ thống.

Các chức năng bao gồm:

- Cho phép các service đăng ký khi khởi động.
- Cập nhật trạng thái hoạt động của các service.
- Hỗ trợ Gateway và các service khác tìm kiếm địa chỉ service.

2.2.4. Nhóm chức năng giao diện người dùng (Frontend Service)

Frontend Service cung cấp giao diện người dùng nhằm minh họa và tương tác với hệ thống.

Các chức năng chính:

- Hiển thị danh sách hợp đồng thuê phòng.
- Cung cấp form tạo mới hợp đồng.
- Gửi request đến Gateway để thực hiện nghiệp vụ.
- Nhận và hiển thị kết quả phản hồi từ hệ thống.

Đặc điểm:

- Không truy cập trực tiếp cơ sở dữ liệu.
- Hoạt động như một client của hệ thống.

2.3. Tổng kết chương

Chương II đã trình bày chi tiết việc phân tích yêu cầu hệ thống **ThuePhongService**, bao gồm mục tiêu nghiệp vụ và các chức năng cụ thể sẽ được triển khai. Việc phân tách rõ ràng các chức năng theo từng microservice không chỉ giúp hệ thống hoạt động hiệu quả mà còn đảm bảo khả năng mở rộng, bảo trì và phát triển trong tương lai.

CHƯƠNG III: THIẾT KẾ PHẦN MỀM

3.1. Nguyên tắc và định hướng thiết kế

Hệ thống **ThuePhongService** được thiết kế dựa trên các nguyên tắc cốt lõi của kiến trúc Microservices, nhằm đảm bảo tính linh hoạt, khả năng mở rộng và dễ bảo trì trong quá trình phát triển và vận hành. Các nguyên tắc chính được áp dụng trong thiết kế phần mềm bao gồm:

- **Phân tách nghiệp vụ rõ ràng**: Mỗi microservice đảm nhiệm một nghiệp vụ cụ thể.
- **Triển khai độc lập**: Các service có thể được build, chạy và mở rộng độc lập.
- **Cơ sở dữ liệu độc lập**: Mỗi service sở hữu một database riêng, không chia sẻ trực tiếp dữ liệu.
- **Giao tiếp thông qua API**: Các service trao đổi dữ liệu bằng giao thức HTTP/REST.
- **Loose Coupling – High Cohesion**: Giảm sự phụ thuộc giữa các service, tăng tính gắn kết nội bộ.

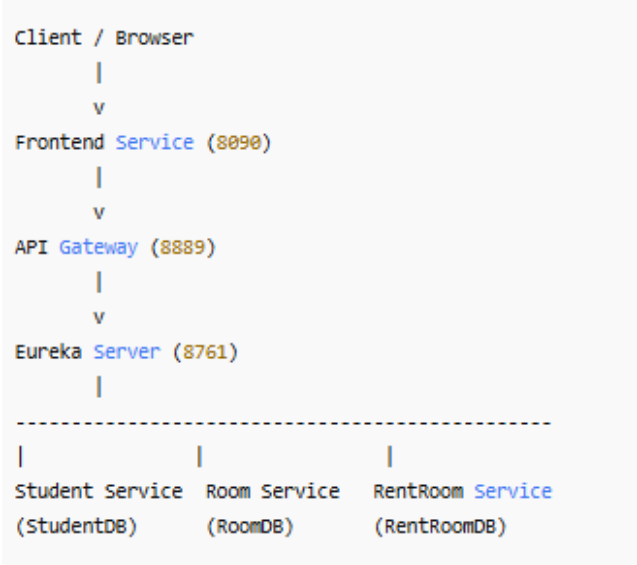
3.2. Thiết kế kiến trúc tổng thể hệ thống

3.2.1. Mô hình kiến trúc Microservices

Hệ thống được thiết kế theo mô hình **Microservices Architecture**, bao gồm các thành phần chính:

- Eureka Server
- API Gateway
- Các Microservices nghiệp vụ
- Frontend Service
- Hạ tầng triển khai (Docker)

  Mô hình kiến trúc tổng thể có thể biểu diễn như sau:



3.2.2. Vai trò của từng thành phần trong kiến trúc

- **Frontend Service**: Giao diện người dùng, tiếp nhận thao tác từ người sử dụng.
- **API Gateway**: Cổng giao tiếp duy nhất, định tuyến request.
- **Eureka Server**: Quản lý đăng ký và phát hiện dịch vụ.
- **StudentService / RoomService**: Quản lý dữ liệu gốc.
- **RentRoomService**: Xử lý nghiệp vụ tổng hợp (hợp đồng thuê phòng).

