**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH**

**LÂM PHÁT THỊNH**

**XÂY DỰNG HỆ THỐNG QUẢN LÝ NHÀ TRỌ VÀ SỐ HÓA CÁC TIỆN ÍCH**

**ĐỒ ÁN NGÀNH**

**NGÀNH KHOA HỌC MÁY TÍNH**

**TP. HỒ CHÍ MINH, 2024BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH**

**LÂM PHÁT THỊNH**

**XÂY DỰNG HỆ THỐNG QUẢN LÝ NHÀ TRỌ VÀ SỐ HÓA CÁC TIỆN ÍCH**

**Mã số sinh viên: 2151013094**

**ĐỒ ÁN NGÀNH**

**NGÀNH KHOA HỌC MÁY TÍNH**

**Giảng viên hướng dẫn: ThS. DƯƠNG HỮU THÀNH**

**TP. HỒ CHÍ MINH, 2024**

LỜI CẢM ƠN

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

TÓM TẮT ĐỐ ÁN NGÀNH

Dự án "Xây dựng hệ thống quản lý nhà trọ và số hóa các tiện ích" có mục tiêu chính là phát triển một nền tảng quản lý toàn diện cho các hoạt động liên quan đến việc cho thuê nhà trọ, nhằm mang lại sự tiện lợi cho cả chủ trọ và người thuê. Hệ thống được xây dựng dựa trên kiến trúc microservices, sử dụng Spring Boot làm nền tảng phát triển backend và cơ sở dữ liệu PostgreSQL và MongoDB để lưu trữ dữ liệu, giúp tối ưu hóa quá trình quản lý và truy xuất thông tin. Các chức năng chính của hệ thống bao gồm quản lý hóa đơn hàng tháng, phòng trọ, thông tin người thuê, hợp đồng thuê, và đặc biệt là tích hợp công nghệ nhận diện gương mặt bằng OpenCV để tự động phát hiện và cảnh báo khi có sự xuất hiện của người lạ trong khu vực nhà trọ.

Phía giao diện người dùng, ứng dụng dành cho admin được phát triển bằng ReactJS, giúp quản lý các tác vụ CRUD (tạo, đọc, cập nhật, xóa) liên quan đến các đối tượng như phòng, hợp đồng, người thuê, và hóa đơn một cách dễ dàng và trực quan. Bên cạnh đó, hệ thống cũng cung cấp một ứng dụng di động phát triển bằng React Native dành cho người thuê nhà, giúp họ truy cập vào các tiện ích trực tuyến như thanh toán hóa đơn, theo dõi thông tin hợp đồng, và nhận các thông báo quan trọng từ chủ trọ về tình trạng phòng và hóa đơn ngay trên điện thoại di động của mình.

Kết quả của dự án là hoàn thành một hệ thống quản lý nhà trọ thông minh với đầy đủ các chức năng cần thiết, đáp ứng nhu cầu quản lý của chủ trọ và mang lại sự tiện lợi cho người thuê thông qua các tính năng trực tuyến. Hệ thống không chỉ tối ưu hóa quy trình quản lý mà còn giúp nâng cao tính an ninh cho khu vực nhà trọ thông qua công nghệ cảnh báo người lạ, tạo nên một giải pháp toàn diện và hiện đại cho việc quản lý nhà trọ.

**MỤC LỤC**

[LỜI CẢM ƠN 1](#_Toc4324)

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN 2](#_Toc27244)

[TÓM TẮT ĐỐ ÁN NGÀNH 3](#_Toc29969)

[DANH MỤC TỪ VIẾT TẮT 6](#_Toc1880)

[DANH MỤC HÌNH VẼ 7](#_Toc4618)

[MỞ ĐẦU 9](#_Toc704)

[Chương 1. TỔNG QUAN ĐỀ TÀI 10](#_Toc23517)

[1.1. Giới thiệu đề tài 10](#_Toc20855)

[1.2. Lý do chọn đề tài 10](#_Toc3720)

[1.3. Mục tiêu và phạm vi đề tài 10](#_Toc23466)

[1.4. Phương pháp nghiên cứu 10](#_Toc3448)

[1.5. Bố cục báo cáo 11](#_Toc31200)

[Chương 2. CƠ SỞ LÝ THUYẾT 12](#_Toc1762)

[2.1. Spring Boot 12](#_Toc8736)

[2.1.1. Giới thiệu về Spring Boot 12](#_Toc7882)

[2.1.2. Kiến trúc Spring Boot 12](#_Toc15883)

[2.1.3. Cấu trúc dự án Spring Boot 16](#_Toc24378)

[2.2. React 18](#_Toc714)

[2.2.1. Giới thiệu về React 18](#_Toc14061)

[2.2.2. Kiến trúc React 18](#_Toc24960)

[2.2.3. Cấu trúc dự án React 19](#_Toc25226)

[2.3. Kiến trúc Microservice 22](#_Toc7187)

[2.3.1. Giới thiệu Microservice 22](#_Toc15538)

[2.3.2. Cấu trúc dự án microservice 23](#_Toc10642)

[2.4. Các dịch vụ và công cụ hỗ trợ 25](#_Toc4965)

[2.4.1. Docker 25](#_Toc11911)

[Chương 3. HỆ THỐNG QUẢN LÝ NHÀ TRỌ VÀ SỐ HÓA CÁC TIỆN ÍCH 26](#_Toc21889)

[3.1. Giới thiệu bài toán 26](#_Toc12517)

[3.2. Phân tích hệ thống 26](#_Toc16757)

[3.3. Thiết kế hệ thống 30](#_Toc11391)

[3.3.1. Lược đồ tuần tự (Sequence diagram) 30](#_Toc18016)

[3.3.2. Thiết kế cơ sở dữ liệu 33](#_Toc5166)

[3.3.3. Thiết kế giao diện 33](#_Toc4903)

[3.4. Kiến trúc hệ thống 33](#_Toc27905)

[3.5. Kết quả thực hiện 35](#_Toc27473)

[Chương 4. KẾ LUẬN VÀ HƯỚNG PHÁT TRIỂN 36](#_Toc23610)

[4.1. Kết luận 36](#_Toc22437)

[4.2. Hướng phát triển 36](#_Toc1506)

[TÀI LIỆU THAM KHẢO 37](#_Toc14561)

[PHỤ LỤC 38](#_Toc18738)

DANH MỤC TỪ VIẾT TẮT

DANH MỤC HÌNH VẼ

[Hình 2.1 Kiến trúc Spring Boot 12](#_Toc8317)

[Hình 2.2 Cấu trúc của một dự án Spring Boot 16](#_Toc14668)

[Hình 2.3 Kiến trúc React 19](#_Toc1095)

[Hình 2.4 Cấu trúc dự án React 20](#_Toc25399)

[Hình 2.5 Kiến trúc microservice 23](#_Toc23805)

[Hình 2.6 Cấu trúc một dự án microservice 23](#_Toc7572)

[Hình 3.1 Use case hệ thống 28](#_Toc12003)

[Hình 3.2 Sơ đồ tuần tự chức năng “Thêm dữ liệu gương mặt” 30](#_Toc31075)

[Hình 3.3 Sơ đồ tuần tự chức năng “Thêm cư dân” 31](#_Toc21152)

[Hình 3.4 Sơ đồ tuần tự chức năng “Thanh toán” 31](#_Toc20770)

[Hình 3.5 Sơ đồ tuần tự chức năng “Hoàn thành hóa đơn” 32](#_Toc6211)

[Hình 3.6 Sơ đồ tuần tự chức năng “Nhắn tin” 32](#_Toc9201)

[Hình 3.7 Lược đồ cơ sở dữ liệu quan hệ 33](#_Toc7043)

[Hình 3.8 Kiến trúc hệ thống 35](#_Toc11253)

DANH MỤC BẢNG

[Bảng 3.1 Đặc tả use case “Thêm dữ liệu gương mặt” 28](#_Toc17875)

[Bảng 3.2 Đặc tả use case “Thêm cư dân” 28](#_Toc7889)

[Bảng 3.3 Đặc tả use case “Thanh toán hóa đơn” 29](#_Toc28525)

[Bảng 3.4 Đặc tả use case “Hoàn thành hóa đơn” 29](#_Toc9407)

[Bảng 3.5 Đặc tả use case “Nhắn tin” 30](#_Toc10604)

MỞ ĐẦU

# TỔNG QUAN ĐỀ TÀI

## Giới thiệu đề tài

Trong thời đại số hóa hiện nay, việc quản lý nhà trọ không chỉ đơn thuần là ghi chép thông tin cư trú và thu tiền thuê, mà còn yêu cầu một hệ thống hiện đại để nâng cao hiệu quả và trải nghiệm cho cả chủ nhà lẫn người thuê. Đề tài "Xây dựng hệ thống quản lý nhà trọ và số hóa các tiện ích" nhằm phát triển một nền tảng trực tuyến hỗ trợ quản lý toàn diện các hoạt động liên quan đến kinh doanh nhà trọ.

## Lý do chọn đề tài

Trong bối cảnh công nghệ phát triển mạnh mẽ, số hóa và tự động hóa quy trình là xu hướng tất yếu. Việc quản lý nhà trọ bằng các phương pháp truyền thống như ghi chép tay, quản lý qua giấy tờ dần trở nên lỗi thời và kém hiệu quả. Hệ thống quản lý số hóa giúp tối ưu hóa quy trình, tăng tính chính xác và giảm thiểu sai sót. Người thuê cũng mong muốn một quy trình thuê nhà tiện lợi hơn, từ tìm kiếm, thanh toán đến quản lý các dịch vụ và tiện ích đi kèm. Các phương pháp quản lý thủ công thường gây ra nhiều bất tiện như khó khăn trong việc lưu trữ và tra cứu thông tin, không có cảnh báo kịp thời về thời hạn thanh toán, và việc tính toán sai lệch các chi phí tiện ích như điện, nước. Một hệ thống số hóa sẽ giúp khắc phục những nhược điểm này, giúp các bên liên quan làm việc hiệu quả hơn.

## Mục tiêu và phạm vi đề tài

Phát triển một hệ thống quản lý hiện đại giúp chủ nhà trọ theo dõi, quản lý tất cả các hoạt động liên quan đến nhà trọ một cách dễ dàng và hiệu quả. Hệ thống sẽ bao gồm quản lý thông tin người thuê, hợp đồng, thanh toán cũng như tăng cường an ninh bằng giải pháp nhận diện gương mặt phát hiện người lạ trong khu và đưa ra cảnh báo tức thời. Đồng thời cung cấp ứng dụng di động dành cho người thuê, giúp đồng bộ quy trình quản lý.

