HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Hotel booking system

Nguyễn Trinh Vũ
vu.nt184332@sis.hust.edu.vn

Supervisor: PhD. Bùi Quốc Trung _____

Signature

Department: Information Technology

School: Information and Communications Technology

HANOI, 08/2023

ACKNOWLEDGMENTS

Firstly, I would like to send a sincere thanks to my supervisor **PhD Bùi Quốc Trung** for his exceptional guidance, unwavering support, and invaluable mentorship throughout my graduation thesis. Without him, it is impossible for me to do this thesis

Secondly, I would like to thank all the teachers of Hanoi University of Science and Technology and the School of Information and Communication Technology who have taught me since the day I started studying here. Their commitment to teaching and willingness to go above and beyond to ensure our understanding have inspired me to continually strive for excellence.

Finally, I would like to thanks to my family and friends for having always been standing by my side to encourage and support me throughout my 5-year student life.

Thank you all sincerely for being an integral part of my journey.

ABSTRACT

The traveling industry is experiencing a remarkable resurgence after the challenging period of the pandemic. As vaccination efforts progress and travel restrictions are lifted, there is a renewed sense of excitement among travelers. People are eager to embark on long-awaited vacations, reunite with loved ones, and explore new destinations. Realizing the potential to exploit the rental platform, I chose the topic of developing a hotel booking system.

The Hotel Booking Application utilizing Microservices Architecture, Spring Boot, and ReactJS is a modern and efficient solution designed to streamline the hotel reservation process for users. This application leverages the power of microservices to create a scalable, flexible, and resilient system, while Spring Boot and ReactJS provide the foundation for the backend and frontend components, respectively. The backend is developed using Spring Boot, a robust and developer-friendly framework that supports the microservices paradigm. Each microservice handles distinct functionalities, such as user authentication, hotel search, booking management, and payment processing. The communication between these microservices is efficiently managed, ensuring loose coupling and promoting system reliability.

On the frontend side, ReactJS is employed to build a dynamic and interactive user interface. The application provides users with an intuitive hotel search interface, real-time availability updates, and a secure payment gateway. ReactJS facilitates smooth user interactions and optimizes the application's performance, enhancing the overall user experience.

Moreover, the adoption of microservices allows the application to handle high traffic loads without compromising on performance. Load balancing and autoscaling capabilities can be seamlessly integrated into the system, ensuring optimal resource utilization during peak usage.

In conclusion, the Hotel Booking Application using Microservices Architecture, Spring Boot, and ReactJS presents a modern, scalable, and user-friendly solution for hotel reservations. By leveraging the strengths of microservices and the power of Spring Boot and ReactJS, the application meets the demands of a rapidly evolving market and offers users a seamless hotel booking experience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBRIVIATIONS

Abriviation	Full Expression
API	Set of defined rules that enable different applications to communicate with each other (Application Programming Interface).
ERD	Visual representation of the relationships between entities in a database. It helps to model and understand the structure of a database and the relationships between different tables or entities (Entity Relationship Diagram).
IDE	Software application that provides comprehensive facilities for software development (Integrated Development Environment).
JWT	Secure token format used for transmitting information between parties. It's commonly used for authentication and authorization in web applications (JSON Web Token).
UI	It refers to the visual and interactive elements of a software application or website that users interact with (User Interface).

CHAPTER 1. INTRODUCTION

1.1 Motivation

In today's digital age, travelers are increasingly reliant on technology to plan and manage their trips. The traditional approach of booking hotels through phone calls or travel agencies no longer aligns with the expectations of modern travelers. The motivation to develop a scalable hotel booking application arises from the need to cater to the tech-savvy and mobile-first generation of travelers who seek quick and hassle-free hotel reservations.

The development of a hotel booking application is driven by the desire to provide travelers with a seamless and intuitive user experience. The application empowers users to explore a vast array of hotels, view real-time availability, compare prices, and read authentic reviews, all within a single platform. By simplifying the booking process and offering transparent information, the application aims to enhance customer satisfaction and engagement.