3.3. Thiết kế các Service nghiệp vụ (Microservices)

Hệ thống chia nhỏ thành các phần riêng biệt để dễ quản lý và sửa chữa.

a, StudentService & Room Service (Service Cơ bản)

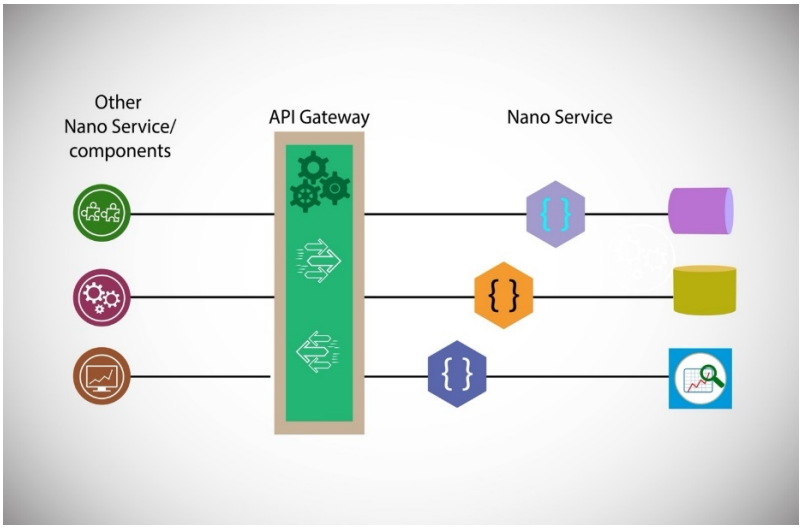
- **Chức năng:** Quản lý dữ liệu gốc (Master Data).
- **Nhiệm vụ:** Chỉ thực hiện các tác vụ CRUD (Thêm, Sửa, Xóa, Xem) đơn giản trên bảng `SinhVien` và `Phong`.
- **Đặc điểm:** Độc lập hoàn toàn, không phụ thuộc vào ai. Hỏng service này không ảnh hưởng service kia.

b, RentRoom Service (Service Nghiệp vụ/Composite)

- **Chức năng:** Đây là nơi xử lý logic phức tạp nhất - **Quản lý Hợp đồng**.
- **Nhiệm vụ:**
  - Kết nối dữ liệu: Một hợp đồng cần ID của Sinh viên và ID của Phòng.
  - **Giao tiếp (OpenFeign):** Service này đóng vai trò là "khách hàng". Nó sử dụng thư viện **OpenFeign** để gọi sang `student-service` (check xem sinh viên có thật không) và `room-service` (check phòng có trống không) trước khi tạo hợp đồng.

- **Tại sao quan trọng?** Nó chứng minh được sự tương tác giữa các service trong kiến trúc Microservices (Service này gọi Service kia).

3.4. Thiết kế API Gateway



API Gateway được thiết kế như một lớp trung gian giữa client và các microservices. **Ví dụ đời thường:** Đây là **Lễ tân** hoặc **Bảo vệ** tòa nhà. Khách (Frontend) muốn gặp ai thì phải qua lễ tân, lễ tân sẽ chỉ đường đến đúng phòng.

Chức năng:

- **Cổng duy nhất (Single Entry Point):** Frontend không cần biết bên trong có bao nhiêu service, cổng nào (8084, 8085...). Frontend chỉ cần nhớ đúng một địa chỉ duy nhất: `localhost:8889`.
- **Định tuyến (Routing):**
  - Thấy đường dẫn `/api/sinhvien/...` --> Chuyển sang `student-service`.
  - Thấy đường dẫn `/api/phong/...` --> Chuyển sang `room-service`.
- **Bảo mật (Security):** Kiểm tra xem người dùng có Token đăng nhập chưa (`AuthFilter`). Nếu chưa, chặn ngay tại cổng, không cho vào trong làm phiền các service.

3.5. Thiết kế Eureka Server (Service Discovery)

Eureka Server được thiết kế để giải quyết bài toán động địa chỉ trong hệ thống phân tán. **Ví dụ đời thường:** Đây chính là cuốn **Danh bạ điện thoại vàng** hoặc **Bác tổ trưởng dân phố**.

- **Vấn đề:** Trong hệ thống phân tán, các service (Student, Room...) có thể đổi địa chỉ IP hoặc Port liên tục (lúc thì 8084, lúc thì 8085...). Gateway không thể nhớ cứng hết được.
- **Chức năng:**
    - **Đăng ký (Registration):** Khi `student-service` khởi động, nó sẽ hét lên với Eureka: *"Tôi là STUDENT-SERVICE, tôi đang ở địa chỉ 192.168.1.5, cổng 8084!"*. Eureka sẽ ghi vào sổ.
    - **Khám phá (Discovery):** Khi Gateway hoặc RentRoom cần tìm Student, nó sẽ hỏi Eureka: *"Ai là STUDENT-SERVICE?"*, Eureka sẽ trả về địa chỉ IP/Port chính xác.