## Phương pháp nghiên cứu

Tiến hành khảo sát thực tế đối với chủ nhà trọ và người thuê nhà để thu thập thông tin về nhu cầu và khó khăn trong việc quản lý nhà trọ, quy trình thanh toán, và quản lý tiện ích. Khảo sát này giúp xác định rõ các vấn đề cần giải quyết và các tính năng cần có trong hệ thống. Nghiên cứu quy trình quản lý nhà trọ truyền thống để hiểu rõ các bước quản lý từ ghi nhận thông tin người thuê, quản lý hợp đồng, tính toán chi phí tiện ích, và thu tiền. Điều này giúp hệ thống được thiết kế hợp lý và có thể cải tiến các bước quản lý. Sử dụng công nghệ React để dễ dàng tạo ra giao diện thân thiện, dễ sử dụng cho cả chủ nhà và người thuê. Và Spring Boot để xây dựng server an toàn và linh hoạt.

## Bố cục báo cáo

Chương 1: Tổng quan đề tài, khái quát về đề tài và phương pháp nghiên cứu.

Chương 2: Cơ sở lý thuyết.

Chương 3: Hệ thống quản lý nhà trọ và số hóa các tiện ích.

Chương 4: Kết luận và hướng phát triển.

# CƠ SỞ LÝ THUYẾT

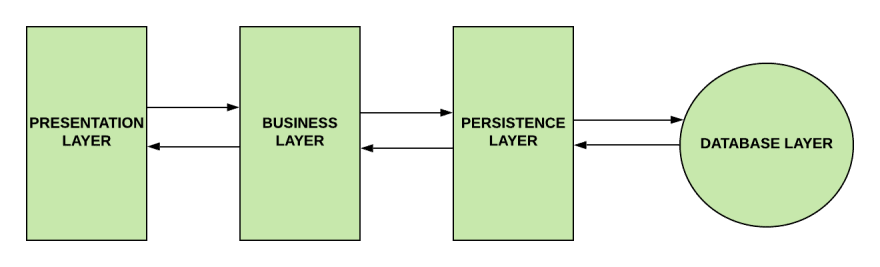
## Spring Boot

### Giới thiệu về Spring Boot

S**pring Boot** là một framework của Java được phát triển dựa trên Spring Framework. Mục tiêu của Spring Boot là giúp các nhà phát triển xây dựng và triển khai các ứng dụng web một cách nhanh chóng, dễ dàng mà không cần phải cấu hình phức tạp. Spring Boot tự động cấu hình các thành phần, giúp rút ngắn thời gian phát triển ứng dụng và cung cấp một cơ sở hạ tầng mạnh mẽ để xây dựng các ứng dụng microservices, API RESTful, hoặc ứng dụng doanh nghiệp phức tạp.

### Kiến trúc Spring Boot

Kiến trúc của Spring Boot bao gồm các lớp chính như **Presentation Layer** (Lớp trình bày) xử lý các yêu cầu HTTP thông qua các Controller, **Service Layer** (Lớp dịch vụ) chứa logic nghiệp vụ và quản lý tương tác giữa Controller và lớp dữ liệu, **Data Access Layer** (Lớp truy cập dữ liệu) xử lý việc giao tiếp với cơ sở dữ liệu hoặc các nguồn dữ liệu khác, và **Database Layer** (Lớp cơ sở dữ liệu) tương tác trực tiếp với cơ sở dữ liệu. Spring Boot hoạt động dựa trên nguyên tắc **Inversion of Control (IoC)** và **Dependency Injection (DI)**, giúp quản lý các thành phần của ứng dụng một cách hiệu quả và linh hoạt.



Hình 2.1 Kiến trúc Spring Boot

#### Presentation Layer

1. Mục đích: Lớp này chịu trách nhiệm xử lý các yêu cầu từ người dùng, thường thông qua giao thức HTTP. Đây là nơi mà các API RESTful hoặc giao diện người dùng tương tác với ứng dụng.
2. Các thành phần chính: Controller, Views (nếu sử dụng MVC với các công cụ như Thymeleaf hoặc JSP).
3. Chi tiết:

Controller: Là thành phần nhận các yêu cầu HTTP từ phía người dùng hoặc client. Controller chuyển tiếp yêu cầu này tới Service Layer để xử lý nghiệp vụ và trả về kết quả cho người dùng thông qua response. Trong Spring Boot, các Controller được định nghĩa bằng annotation @RestController hoặc @Controller.

Views (optional): Nếu sử dụng mô hình MVC, lớp này sẽ chứa các view template để hiển thị dữ liệu tới người dùng (ví dụ: các tệp HTML với Thymeleaf).

1. Vai trò chính: Điều phối và trả về dữ liệu đã xử lý dưới dạng JSON, HTML, hoặc XML (tùy thuộc vào loại ứng dụng).

#### Service Layer (Lớp dịch vụ)

1. Mục đích: Chứa các logic nghiệp vụ chính của ứng dụng. Lớp này là nơi tất cả các quy tắc và điều kiện nghiệp vụ được thực hiện.
2. Các thành phần chính: Service classes, thường được đánh dấu bằng annotation @Service.
3. Chi tiết:

Business Logic: Service Layer là nơi toàn bộ logic nghiệp vụ của ứng dụng diễn ra. Nó lấy dữ liệu từ Presentation Layer (thông qua các request từ Controller), xử lý, và giao tiếp với Data Access Layer để lưu hoặc lấy dữ liệu.

Transaction Management: Đây cũng là nơi mà các giao dịch dữ liệu được quản lý. Spring hỗ trợ quản lý giao dịch thông qua các annotation như @Transactional.

1. Vai trò chính: Xử lý toàn bộ logic của ứng dụng, giữ cho Controller gọn gàng và tập trung vào việc xử lý yêu cầu từ client.

#### Data Access Layer (Lớp truy cập dữ liệu)

1. Mục đích: Lớp này chịu trách nhiệm tương tác với cơ sở dữ liệu hoặc các nguồn dữ liệu khác (API, file, v.v.). Nó cung cấp một lớp abstraction giữa Service Layer và Database Layer, đảm bảo việc truy xuất dữ liệu được tách biệt và dễ quản lý.
2. Các thành phần chính: Repository classes, đánh dấu bằng annotation @Repository.
3. Chi tiết:

Data Access Object (DAO): Trong Spring Boot, các DAO hoặc Repository đảm nhận việc thực hiện các hoạt động CRUD (Create, Read, Update, Delete) trên cơ sở dữ liệu. Spring Boot cung cấp Spring Data JPA giúp giảm bớt việc viết mã bằng cách cung cấp các interface như CrudRepository hoặc JpaRepository.

ORM (Object Relational Mapping): Spring Boot thường sử dụng Hibernate, một framework ORM, để ánh xạ các đối tượng Java thành các bản ghi trong cơ sở dữ liệu.

1. Vai trò chính: Cung cấp phương tiện để Service Layer tương tác với cơ sở dữ liệu mà không phải lo lắng về chi tiết triển khai.

#### Database Layer (Lớp cơ sở dữ liệu)

1. Mục đích: Tương tác trực tiếp với cơ sở dữ liệu thực tế, nơi lưu trữ dữ liệu ứng dụng.
2. Các thành phần chính: Hệ quản trị cơ sở dữ liệu (MySQL, PostgreSQL, Oracle, v.v.).
3. Chi tiết:

Schema: Định nghĩa cấu trúc của cơ sở dữ liệu như bảng, cột, khóa chính, khóa ngoại, v.v.

SQL: Là các câu truy vấn SQL được tự động tạo bởi ORM framework (Hibernate) hoặc có thể do người lập trình viết tay cho các truy vấn phức tạp.

Connection Pooling: Spring Boot thường sử dụng các kết nối đã được tối ưu hóa bằng cách quản lý pool các kết nối tới cơ sở dữ liệu (thông qua các thư viện như HikariCP).

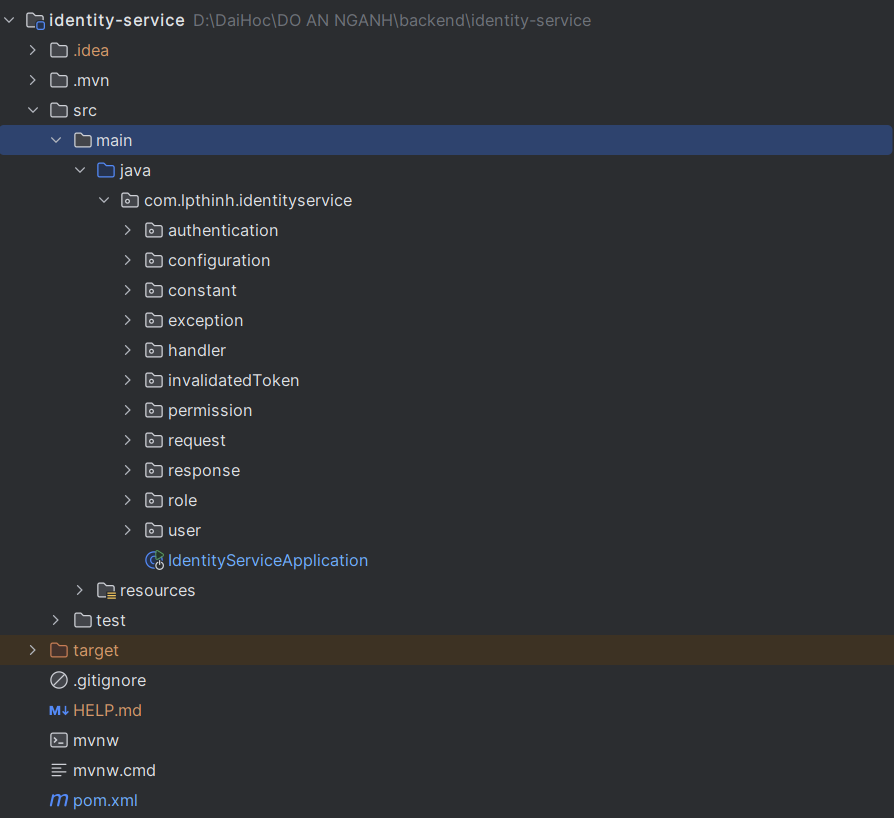
1. Vai trò chính: Lưu trữ và quản lý dữ liệu một cách nhất quán và hiệu quả, đảm bảo tính toàn vẹn của dữ liệu.