From the perspective of hotel owner, the motivation for developing a hotel booking application lies in streamlining hotel management processes. The application centralizes reservation management, automates payment processing, and facilitates efficient communication with guests. This optimization of hotel operations leads to improved guest satisfaction and resource utilization.

In conclusion, the development of a hotel booking application is driven by the desire to cater to the changing needs of travelers, provide a seamless user experience, optimize hotel operations. As technology continues to evolve, the hotel booking application remains a vital tool in shaping the future of the travel industry and delivering unparalleled experiences to both travelers and hotel establishments.

1.2 Objectives and scope of the graduation thesis

The objectives of the hotel booking system are to create a user-friendly and efficient platform that streamlines the booking process for travelers. The system aims to offer a wide selection of accommodations, including hotels, resorts, apartments, and more, catering to diverse traveler preferences and budgets. Real-time updates on room availability, prices, and promotions will ensure accurate and up-to-date information for users. Integration of authentic customer reviews and ratings will empower travelers to make informed decisions.

The scope of the hotel booking system includes user registration and profiles, extensive accommodation listings, advanced search and filtering options, booking

management. By achieving these objectives and encompassing this scope, the hotel booking system aims to become a reliable and user-centric platform for travelers worldwide.

1.3 Tentative solution

A tentative solution for developing the hotel booking system involves using a combination of ReactJS for the frontend, Spring Boot for the backend, and implementing a microservices architecture. ReactJS provides a fast and efficient way to create dynamic and interactive frontend components. With ReactJS, we can develop a responsive and user-friendly booking interface that allows users to search, compare, and book accommodations seamlessly.

Spring Boot is a powerful Java-based framework that simplifies the development of backend applications. It provides essential features for building robust and scalable web services. With Spring Boot, we can create a secure and efficient backend to handle user requests, manage hotel information, and process bookings.

The microservices architecture is a software design approach that structures the application as a collection of small, independent services, each focused on specific functionalities. In the context of a hotel booking system, each microservice can handle separate tasks, such as user authentication, hotel search, booking management and user profiles.

By combining ReactJS, Spring Boot, and a microservices architecture, we can create a modern and robust hotel booking system that provides an excellent user experience and efficiently manages bookings, hotel data, and user interactions. This solution ensures a separation of concerns, making the development process more manageable and enabling us to evolve and scale the system as needed.

1.4 Thesis organization

The layout of this graduation project is organized as follows:

- Chapter 2, we make a presentation on surveying the current status of booking platforms; analyze the actors and basic functions of sharing platform.
- Chapter 3, we introduce the technologies and frameworks applied.
- Chapter 4, we mentioned software architecture, implementation of layer design, system database design, interface design. From the analysis and design in chapters 3 and 4 as a premise for building the system and go to test the program, evaluate the performance of the system.
- Chapter 5, we presents a summary of the graduation project, the results achieved

and outlines the future development direction for the current project.

Finally, the list of references.

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS

2.1 Status survey

Nowadays, there are many approaches that help users rerserve hotels when they travel. Some of the most popular ways people often use are direct booking systems from hotel websites and hotel aggregators such as Booking.com and Agoda.

Direct booking systems from hotel websites offer several advantages to both travelers and hotels. One significant benefit is the direct interaction with the hotel, eliminating intermediaries and ensuring accurate information about room prices and availability. This transparency can instill confidence in travelers and foster trust in the booking process. Moreover, hotels can leverage this direct connection to offer exclusive deals and promotions, rewarding loyal customers and attracting repeat business. Additionally, the convenience of tracking booking details and making special requests directly with the hotel enhances the overall experience for travelers, as it allows for personalized communication and quick resolution of any concerns.

However, there are certain drawbacks to consider. Direct booking systems may have limited options for comparing prices and room choices from multiple hotels, potentially leading to less competitive pricing. Additionally, some hotel websites may lack integrated customer reviews and ratings, making it challenging for travelers to gauge the quality of accommodations accurately. Lastly, the user interface and overall experience on hotel websites might not be as optimized and seamless as those offered by large online booking platforms. This could impact user convenience and deter potential customers who prefer a more streamlined and efficient booking process.