3.6. Thiết kế giao diện người dùng (Frontend Service)

Công nghệ: Spring Boot + Thymeleaf (Server-side rendering).

Chức năng:

- s

3.7. Thiết kế cơ sở dữ liệu

- Mỗi service có một CSDL riêng.
- Không join bảng giữa các service.
- Chỉ liên kết bằng ID logic.

Service	**Database**
StudentService	StudentDB
RoomService	RoomDB
RentRoomService	RentRoomDB

3.8. Thiết kế triển khai và hạ tầng

Hệ thống sử dụng **Docker & Docker Compose** để triển khai hạ tầng.

Ví dụ đời thường: Hãy tưởng tượng Docker giống như những chiếc **thùng container** chở hàng trên tàu biển. Dù bên trong là xe hơi, lúa gạo hay linh kiện điện tử, thì bên ngoài chúng đều là thùng sắt vuông vức giống hệt nhau. Cần cẩu nào cũng gắp được, tàu nào cũng chở được.

- **Chức năng kỹ thuật:** Docker giúp đóng gói code của bạn + môi trường chạy (Java JDK) + thư viện thành một khối thống nhất gọi là **Container**.
- **Tại sao dùng trong dự án này?**
    - Để chạy Database (PostgreSQL) nhanh chóng mà không cần cài đặt thủ công phức tạp vào máy.

- Đảm bảo code chạy trên máy bạn (Dev) cũng y hệt như chạy trên máy giảng viên hoặc Server thật (Production), tránh lỗi "máy tôi chạy được mà máy thầy không chạy".

3.9. Tổng kết chương

Chương III đã trình bày chi tiết thiết kế phần mềm của hệ thống **ThuePhongService**, từ kiến trúc tổng thể đến thiết kế từng microservice, cơ chế giao tiếp, cơ sở dữ liệu và hạ tầng triển khai. Thiết kế này đảm bảo hệ thống tuân thủ đúng kiến trúc Microservices, đáp ứng tốt các yêu cầu nghiệp vụ đã phân tích ở Chương II và tạo nền tảng vững chắc cho việc triển khai, mở rộng trong tương lai.

CHƯƠNG IV: TRIỂN KHAI VÀ CHẠY DEMO HỆ THỐNG

4.1. Mục tiêu triển khai hệ thống

Chương này nhằm trình bày quá trình triển khai thực tế hệ thống **ThuePhongService**, bao gồm việc chuẩn bị môi trường, cài đặt hạ tầng, build source code, khởi chạy các microservice và thực hiện kịch bản demo để kiểm chứng tính đúng đắn của hệ thống.

Thông qua chương này, hệ thống sẽ được chứng minh:

- Có thể triển khai và vận hành trên môi trường thực tế
- Các microservice hoạt động độc lập nhưng phối hợp đúng thiết kế
- Luồng xử lý nghiệp vụ hoạt động xuyên suốt từ giao diện người dùng đến cơ sở dữ liệu

4.2. Môi trường và công cụ triển khai

4.2.1. Môi trường phần cứng

- Máy tính cá nhân (PC/Laptop)
- Hệ điều hành: Windows / Linux / macOS

4.2.2. Môi trường phần mềm

Hệ thống được triển khai trong môi trường phát triển (Development) với các công cụ sau:

Thành phần	Phiên bản / Công cụ
Java Development Kit	JDK 17
Maven	Apache Maven
Framework	Spring Boot 3, Spring Cloud
Database	PostgreSQL
Container	Docker, Docker Compose
IDE	IntelliJ IDEA
Trình duyệt	Google Chrome

4.3. Triển khai hạ tầng hệ thống

Bước 1: Chuẩn bị môi trường (Prerequisites)

Đảm bảo máy bạn đã cài đủ các công cụ sau:

- **Java JDK 17** (Do project dùng Spring Boot 3).
- **Maven** (Để build và quản lý thư viện).

- **PostgreSQL** (Đã cài đặt và đang chạy, hoặc chạy qua Docker).
- **IntelliJ IDEA** (Khuyên dùng để chạy dashboard cho tiện).

Bước 2: Khởi tạo Database (Infrastructure)

Bạn cần tạo sẵn 3 Database rỗng trong PostgreSQL để các Service kết nối vào.

1. Mở **pgAdmin** hoặc **DBeaver** (hoặc dùng dòng lệnh psql).
2. Tạo lần lượt 3 Database với tên chính xác như sau (khớp với cấu hình trong `application.yml` của từng service):
   - `StudentDB`
   - `RoomDB`
   - `RentRoomDB` (hoặc `ContractDB` tùy lúc bạn cấu hình).
3. **Lưu ý:** Đảm bảo username/password kết nối DB trong code (thường là `postgres/password` hoặc `123456`) khớp với máy của bạn.