#### Nguyên tắc kiến trúc chính

Inversion of Control (IoC): Trong Spring Boot, nguyên tắc IoC cho phép framework kiểm soát luồng ứng dụng thay vì để các lớp tự kiểm soát. Thông qua IoC Container, Spring tự động khởi tạo các đối tượng và quản lý các phụ thuộc của chúng.

Dependency Injection (DI): DI là một trong những cơ chế của IoC, cho phép Spring Boot "inject" các đối tượng vào trong các lớp phụ thuộc vào chúng, thay vì để các lớp tự khởi tạo các đối tượng cần thiết. Điều này giúp mã nguồn dễ bảo trì và kiểm thử hơn.

### Cấu trúc dự án Spring Boot



Hình 2.2 Cấu trúc của một dự án Spring Boot

1. Thư mục src/main/java

Đây là thư mục chính chứa mã nguồn của ứng dụng. Nó thường được chia thành các thành phần chính theo kiến trúc 3 tầng (MVC: Model, View, Controller hoặc 4 tầng). Trong đó thư mục com.example.lpthinh (package gốc): Đây là nơi chứa các package con bao gồm:

* Controller: Chứa các lớp điều khiển (controller), xử lý các yêu cầu từ người dùng hoặc hệ thống bên ngoài, và trả về kết quả. Các lớp thường có annotation như @RestController hoặc @Controller. Nhận yêu cầu HTTP, gọi các dịch vụ, và trả về phản hồi (thường là JSON hoặc HTML).
* Service: Chứa các lớp cung cấp logic nghiệp vụ chính. Lớp này thường được đánh dấu với annotation @Service. Xử lý nghiệp vụ chính, giao tiếp với các lớp DAO (Repository) và cung cấp dữ liệu cho Controller.
* Repository (DAO): Chứa các lớp tương tác trực tiếp với cơ sở dữ liệu, thường sử dụng các annotation như @Repository. Các lớp này thường sử dụng JPA, Hibernate, hoặc các thư viện ORM khác để thực hiện các thao tác CRUD với cơ sở dữ liệu. Thực hiện các truy vấn vào cơ sở dữ liệu và trả về dữ liệu dưới dạng đối tượng Java.
* Model/Entity: Chứa các lớp mô hình hóa dữ liệu, đại diện cho các bảng trong cơ sở dữ liệu. Các lớp này thường được chú thích với @Entity để ánh xạ với các bảng dữ liệu. Đại diện cho các đối tượng dữ liệu, ánh xạ trực tiếp tới các bảng cơ sở dữ liệu.
* DTO (Data Transfer Object): Các đối tượng trung gian dùng để chuyển dữ liệu giữa các lớp, tránh việc sử dụng các Entity trực tiếp. Truyền tải dữ liệu giữa client và server, hoặc giữa các tầng trong ứng dụng.
* Config: Chứa các lớp cấu hình (như cấu hình bảo mật, cấu hình cơ sở dữ liệu) với các annotation như @Configuration. Cung cấp các cấu hình chung cho ứng dụng, như cấu hình bảo mật Spring Security, hoặc cấu hình kết nối đến cơ sở dữ liệu.

1. Thư mục src/main/resources

Chứa các tài nguyên không phải là mã nguồn Java, chẳng hạn như tệp cấu hình, tệp HTML, tệp static.

* application.properties/application.yml: Tệp cấu hình chính của Spring Boot. Tại đây bạn có thể khai báo cấu hình cho kết nối cơ sở dữ liệu, port của server, các thông số bảo mật... Cấu hình các thông số chung cho toàn ứng dụng.
* static/: Chứa các tệp tĩnh như hình ảnh, CSS, JavaScript. Chứa các tài nguyên tĩnh để hiển thị cho frontend.
* templates/: Chứa các tệp HTML (nếu bạn sử dụng Thymeleaf hoặc một engine template khác). Tạo giao diện động cho người dùng (được sử dụng trong các ứng dụng web MVC).

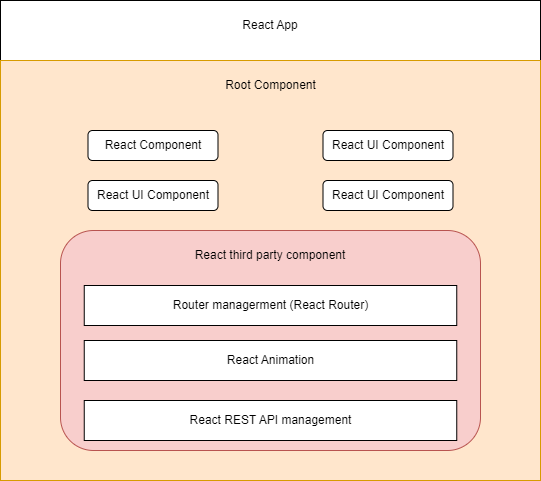
## React

### Giới thiệu về React

**React** (ReactJs) là một thư viện JavaScript phát triển bởi Facebook, nổi bật với khả năng xây dựng giao diện người dùng (UI) một cách hiệu quả và linh hoạt. React sử dụng “**Component-Based Architecture**”, cho phép chia nhỏ giao diện thành các thành phần (components) tái sử dụng được, quản lý trạng thái và logic riêng biệt cho từng phần của ứng dụng. Thư viện này sử dụng “**Virtual DOM**” để cải thiện hiệu suất bằng cách chỉ cập nhật những phần giao diện có thay đổi, thay vì toàn bộ trang. React hỗ trợ “**JSX”**, một cú pháp kết hợp giữa HTML và JavaScript, giúp viết mã giao diện dễ hiểu và gần gũi với các nhà phát triển.

### Kiến trúc React

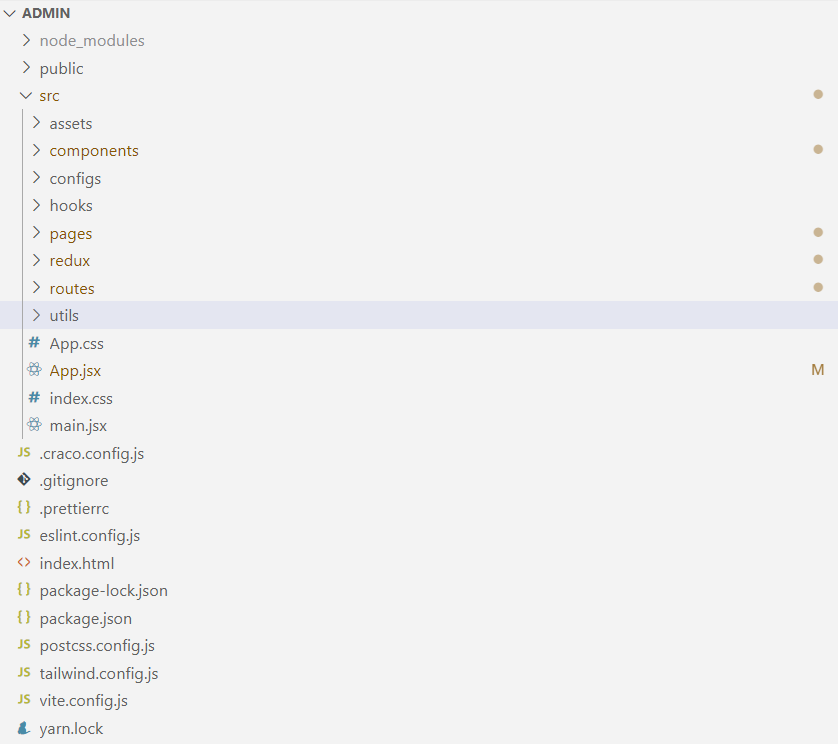
Ứng dụng React bắt đầu với một thành phần gốc duy nhất, từ đó xây dựng các thành phần khác để tạo ra giao diện người dùng. Thành phần gốc được tạo ra bằng cách sử dụng một hoặc nhiều thành phần nhỏ hơn, và các thành phần này có thể được lồng vào nhau ở bất kỳ mức độ nào. Thành phần là một trong những khái niệm cốt lõi của React, và thay vì kế thừa, mỗi thành phần thường được xây dựng bằng cách kết hợp các thành phần nhỏ hơn. Hầu hết các thành phần trong React đóng vai trò là các thành phần giao diện người dùng. Để mở rộng chức năng của ứng dụng, React cũng cho phép tích hợp các thành phần của bên thứ ba, như các công cụ định tuyến, hoạt ảnh, và quản lý trạng thái.



Hình 2.3 Kiến trúc React

### Cấu trúc dự án React

Cấu trúc của một dự án ReactJS thường tuân theo nguyên tắc tổ chức mã nguồn nhằm quản lý tốt các thành phần (components), dịch vụ, và tài nguyên. Dưới đây là cấu trúc thư mục phổ biến trong một dự án ReactJS, kèm với vai trò của từng thành phần:



Hình 2.4 Cấu trúc dự án React

1. Thư mục gốc dự án

Đây là thư mục cấp cao nhất chứa các tệp cấu hình, tài liệu và các công cụ quản lý dự án.

* package.json: Tệp cấu hình quan trọng của npm (Node Package Manager), chứa thông tin về dự án, dependencies (các thư viện cần thiết), scripts, và phiên bản dự án. Xác định tất cả các dependency, script chạy, phiên bản, và thông tin về dự án.
* node\_modules/: Thư mục chứa tất cả các gói (packages) và thư viện mà dự án yêu cầu, được cài đặt thông qua npm hoặc yarn. Chứa các thư viện cần thiết cho dự án.
* .gitignore: Tệp chứa danh sách các tệp và thư mục mà Git sẽ bỏ qua (không theo dõi). Ngăn các tệp không cần thiết như node\_modules/ khỏi việc được thêm vào Git.
* public/: Thư mục chứa các tài nguyên tĩnh mà React không quản lý như tệp HTML chính, favicon, và các tệp tĩnh khác.

1. Thư mục src/

Đây là thư mục chứa mã nguồn chính của dự án ReactJS, bao gồm các component, tệp cấu hình, và các tài nguyên ứng dụng.

* index.js / index.tsx: Tệp điểm vào chính của ứng dụng React. Tại đây, ReactDOM.render được gọi để render ứng dụng vào tệp index.html trong thư mục public/. Khởi chạy ứng dụng React và liên kết nó với DOM.
* App.js / App.tsx: Component gốc (root component) của ứng dụng. Tất cả các thành phần khác sẽ nằm trong component này. Đóng vai trò như là root component chứa các routes và components con.
* components/: Thư mục này chứa các component độc lập có thể tái sử dụng được. Component là những khối xây dựng chính trong React. Quản lý các phần UI độc lập và có thể tái sử dụng, như nút bấm, form, hoặc các module giao diện khác.
* pages/: Thư mục chứa các trang của ứng dụng. Mỗi trang có thể là một component lớn hơn, đại diện cho một màn hình hoặc route trong ứng dụng. Đóng vai trò hiển thị các màn hình chính của ứng dụng như Home, About, hoặc Dashboard.
* services/ (hoặc api/): Thư mục chứa các tệp quản lý việc giao tiếp với backend, API. Nó có thể chứa các hàm gọi API, quản lý token, hoặc xử lý các request HTTP. Tương tác với backend, gửi request API, và nhận dữ liệu từ server.
* hooks/: Thư mục này chứa các hook tùy chỉnh (custom hooks) trong React. Hook là các hàm giúp quản lý trạng thái và logic một cách dễ dàng hơn. Đóng gói logic có thể tái sử dụng bằng cách tạo ra các custom hooks cho các chức năng như quản lý form, quản lý authentication, v.v.
* redux/: Nếu ứng dụng sử dụng Redux để quản lý trạng thái, thư mục này sẽ chứa các reducer, action, và store. Quản lý toàn bộ trạng thái của ứng dụng trong Redux store.
* assets/: Thư mục chứa các tài nguyên tĩnh như hình ảnh, font, hoặc video. Chứa các tài nguyên như hình ảnh, biểu tượng, hoặc file media mà ứng dụng sử dụng.
* utils/ (hoặc helpers/): Thư mục chứa các hàm tiện ích dùng chung trong ứng dụng, giúp tái sử dụng mã và quản lý logic phức tạp hơn. Chứa các hàm tiện ích hoặc logic phụ trợ có thể tái sử dụng nhiều nơi trong ứng dụng.

1. Thư mục public/

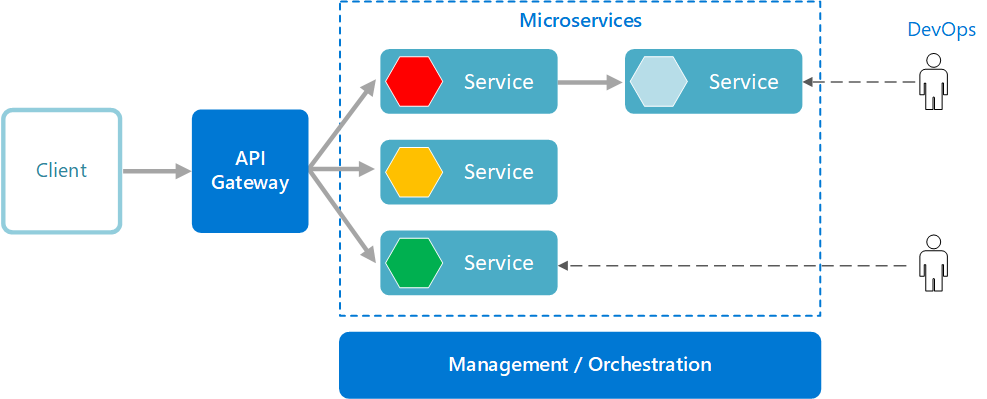
Thư mục này chứa các tệp tài nguyên công khai, không qua xử lý bởi Webpack. Nó bao gồm các tệp tĩnh như hình ảnh, favicon, và tệp HTML chính.

* index.html: Tệp HTML gốc nơi ứng dụng React sẽ được render vào. Đây là nơi chứa thẻ <div id="root"></div>, và ứng dụng React sẽ được nhúng vào thẻ này. Cung cấp khung HTML cơ bản cho ứng dụng React.
* favicon.ico: Icon nhỏ xuất hiện trên tab của trình duyệt.

## Kiến trúc Microservice

### Giới thiệu Microservice

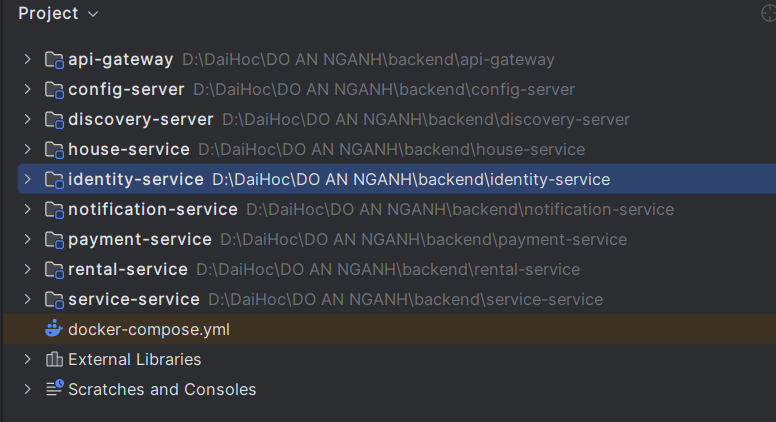
Microservice là một kiến trúc phần mềm trong đó ứng dụng được chia thành các dịch vụ nhỏ, độc lập, mỗi dịch vụ đảm nhiệm một chức năng cụ thể. Các microservice có thể được phát triển, triển khai, và mở rộng độc lập, giúp cải thiện tính linh hoạt và khả năng bảo trì của hệ thống. Chúng giao tiếp với nhau qua các giao thức nhẹ như HTTP/REST hoặc thông qua các message broker như Kafka. Mô hình này mang lại nhiều lợi ích như dễ mở rộng, tăng tốc độ phát triển và bảo trì dễ dàng hơn, nhưng cũng đi kèm thách thức về quản lý, theo dõi lỗi và hiệu suất nếu không được tối ưu hóa.



Hình 2.5 Kiến trúc microservice

### Cấu trúc dự án microservice

Cấu trúc của một dự án microservice thường bao gồm nhiều dịch vụ nhỏ gọn, độc lập và mỗi dịch vụ đảm nhận một nhiệm vụ cụ thể. Dưới đây là giải thích về các thành phần phổ biến trong kiến trúc microservices, bao gồm Gateway, Discovery Server, Config Server, và Tasks Service:



Hình 2.6 Cấu trúc một dự án microservice

1. API Gateway:

Chức năng: API Gateway đóng vai trò như một cổng vào duy nhất cho toàn bộ hệ thống microservice. Nó nhận các yêu cầu từ client và định tuyến đến các dịch vụ cụ thể tương ứng trong hệ thống.

Vai trò:

* Routing: Định tuyến yêu cầu đến đúng microservice dựa trên các endpoint URL.
* Load Balancing: Cân bằng tải giữa các instance của microservice.
* Authentication & Authorization: Xác thực và phân quyền cho các yêu cầu từ client trước khi chúng được gửi đến các dịch vụ.
* Rate Limiting: Giới hạn tần suất gọi API để tránh quá tải.
* Công nghệ phổ biến: Spring Cloud Gateway, Netflix Zuul.

1. Discovery Server (Service Registry):

Chức năng: Discovery Server là một máy chủ đăng ký dịch vụ, nơi các microservice tự động đăng ký và cung cấp thông tin về địa chỉ của chúng.

Vai trò:

* Service Registration: Các dịch vụ khi khởi động sẽ tự động đăng ký với Discovery Server bằng cách cung cấp thông tin như IP, port và tên dịch vụ.
* Service Discovery: Các microservice khác hoặc API Gateway có thể tìm kiếm và kết nối với nhau thông qua Discovery Server.
* Công nghệ phổ biến: Netflix Eureka, Consul, Zookeeper.

1. Config Server:

Chức năng: Config Server cung cấp cấu hình cho các microservice từ một vị trí tập trung. Điều này giúp dễ dàng quản lý và thay đổi cấu hình mà không cần khởi động lại các dịch vụ.

Vai trò:

* Centralized Configuration: Tất cả các microservice đều có thể lấy cấu hình từ một nơi duy nhất.
* Dynamic Configuration: Các dịch vụ có thể tự động cập nhật cấu hình mà không cần phải khởi động lại khi có sự thay đổi.
* Environment-Specific Configuration: Cấu hình có thể thay đổi theo môi trường như dev, prod, test.
* Công nghệ phổ biến: Spring Cloud Config.

1. Tasks Service (Microservices cụ thể)

Chức năng: Đây là các dịch vụ cụ thể đảm nhiệm các chức năng nghiệp vụ khác nhau của hệ thống. Ví dụ: User Service để quản lý người dùng hoặc Payment Service để xử lý thanh toán.

Vai trò:

* Business Logic: Thực hiện các tác vụ nghiệp vụ cụ thể mà nó được giao.
* Communication: Tương tác với các dịch vụ khác thông qua API hoặc message queue.
* Scalability: Các microservice này có thể mở rộng độc lập tùy theo nhu cầu sử dụng.
* Công nghệ phổ biến: Spring Boot, Node.js, .NET Core.

## Các dịch vụ và công cụ hỗ trợ

### Docker

**Docker** là nền tảng mã nguồn mở cho phép tạo, triển khai và chạy các ứng dụng trong các **container**, cung cấp môi trường ảo hóa nhẹ chứa tất cả các thành phần cần thiết cho ứng dụng, bao gồm mã nguồn, thư viện, và phụ thuộc. Docker đảm bảo tính nhất quán giữa các môi trường phát triển, kiểm thử và triển khai bằng cách đóng gói ứng dụng vào một container duy nhất, giúp triển khai dễ dàng trên nhiều nền tảng như máy chủ vật lý, máy ảo, và dịch vụ đám mây. Với **Docker Compose**, quản lý nhiều container cùng lúc trở nên đơn giản, hỗ trợ các ứng dụng phức tạp bao gồm frontend, backend, và các dịch vụ phụ trợ. Docker giúp quy trình phát triển, kiểm thử và triển khai trở nên hiệu quả hơn bằng cách cung cấp tính nhất quán, khả năng mở rộng linh hoạt và triển khai dễ dàng.

# **HỆ THỐNG QUẢN LÝ NHÀ TRỌ VÀ SỐ HÓA CÁC TIỆN ÍCH**

## **Giới thiệu bài toán**

**Hệ thống Quản lý Nhà trọ và Số hóa các Tiện ích** là một giải pháp nhằm hiện đại hóa việc quản lý các cơ sở lưu trú như nhà trọ hoặc căn hộ cho thuê. Hệ thống này sử dụng kiến trúc **microservice** với **Spring Boot** cho backend và **ReactJS** cho frontend, giúp quản lý các chức năng như quản lý người thuê, hợp đồng, thanh toán và các tiện ích liên quan.