Bước 3: Khởi chạy Hạ tầng (Infrastructure) bằng Docker

Thay vì cài đặt và tạo Database thủ công từng cái, chúng ta sẽ dựng toàn bộ hạ tầng (PostgreSQL, v.v.) chỉ bằng một lệnh duy nhất.

1. Mở **Terminal** (hoặc CMD/PowerShell) tại **thư mục gốc** của dự án (nơi chứa file `docker-compose.yml`).
2. Chạy lệnh sau để dựng các container lên:
   Bash
   ```
   docker compose up -d
   ```
   (Tham số `-d` nghĩa là "detach", giúp Docker chạy ngầm, không chiếm dụng cửa sổ terminal).
3. **Kiểm tra:** Sau khi chạy xong, gõ lệnh sau để chắc chắn Database đã hoạt động:
   Bash
   ```
   docker ps
   ```
   (Nếu thấy tên container như `postgres` hoặc `thuephong-db` hiện ra ở cột STATUS là thành công).

Bước 4: Build code Java (Compile)

Tải thư viện và đóng gói toàn bộ dự án (bao gồm cả Frontend vì đã gộp module).

1. Mở Terminal tại thư mục gốc **ThuePhongService**.
2. Chạy lệnh:
   Bash
   ```
   mvn clean install -DskipTests
   ```

(Lệnh này sẽ build từ cha xuống con, bỏ qua test để tiết kiệm thời gian).

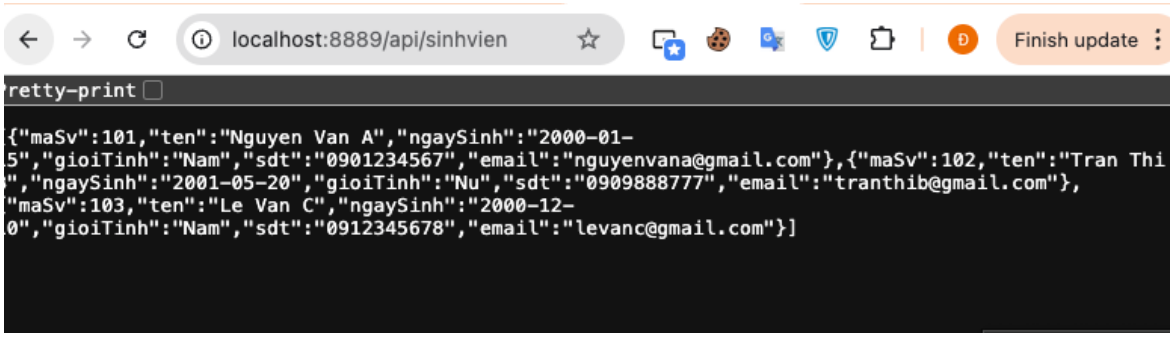
3. Chờ đến khi thấy thông báo **BUILD SUCCESS** màu xanh lá cây ở tất cả các module.

Bước 5: Khởi chạy hệ thống (Execution)

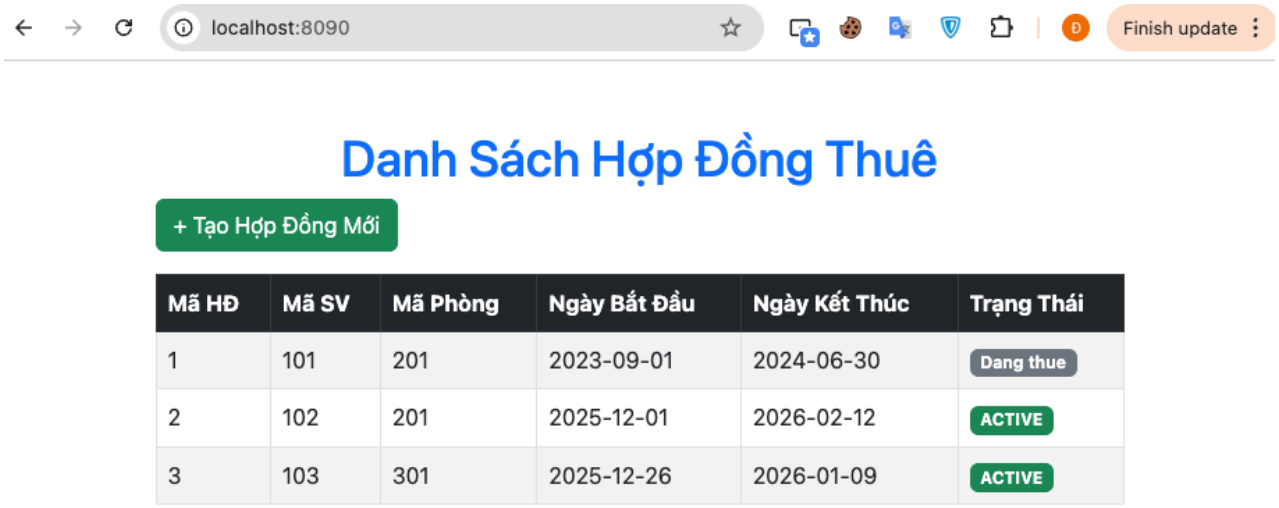
Quy tắc vàng: Phải chạy theo đúng thứ tự dưới đây để các service tìm thấy nhau.

1. **Chạy Eureka Server (Bộ não):**
   - Tìm file `EurekaServerApplication` -> Run.
   - Chờ báo `Started`. Kiểm tra tại: `http://localhost:8761`.
2. **Chạy các Service nghiệp vụ (Microservices):**
   - Chạy `StudentServiceApplication`.
   - Chạy `RoomServiceApplication`.
   - Chạy `RentRoomServiceApplication`.
   - *Check:* F5 lại trang Eureka, đảm bảo thấy đủ 3 service này hiện lên.
3. **Chạy Gateway (Cổng chính):**
   - Chạy `GatewayApplication`.
   - (Đây là cổng `8889` sẽ điều phối mọi request).

ví dụ: http://localhost:8889/api/sinhvien sẽ ra json là



4. **Chạy Frontend (Giao diện):**
   - Chạy `ThuePhongFEApplication`.
   - (Web chạy tại cổng `8090`). Link là http://localhost:8090/



4.4. Kịch bản chạy demo hệ thống

4.4.1. Mục tiêu demo

Demo nhằm chứng minh:

- Frontend không truy cập trực tiếp database
- Gateway định tuyến chính xác
- Các microservice giao tiếp đúng thiết kế
- Dữ liệu được lưu trữ đúng service

4.4.2. Các bước thực hiện demo

Bước 1: Truy cập giao diện

- Mở trình duyệt
- Truy cập: `http://localhost:8090`

Bước 2: Tạo hợp đồng thuê phòng

- Chọn chức năng "Tạo hợp đồng mới"
- Nhập dữ liệu:
    - Mã sinh viên: `1`
    - Mã phòng: `101`
    - Ngày bắt đầu: tùy chọn

Bước 3: Gửi yêu cầu

Nhấn nút "Lưu" Thực hiện các bước sau để chứng minh hệ thống hoạt động trước giảng viên:

1. Truy cập trình duyệt: **http://localhost:8090**

2. Chọn chức năng: **"Tạo Hợp Đồng Mới"**.
3. Nhập dữ liệu test (Giả sử DB Sinh viên và Phòng đã có dữ liệu mẫu):
    - **Mã SV:** `1`
    - **Mã Phòng:** `101`
    - **Ngày:** Chọn bất kỳ.
4. Bấm **Lưu**.
5. **Kết quả:**
    - Hệ thống quay về trang chủ và hiển thị dòng dữ liệu vừa thêm.
    - *Giải thích:* "Frontend (8090) đã gọi qua Gateway (8889), Gateway chuyển tiếp vào RentRoom Service. RentRoom Service dùng **Feign Client** xác thực với **StudentService** và Room Service trước khi lưu vào RentRoomDB."

4.4.3. Phân tích luồng xử lý demo

Giai đoạn 1: Tại Frontend (Văn phòng tiếp dân) - Cổng 8090

1. Người dùng bấm nút "Lưu Hợp Đồng":

- Bạn điền Form: `Mã SV: 1`, `Mã Phòng: 101`.
- Trình duyệt gửi một yêu cầu `POST` tới Controller của Frontend (file `WebController.java`).

2. Frontend đóng gói hồ sơ:

- Frontend (Java) nhận được dữ liệu. Nó không có quyền lưu trực tiếp.
- Nó dùng `RestTemplate` đóng gói dữ liệu thành **JSON**:
    - JSON
    - `{ "sinhVien": 1, "phong": 101, "ngayBatDau": "..." }`
- Nó gửi gói hàng này tới địa chỉ: **`http://localhost:8889/api/hopdong`** (Đây là địa chỉ của Gateway).

Giai đoạn 2: Tại Gateway (Bác bảo vệ chung) - Cổng 8889

3. Gateway chặn cửa:

- Gói tin đến cổng 8889. Gateway xem nhãn dán: `/api/hopdong`.
- Gateway mở bản đồ (cấu hình `routes`) ra xem: "À, cái đuôi `/hopdong` này là việc của thằng **RentRoom Service**".