Với **Spring Boot** microservice, hệ thống được phân chia thành nhiều dịch vụ nhỏ độc lập như dịch vụ quản lý người thuê, dịch vụ quản lý hợp đồng, và dịch vụ thanh toán, mỗi dịch vụ có thể phát triển và triển khai một cách riêng biệt. Điều này mang lại tính linh hoạt, khả năng mở rộng, và dễ bảo trì cho toàn bộ hệ thống. Các dịch vụ này giao tiếp qua giao thức HTTP/REST hoặc sử dụng message broker như Kafka để đảm bảo trao đổi dữ liệu mượt mà giữa các thành phần.

Ở phía frontend, **ReactJS** cung cấp giao diện người dùng hiện đại và thân thiện, cho phép người dùng dễ dàng tương tác với hệ thống. Chủ nhà và người thuê có thể quản lý hợp đồng, theo dõi trạng thái thanh toán, và sử dụng các tiện ích trực tuyến như yêu cầu bảo trì, xem hóa đơn điện nước, và nhận thông báo.

## **Phân tích hệ thống**

1. Lược đồ Use case

Hệ thống có 2 actor chính:

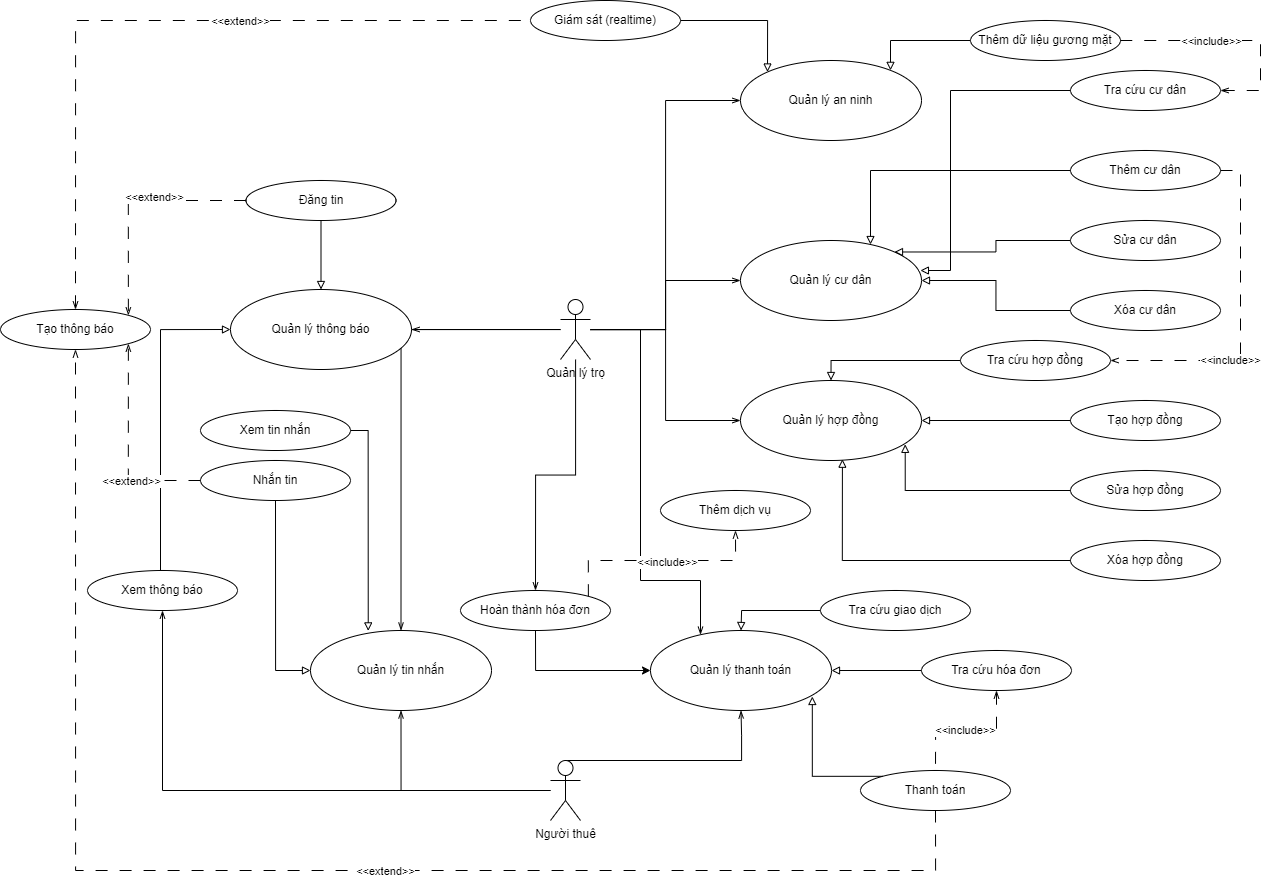
* Người thuê trọ (Tenant): Đây là người dùng của hệ thống, họ có các nhu cầu như thuê phòng, ký hợp đồng, thanh toán tiền thuê.
* Quản lý trọ (Manager): Đây là người chịu trách nhiệm quản lý toàn bộ hệ thống. Quản lý trọ có thể là chủ nhà hoặc người được ủy quyền để giám sát cư dân, hợp đồng thuê nhà, thanh toán, và an ninh của tòa nhà.

Các use case chính trong hệ thống gồm:

* Quản lý cư dân (Manage Residents): Quản lý cư dân bao gồm việc thêm mới cư dân, cập nhật thông tin cá nhân, và theo dõi tình trạng cư trú.
* Quản lý an ninh (Manage Security): Bao gồm giám sát an ninh qua camera, nhận diện người lạ và thông báo đến người thuê trọ.
* Quản lý hợp đồng (Manage Contracts): Chức năng này cho phép tạo mới, chỉnh sửa, hoặc kết thúc hợp đồng thuê. Quản lý trọ có thể theo dõi thời hạn hợp đồng và tình trạng ký kết.
* Quản lý thanh toán (Manage Payments): Bao gồm việc theo dõi và xử lý các khoản thanh toán từ người thuê trọ, như tiền thuê hàng tháng, phí dịch vụ, và xuất hóa đơn.

Mối quan hệ giữa Actor và Use Case

* Người thuê trọ (Tenant): Tương tác với hệ thống thông qua các use case như quản lý hợp đồng (ký và theo dõi hợp đồng), quản lý thanh toán (thanh toán tiền thuê), và quản lý an ninh (nhận cảnh báo).
* Quản lý trọ (Manager): Có quyền truy cập vào tất cả các use case bao gồm quản lý cư dân, quản lý an ninh, quản lý hợp đồng, và quản lý thanh toán. Quản lý sẽ thực hiện việc quản lý, giám sát và điều chỉnh toàn bộ hệ thống.



Hình 3.1 Use case hệ thống

1. Đặc tả Use case

Đặc tả use case “Thêm dữ liệu gương mặt”:

|  |  |
| --- | --- |
| **Use case Id** | UC-01 |
| **Tên use Case** | Thêm dữ liệu gương mặt |
| **Mô tả** | Use case này cho phép quản lý trọ lấy hình ảnh gương mặt của người thuê để đưa vào dữ liệu hệ thống. |
| **Actor chính** | Quản lý trọ |
| **Actor phụ** | Không có |
| **Tiền điều kiện** | Thông tin người  thuê đã được đưa vào dữ liệu hệ thống |
| **Hậu điều kiện** | Không có |
| **Luồng hoạt động** | 1. Tra cứu cư dân 2. Chọn một cư dân   3. Lấy hình ảnh của cư dân thông qua camera  4. Hoàn tất thêm dữ liệu |
| **Luồng thay thế** | Không có |
| **Luồng ngoại lệ** | Không có |

Bảng 3.1 Đặc tả use case “Thêm dữ liệu gương mặt”

Đặc tả use case “Thêm cư dân”:

|  |  |
| --- | --- |
| **Use case Id** | UC-02 |
| **Tên use Case** | Thêm cư dân |
| **Mô tả** | Use case này cho phép quản lý trọ thực hiện chức năng nhập thông tin cư dân để đưa vào dữ liệu hệ thống |
| **Actor chính** | Quản lý trọ |
| **Actor phụ** | Không có |
| **Tiền điều kiện** | Đã tồn tại hợp đồng cho thuê |
| **Hậu điều kiện** | Không có |
| **Luồng hoạt động** | 1. Truy cập vào trang “Cư dân”  2. Chọn chức năng tạo mới  3. Nhập các thông tin của người thuê  4. Chọn hợp đồng đã tồn tại  5. Hoàn thành chức năng thêm cư dân |
| **Luồng thay thế** | Không có |
| **Luồng ngoại lệ** | Không có |

Bảng 3.2 Đặc tả use case “Thêm cư dân”

Đặc tả use case “Thanh toán hóa đơn”:

|  |  |
| --- | --- |
| **Use case Id** | UC-03 |
| **Tên use Case** | Thanh toán hóa đơn |
| **Mô tả** | Use case cho phép người thuê, quản lý trọ thực hiện chức năng thanh toán trực tuyến các hóa đơn hằng tháng. |
| **Actor chính** | Cư dân, quản lý trọ |
| **Actor phụ** | Stripe |
| **Tiền điều kiện** | Sử dụng thẻ thanh toán quốc tế |
| **Hậu điều kiện** | Không có |
| **Luồng hoạt động** | 1. Truy cập vào trang “Hóa đơn của tôi”  2. Chọn một hóa đơn trong trang  3. Click vào nút thanh toán  4. Chuyển trang đến Stripe Payment  5. Nhập thông tin thẻ thanh toán quốc tế  6. Hoàn thành thanh toán  7. Email xác nhận thanh toán  8. Thông báo đến quản lý trọ |
| **Luồng thay thế** | 1. Truy cập vào trang “Hóa đơn của tôi”  2. Chọn một hóa đơn trong trang  3. Chọn chuyển khoản qua mã QR  4. Click vào nút đã thanh toán  5. Thông báo đến quản lý trọ  6. Xác nhận thanh toán  7. Hoàn thành thanh toán. |
| **Luồng ngoại lệ** | 1. Truy cập vào trang “Hóa đơn của tôi”  2. Chọn một hóa đơn trong trang  3. Click vào nút thanh toán  4. Chuyển trang đến Stripe Payment  5. Nhập thông tin thẻ thanh toán quốc tế  6. Số dư không đủ  7. Thanh toán không thành công |

Bảng 3.3 Đặc tả use case “Thanh toán hóa đơn”