4. Gateway hỏi đường Eureka:

- Gateway không biết `RentRoom Service` đang nằm ở đâu (IP nào, Port nào).
- Gateway gọi **Eureka Server**: "Alo, cho xin địa chỉ mới nhất của thằng `RENTROOM-SERVICE`".
- Eureka trả lời: "Nó đang ở `192.168.1.5:54321`".
- Gateway chuyển tiếp (Forward) gói JSON sang đúng địa chỉ đó.

Giai đoạn 3: Tại RentRoom Service (Trưởng phòng Hợp đồng)

5. Nhận việc:

`HopDongController` nhận lệnh `POST`. Nó chuyền ngay cho `HopDongService.add()`.

6. Quy trình xác minh (Đoạn quan trọng nhất - dùng Feign): Trước khi ký hợp đồng, `HopDongService` phải đi xác minh lý lịch (vì nó không giữ dữ liệu sinh viên/phòng).

- **Bước 6a (Gọi Sinh viên):**
  - Code chạy dòng: `sinhVienClient.getStudentById(1).`
  - **OpenFeign** âm thầm tạo một request gửi sang **StudentService**.
  - **StudentService** nhận lệnh -> Query `StudentDB` -> Trả về: *"Có sinh viên Nguyễn Văn A"*.
  - Nếu trả về `null` hoặc lỗi 404 -> `RentRoom Service` **HỦY KÈO** ngay lập tức, báo lỗi ra Frontend.
- **Bước 6b (Gọi Phòng):**
  - Code chạy dòng: `phongClient.getRoomById(101).`
  - **OpenFeign** gửi request sang **Room Service**.
  - **Room Service** nhận lệnh -> Query `RoomDB` -> Trả về: *"Có phòng VIP 101"*.

Giai đoạn 4: Lưu trữ & Phản hồi (Chốt đơn)

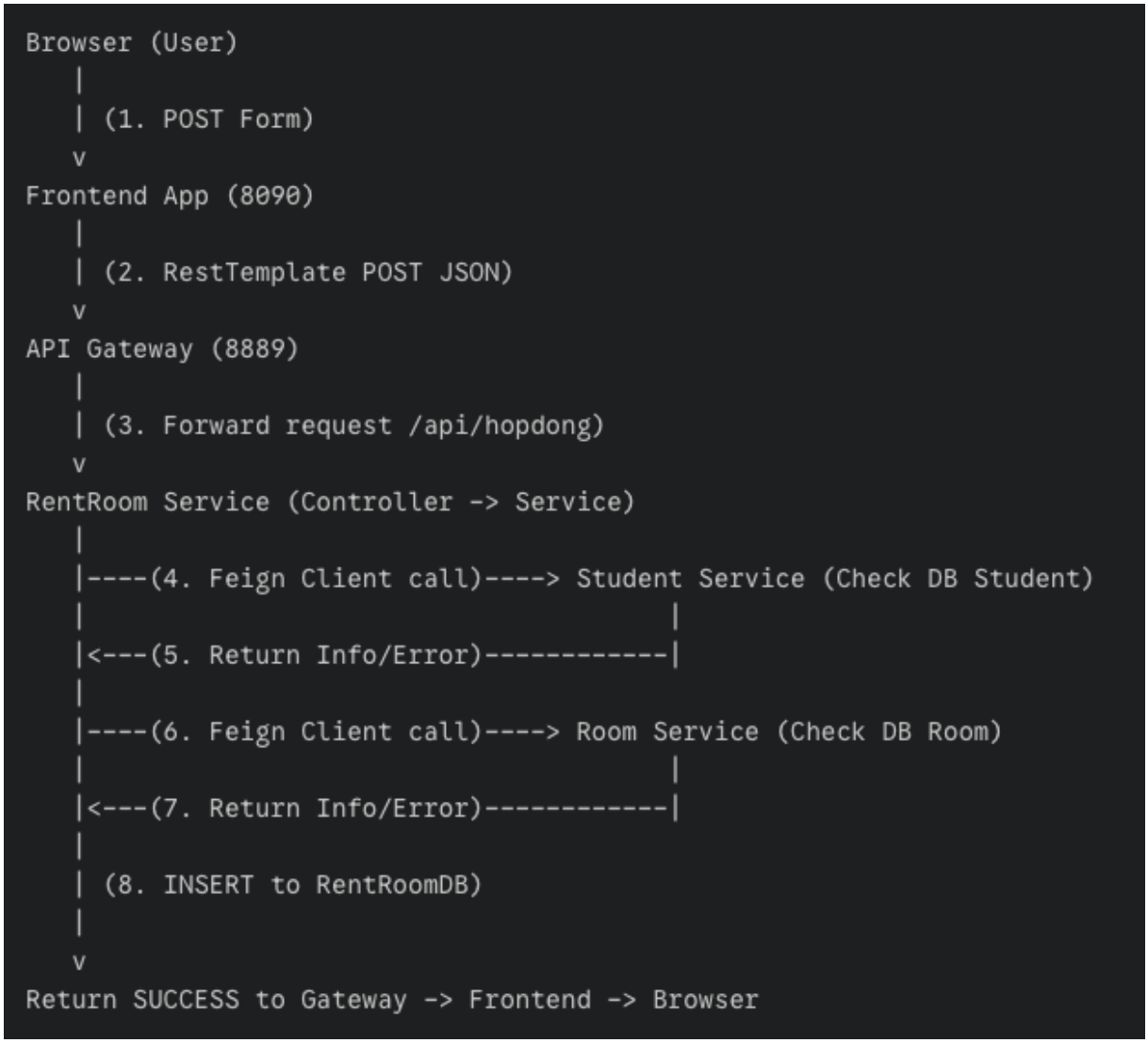
7. Lưu vào kho:

- Sau khi cả 2 bước xác minh trên đều OK.
- `HopDongService` gọi `Repository` -> Thực hiện lệnh SQL `INSERT INTO hop_dong ...` vào **RentRoomDB** (Database riêng của nó).

8. Báo cáo kết quả:

- RentRoom Service trả về: `HTTP 200 OK`.

- Gateway nhận tin -> Chuyền về Frontend.
- Frontend nhận tin -> Hiển thị thông báo "Thêm thành công" và load lại trang danh sách.

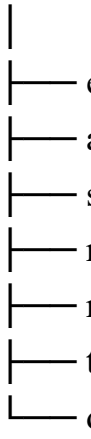


CHƯƠNG V: PHÂN TÍCH SOURCE CODE

5.1. Tổng quan cấu trúc source code hệ thống

Dự án **ThuePhongService** được tổ chức theo mô hình **multi-module Maven**, trong đó mỗi module tương ứng với một thành phần trong kiến trúc Microservices. Cách tổ chức này giúp phân tách rõ ràng các chức năng, thuận tiện cho việc phát triển, bảo trì và mở rộng hệ thống.

Cấu trúc tổng thể của source code như sau:

ThuePhongService

├── eureka-server
├── api-gateway
├── student-service
├── room-service
├── rentroom-service
├── thuephong-frontend
└── docker-compose.yml

Mỗi module là một ứng dụng Spring Boot độc lập, có thể được build và chạy riêng biệt.

5.2. Phân tích source code Eureka Server

5.2.1. Mục đích module

Module **Eureka Server** đóng vai trò là trung tâm đăng ký và phát hiện dịch vụ (Service Discovery) trong hệ thống.

5.2.2. Thành phần chính

- EurekaServerApplication.java
  - Lớp khởi động ứng dụng
  - Được đánh dấu bằng annotation @EnableEurekaServer

5.2.3. Đặc điểm source code

- Không chứa logic nghiệp vụ
- Chỉ cấu hình thông tin server, port và registry
- Đảm bảo các service khác có thể đăng ký và truy vấn

5.3. Phân tích source code API Gateway

5.3.1. Vai trò module

API Gateway là cổng giao tiếp duy nhất giữa client và các microservice.

5.3.2. Các thành phần chính

- GatewayApplication.java
- File cấu hình định tuyến application.yml

5.3.3. Cấu hình định tuyến

Gateway định tuyến request dựa trên path:

- /api/sinhvien/** → **StudentService**
- /api/phong/** → Room Service
- /api/hopdong/** → RentRoom Service

5.3.4. Đặc điểm source code

- Sử dụng Spring Cloud Gateway
- Tích hợp Eureka để định tuyến động
- Có thể mở rộng filter để xử lý bảo mật

5.4. Phân tích source code StudentService

5.4.1. Cấu trúc module

```
student-service
├── controller
├── service
├── repository
├── entity
└── StudentServiceApplication.java
```

5.4.2. Controller Layer

- Nhận request từ Gateway
- Mapping các API REST
- Trả về dữ liệu dạng JSON

5.4.3. Service Layer

- Xử lý logic nghiệp vụ
- Kiểm tra dữ liệu đầu vào
- Gọi repository để thao tác database

5.4.4. Repository Layer

- Kế thừa JpaRepository
- Thực hiện CRUD trên bảng sinh viên
- Tách biệt hoàn toàn với service khác

5.4.5. Entity Layer

- Mapping bảng sinh viên

- Sử dụng annotation JPA (@Entity, @Id…)

5.5. Phân tích source code Room Service

5.5.1. Cấu trúc module

Tương tự **StudentService**, Room Service gồm:

- Controller
- Service
- Repository
- Entity

5.5.2. Đặc điểm

- Quản lý dữ liệu phòng
- Cung cấp API cho RentRoom Service
- Có database riêng (RoomDB)

5.6. Phân tích source code RentRoom Service

5.6.1. Vai trò trung tâm nghiệp vụ

RentRoom Service là module xử lý nghiệp vụ phức tạp nhất trong hệ thống.