Đặc tả use case “Hoàn thành hóa đơn”:

|  |  |
| --- | --- |
| **Use case Id** | UC-04 |
| **Tên use Case** | Hoàn thành hóa đơn |
| **Mô tả** | Use case cho phép quản lý trọ thực hiện chức năng thêm các dịch vụ đề hoàn thành hóa đơn hằng tháng. |
| **Actor chính** | Quản lý trọ |
| **Actor phụ** | Không có |
| **Tiền điều kiện** | Hóa đơn đã đươc tạo |
| **Hậu điều kiện** | Không có |
| **Luồng hoạt động** | 1. Truy cập vào trang “Quản lý hóa đơn” 2. Hiển thị tất cả các hóa đơn trong tháng 3. Chọn một hóa đơn 4. Điền thông tin các dịch vụ đã sử dụng 5. Hoàn thành hóa đơn |
| **Luồng thay thế** | Không có. |
| **Luồng ngoại lệ** | Không có |

Bảng 3.4 Đặc tả use case “Hoàn thành hóa đơn”

Đặc tả use case “Nhắn tin”:

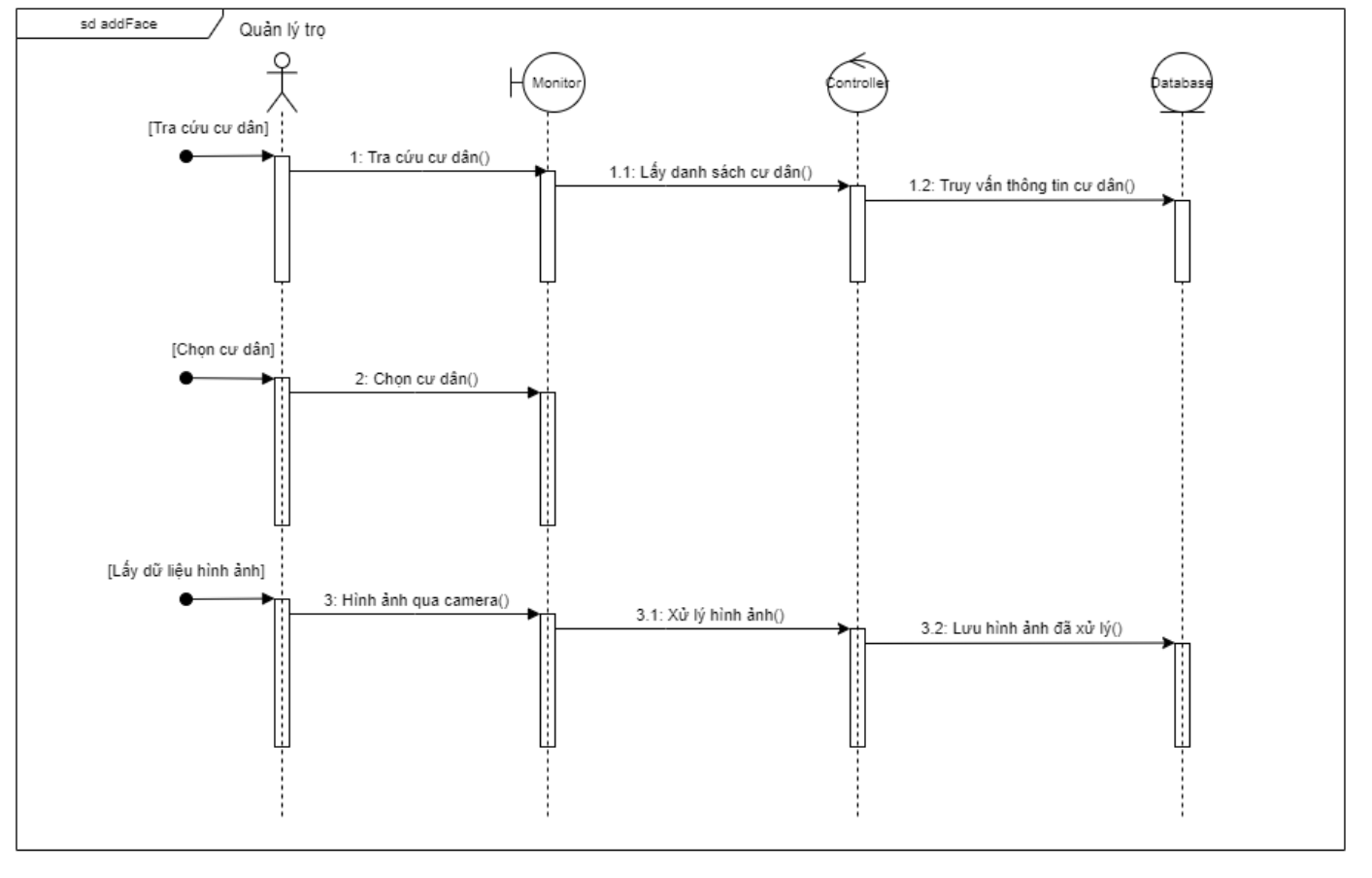
|  |  |
| --- | --- |
| **Use case Id** | UC-05 |
| **Tên use Case** | Nhắn tin |
| **Mô tả** | Use case cho phép người dùng nhắn tin (realtime) cho các người dùng khác trong hệ thống |
| **Actor chính** | Người dùng |
| **Actor phụ** | Không có |
| **Tiền điều kiện** | Không có |
| **Hậu điều kiện** | Không có |
| **Luồng hoạt động** | 1. Truy cập vào trang “Tin nhắn” 2. Soạn tin nhắn văn bản 3. Click nút gửi 4. Người dùng đã chỉ định nhận tin nhắn (realtime) |
| **Luồng thay thế** | Không có. |
| **Luồng ngoại lệ** | Không có |

Bảng 3.5 Đặc tả use case “Nhắn tin”

## Thiết kế hệ thống

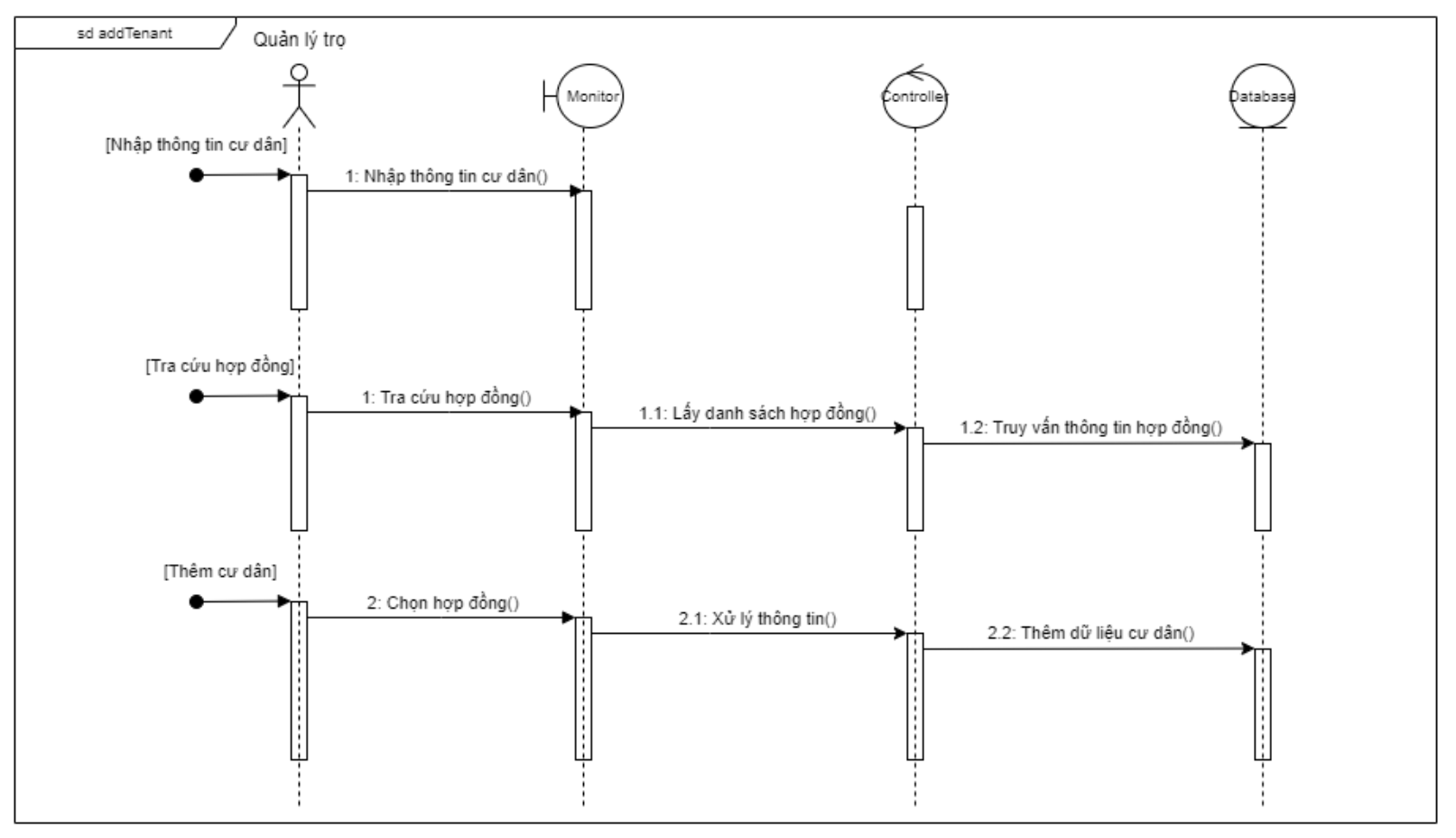
### Lược đồ tuần tự (Sequence diagram)

Sơ đồ tuần tự chức năng “Thêm dữ liệu gương mặt”:



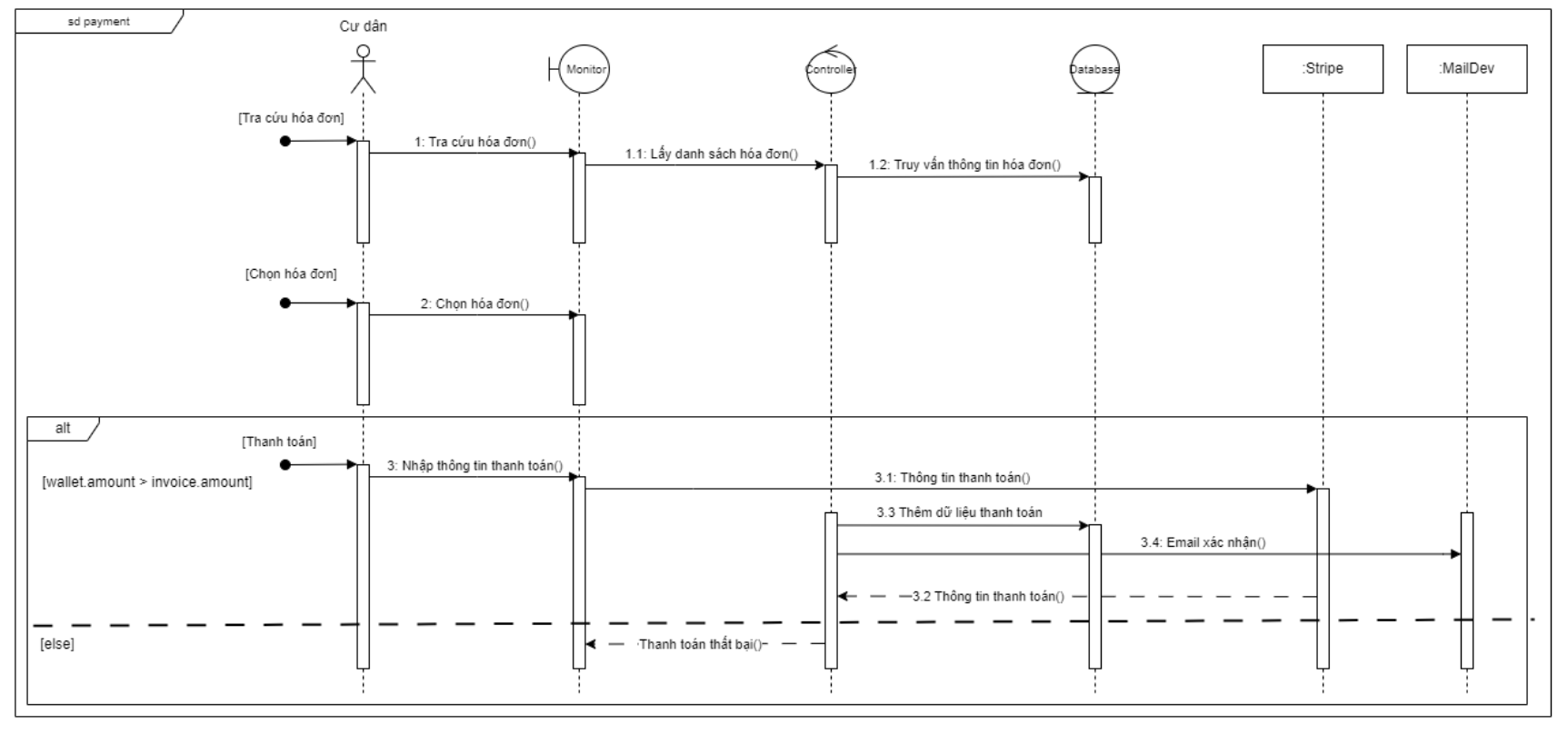
Hình 3.2 Sơ đồ tuần tự chức năng “Thêm dữ liệu gương mặt”

Sơ đồ tuần tự chức năng “Thêm cư dân”:



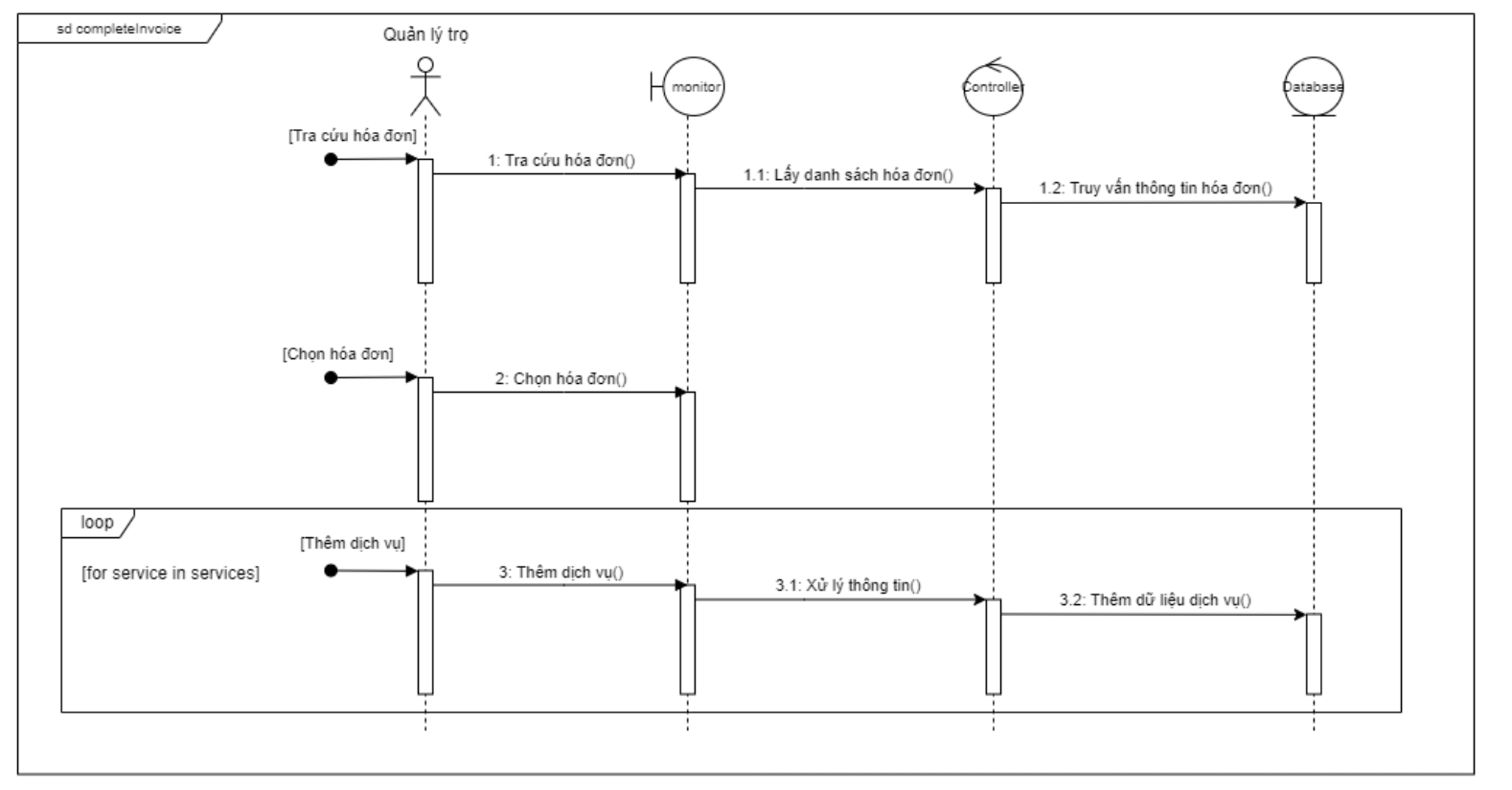
Hình 3.3 Sơ đồ tuần tự chức năng “Thêm cư dân”

Sơ đồ tuần tự chức năng “Thanh toán”:



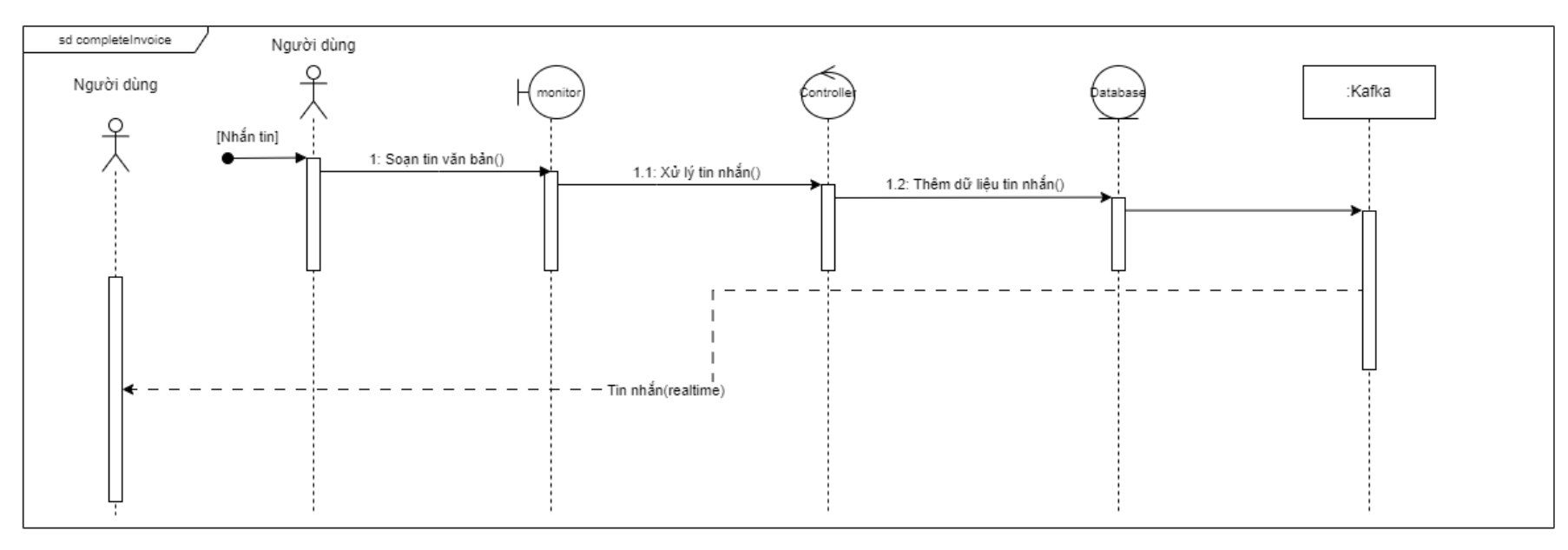
Hình 3.4 Sơ đồ tuần tự chức năng “Thanh toán”

Sơ đồ tuần tự chức năng “Hoàn thành hóa đơn”:



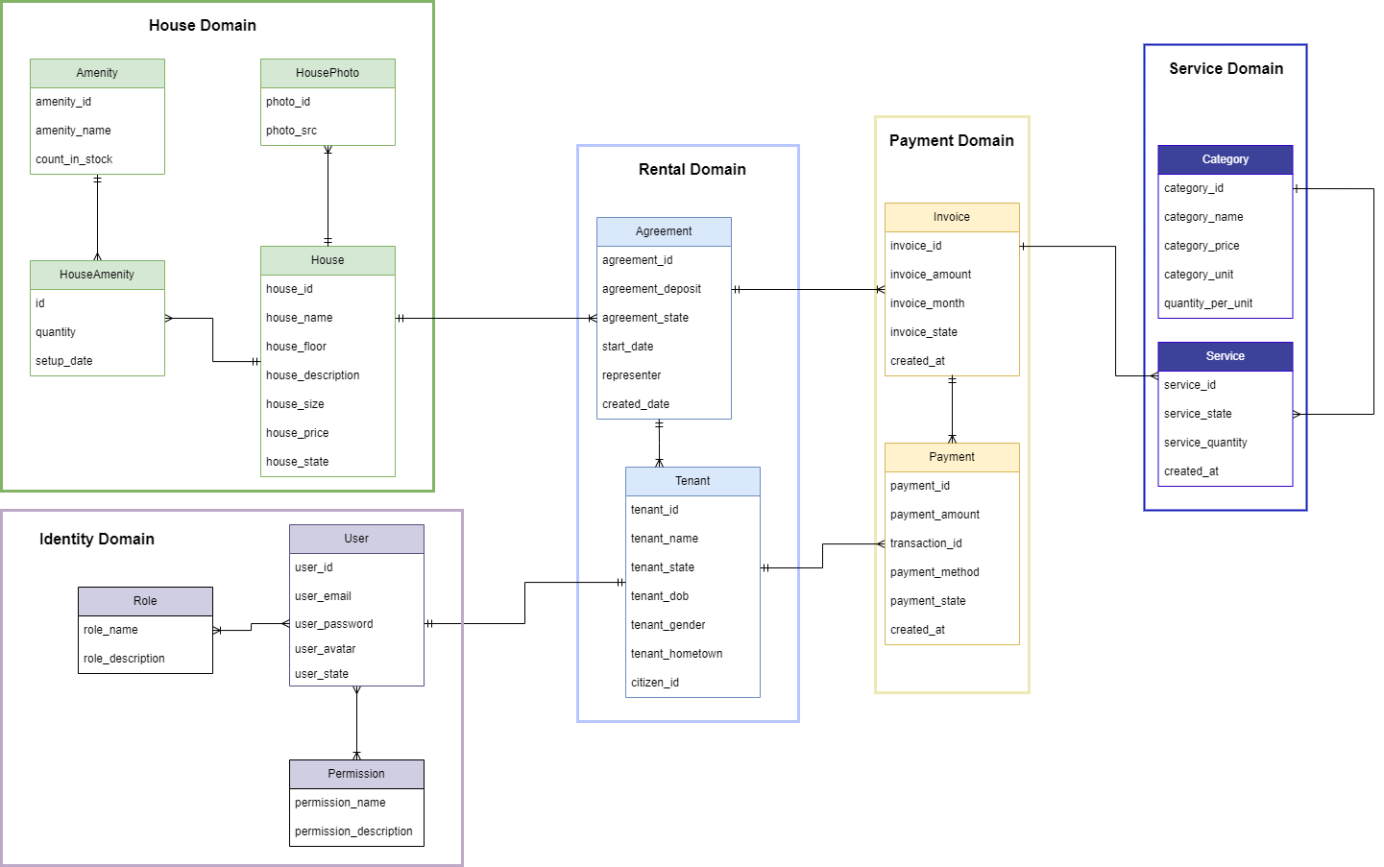
Hình 3.5 Sơ đồ tuần tự chức năng “Hoàn thành hóa đơn”

Sơ đồ tuần tự chức năng “Nhắn tin”:



Hình 3.6 Sơ đồ tuần tự chức năng “Nhắn tin”

### Thiết kế cơ sở dữ liệu



Hình 3.7 Lược đồ cơ sở dữ liệu quan hệ

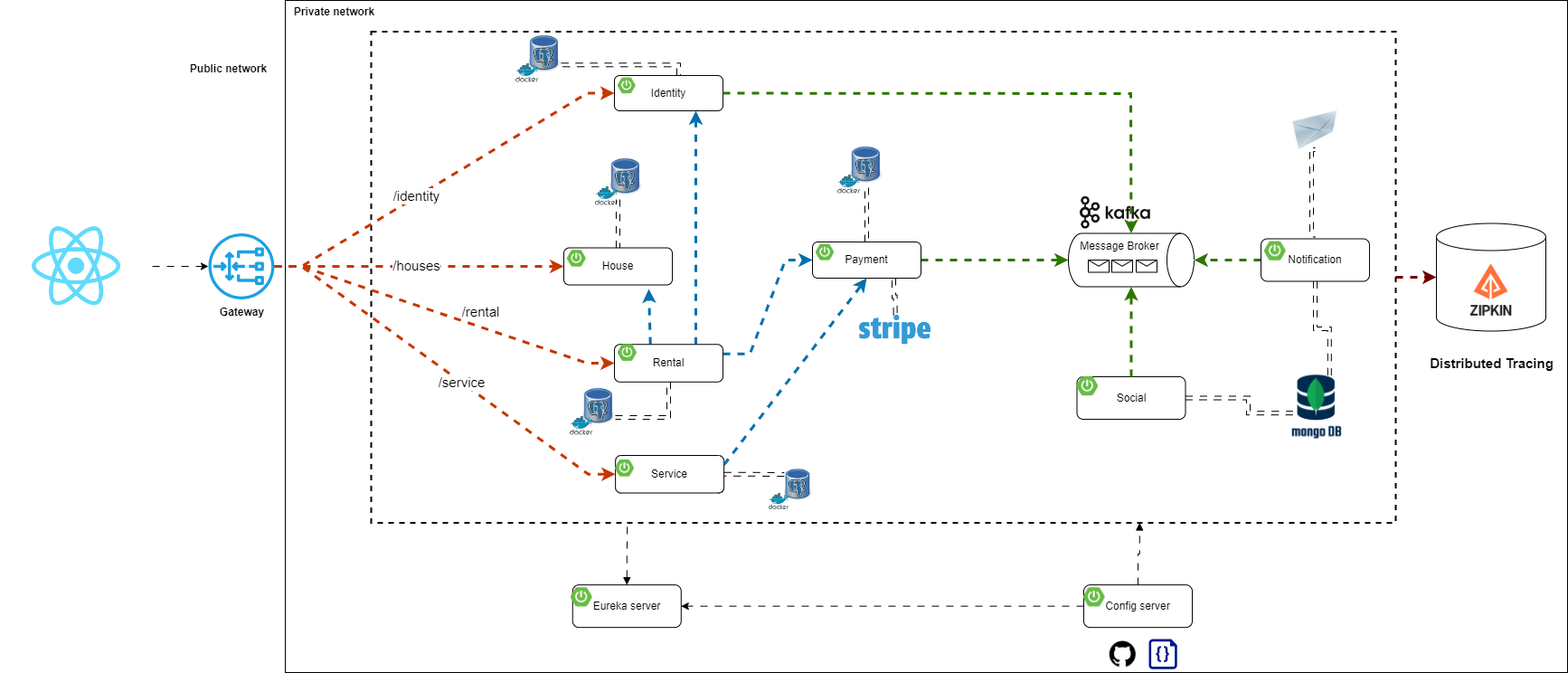
1. House domain
2. Rental domain
3. Identity domain
4. Payment domain
5. Service domain

### Thiết kế giao diện

## Kiến trúc hệ thống

1. Client (React): Phần frontend được xây dựng bằng React, giao tiếp với các dịch vụ backend thông qua các API. Nó cung cấp giao diện người dùng cho cư dân, quản trị viên, và các đối tượng khác, cho phép thực hiện các chức năng như đăng nhập, quản lý hợp đồng, thanh toán, và các dịch vụ khác.
2. Gateway: Gateway đóng vai trò như một cổng kết nối, tiếp nhận tất cả các yêu cầu từ client và phân phối chúng đến các microservice tương ứng. Nó cũng có thể thực hiện các chức năng như xác thực, cân bằng tải, và chuyển đổi giao thức.
3. Zipkin: là một công cụ quan trọng trong kiến trúc microservices của bạn, cung cấp khả năng theo dõi phân tán (distributed tracing). Nó giúp bạn theo dõi luồng yêu cầu từ client qua gateway và các microservice, cho phép phân tích thời gian phản hồi của từng service. Nhờ đó, bạn có thể dễ dàng phát hiện các tắc nghẽn và tối ưu hóa hiệu suất của hệ thống. Bên cạnh đó, Zipkin cũng hỗ trợ việc phát hiện lỗi bằng cách xác định vị trí mà lỗi xảy ra trong quá trình xử lý yêu cầu, từ đó cải thiện khả năng giám sát và bảo trì hệ thống.
4. Eureka server: Eureka đóng vai trò là một service registry (đăng ký dịch vụ), nơi các microservice có thể đăng ký và phát hiện nhau, giúp hệ thống linh hoạt trong việc mở rộng và duy trì kết nối giữa các dịch vụ.
5. Config server: Quản lý cấu hình tập trung cho tất cả các microservice. Nó cho phép mỗi service có thể lấy cấu hình của mình từ một nguồn duy nhất, hỗ trợ việc thay đổi cấu hình mà không cần khởi động lại hệ thống.
6. Các dịch vụ Spring Boot (Microservices):

* identity-service: Quản lý chứng thực và ủy quyền người dùng.
* house-service: Quản lý các phòng, tiện ích (điện, nước), và cơ sở vật chất của các tòa nhà.
* rental-service: Quản lý cư dân, hợp đồng thuê nhà, và các thủ tục liên quan.
* service-service: Quản lý các dịch vụ liên quan như điện, nước, phương tiện xe cộ,...
* payment-service: Quản lý các khoản thanh toán và hóa đơn, theo dõi việc thanh toán của cư dân.
* notification-service: Sử dụng Kafka để quản lý các thông báo gửi đến cư dân hoặc các quản trị viên, đảm bảo thông báo thời gian thực.
* social-service: Quản lý tính năng nhắn tin, post bài trong hệ thống, có thể hỗ trợ tương tác giữa cư dân với nhau hoặc với quản trị viên.



Hình 3.8 Kiến trúc hệ thống

## Kết quả thực hiện

# KẾ LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

## Hướng phát triển

TÀI LIỆU THAM KHẢO

(Theo chuẩn IEEE – *bỏ dòng này khi viết báo cáo*)

PHỤ LỤC