5.6.2. Cấu trúc module

```
rentroom-service
├── controller
├── service
├── repository
├── entity
├── feign
└── RentRoomServiceApplication.java
```

5.6.3. Feign Client

- StudentClient
- RoomClient

Chức năng:

- Gọi API của **StudentService** và Room Service
- Trừu tượng hóa việc gọi HTTP

5.6.4. Service Layer

- Nhận yêu cầu tạo hợp đồng
- Gọi Feign Client xác minh dữ liệu
- Thực hiện lưu hợp đồng khi hợp lệ

5.6.5. Entity & Repository

- Entity mapping bảng hợp đồng
- Repository thao tác RentRoomDB
- Chỉ lưu ID sinh viên và phòng

5.7. Phân tích source code Frontend Service

5.7.1. Cấu trúc module

thuephong-frontend
├── controller
├── templates
├── static
└── ThuePhongFEApplication.java

5.7.2. Controller

- Nhận request từ trình duyệt
- Gọi API Gateway bằng RestTemplate
- Không xử lý nghiệp vụ phức tạp

5.7.3. View (Thymeleaf)

- Render HTML
- Hiển thị danh sách hợp đồng
- Form tạo hợp đồng

5.8. Phân tích file cấu hình và triển khai

5.8.1. application.yml

- Cấu hình port
- Cấu hình database
- Cấu hình Eureka Client

5.8.2. docker-compose.yml

- Triển khai PostgreSQL
- Cấu hình volume và port
- Phục vụ toàn bộ hệ thống

5.9. Đánh giá chất lượng source code

5.9.1. Ưu điểm

- Cấu trúc rõ ràng, dễ hiểu
- Tuân thủ kiến trúc Microservices
- Dễ mở rộng
- Phù hợp mục đích học tập và demo

5.9.2. Hạn chế

- Chưa có kiểm thử tự động
- Chưa xử lý lỗi nâng cao
- Chưa có logging tập trung

5.10. Tổng kết chương

Chương V đã phân tích chi tiết source code của hệ thống **ThuePhongService**, từ cấu trúc tổng thể đến từng module, từng tầng trong mỗi microservice. Qua đó cho thấy hệ thống được xây dựng đúng chuẩn kiến trúc Microservices, đảm bảo tính độc lập, khả năng mở rộng và dễ bảo trì.

CHƯƠNG VI: ĐÁNH GIÁ VÀ KẾT LUẬN

6.1. Đánh giá hệ thống

Hệ thống **ThuePhongService** đã được xây dựng và triển khai thành công theo kiến trúc **Microservices**, đáp ứng đầy đủ các yêu cầu nghiệp vụ đặt ra ban đầu. Các chức năng quản lý sinh viên, phòng và hợp đồng thuê phòng được phân tách rõ ràng, hoạt động ổn định và phối hợp đúng thiết kế.

Về kiến trúc, hệ thống đảm bảo:

- Mỗi service hoạt động độc lập, có cơ sở dữ liệu riêng.
- Giao tiếp giữa các service thông qua API, giảm sự phụ thuộc.
- Sử dụng Eureka Server và API Gateway hiệu quả trong quản lý và điều phối dịch vụ.
- Quy trình triển khai và chạy demo diễn ra ổn định, đúng luồng nghiệp vụ.

6.2. Ưu điểm và hạn chế

Ưu điểm:

- Áp dụng đúng nguyên lý kiến trúc Microservices.
- Cấu trúc source code rõ ràng, dễ bảo trì và mở rộng.
- Triển khai linh hoạt nhờ Docker.
- Phù hợp cho học tập, nghiên cứu và demo.

Hạn chế:

- Chưa tích hợp bảo mật và phân quyền.
- Chưa có cơ chế giám sát, logging tập trung.
- Giao diện người dùng còn đơn giản.

6.3. Hướng phát triển

Trong tương lai, hệ thống có thể được mở rộng bằng cách:

- Bổ sung cơ chế xác thực và bảo mật.
- Tích hợp giám sát và logging.
- Nâng cấp giao diện người dùng hiện đại hơn.
- Tự động hóa triển khai bằng CI/CD.

6.4. Kết luận

Hệ thống **ThuePhongService** là một mô hình minh họa hiệu quả cho việc áp dụng kiến trúc Microservices vào bài toán quản lý thuê phòng. Đề tài giúp làm rõ cách phân tích, thiết kế và triển khai hệ thống phần mềm phân tán, đồng thời tạo nền tảng để phát triển các hệ thống thực tế có quy mô lớn hơn trong tương lai.

---Hết---