Besides the direct approach, the use of hotel aggregator platforms for booking accommodations presents both advantages and drawbacks. One of the key benefits is the ease of search and comparison, as travelers can effortlessly explore room prices from numerous hotels all in one place. This allows for better decision-making by offering a variety of options, including various room types, amenities, and locations. Moreover, the integration of customer reviews and ratings on these platforms empowers users to make well-informed choices when selecting their ideal stay. Additionally, many aggregator platforms frequently provide attractive promotions, discounts, and exclusive deals, making the booking process even more enticing for customers.

On the other hand, the integration of multiple hotels on a single platform can

result in a complex and slow-loading interface, which might frustrate some users seeking a seamless and efficient experience. Furthermore, while aggregator platforms strive to offer competitive prices, the best deals are not always guaranteed as they depend on partnerships with hotels. Additionally, some travelers may be reluctant to proceed with booking due to the requirement of user registration or providing personal information, which can be perceived as time-consuming or intrusive. Lastly, despite the convenience of booking through an intermediary, it might limit direct access to hotels, which could hinder travelers who prefer direct communication or specific arrangements with the accommodation.

From the above analysis, we have recognized that hotel aggregator platforms is the optimal solution. By continuously focusing on scalability, global reach, and staying at the forefront of industry trends, we are confident that this approach will set the foundation for a successful and customer-centric booking platform. Embracing a user-centric mindset, I aspire to create a booking system that not only streamlines the travel planning process but also becomes a reliable companion for travelers, offering them the best deals and tailored experiences.

2.2 Functional Overview

Key features of the software include: authentication and reserve hotels, where authentication helps identify the actor that is using the system.

2.2.1 General use case diagram

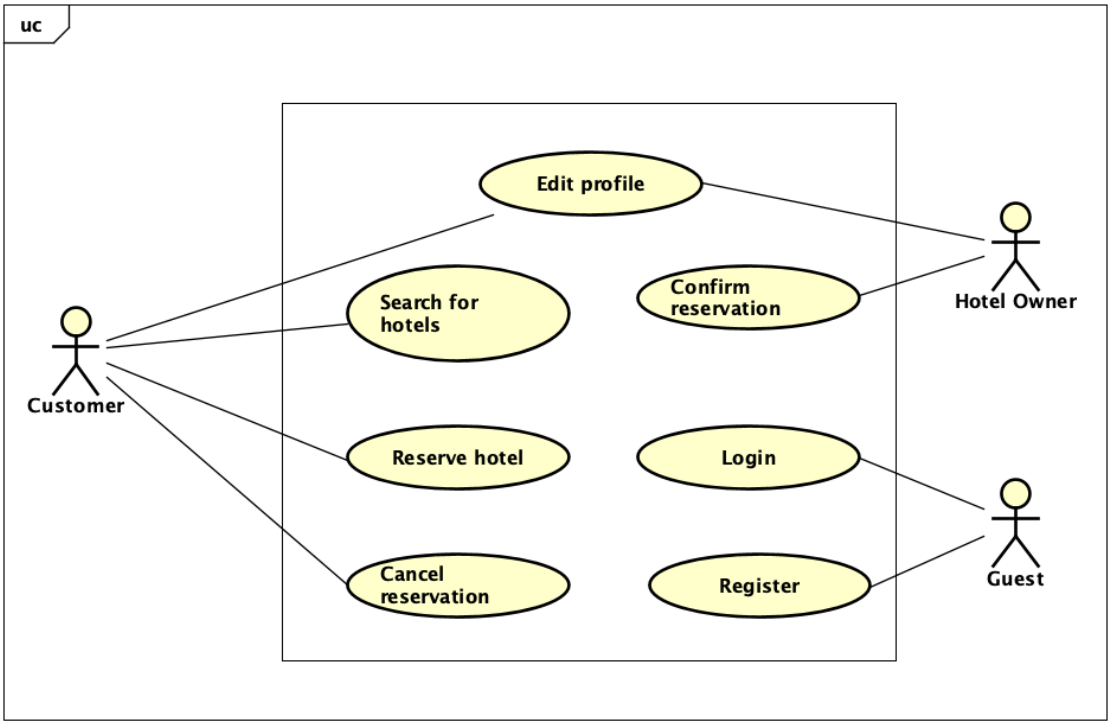

Figure 2.1: General use case diagram

The main actors are Customer and Hotel Owner. Specifically, the system actors and their roles are as follows:

- Guest: is an object that, when not logged in to the website, has limited rights to interact with the website. Guests can manipulate registration and login.
- Customer: is the object after successfully logging in to the website as customer role. Customers can edit their profile, search for hotels, reserve hotels and cancel the reservations.
- Hotel owner: is the object after successfully logging in to the website as customer role. Hotel owners can edit their profile and confirm reservations from customers.

2.3 Functional description

2.3.1 Description of use case "Register"

Use case code	UC01	Use case name	Register
Actor	Guest		
Intended use	Guest create account		
Precondition	No		
Main event flow	1. The guest chooses the registration function. 2. The system displays the registration interface. 3. Guest enters required fields and executes show registration request operation. 4. The system checks the required fields.		
Alternative event flow	4a. The system notifies to fill in the missing fields.		
After-condition	Redirect to login page		

Table 2.1: Register specification table

2.3.2 Description of use case "Login"

Use case code	UC02	Use case name	Login
Actor	Guest		
Intended use	Guest login account		
Precondition	No		
Main event flow	1. The guest chooses the login function. 2. The system displays the login interface. 3. Guest enters required fields and executes show login request operation. 4. The system checks the required fields. 5. The system checks user credential.		
Alternative event flow	4a. The system notifies the missing fields. 5a. The system notifies mismatched email or password information.		
After-condition	Redirect to the main page		

Table 2.2: Login specification table

2.3.3 Description of use case "Search hotels"

Use case code	UC03	Use case name	Search hotels
Actor	Customer		
Intended use	Customers search for hotels		
Precondition	Login successfully		
Main event flow	1. The customer chooses search function. 2. The customer chooses values of the filter. 3. The customer chooses search. 4. The system displays appropriate hotels.		
Alternative event flow	4.1. The system notitfies no appropriate hotels.		
After-condition	No		

Table 2.3: Search hotels specification table

2.3.4 Description of use case "Reserve hotels"

Use case code	UC04	Use case name	Reserve hotels
Actor	Customer		
Intended use	Customers reserve hotels		
Precondition	Login successfully		
Main event flow	1. The customer chooses one room. 2. The customer chooses book now button. 3. The customer chooses check-in and check-out date and confirm. 4. The system add reservation and displays the bill.		
Alternative event flow	No		
After-condition	No		

Table 2.4: Search hotels specification table

2.3.5 Description of use case "Cancel reservation"

Use case code	UC05	Use case name	Cancel reservation
Actor	Customer		
Intended use	Customers cancel reservations		
Precondition	Login successfully		
Main event flow	1. The customer chooses cancel one room from history table. 2. The system display popup to confirm cancellation. 3. The customer chooses confirm. 4. The system set the reservation as cancelled.		
Alternative event flow	3.1. The customer close the popup without cancelling.		
After-condition	No		

Table 2.5: Search hotels specification table

2.3.6 Description of use case "Confirm reservation"

Use case code	UC06	Use case name	Confirm reservation
Actor	Hotel owner		
Intended use	Hotel owner confirm reservations		
Precondition	Login successfully		
Main event flow	1. The owner chooses cancel one room from history table. 2. The system display popup to confirm approval. 3. The owner chooses confirm. 4. The system set the reservation as approved.		
Alternative event flow	3.1. The owner close the popup without approving.		
After-condition	No		

Table 2.6: Search hotels specification table

2.3.7 Description of use case "Edit profile"

Use case code	UC06	Use case name	Login
Actor	Customer or Hotel owner		
Intended use	Guest login account		
Precondition	No		
Main event flow	1. The customer/ owner chooses the edit function. 2. The system displays the edit interface. 3. Guest enters fields and executes save. 4. The system checks the required fields. 5. The system save the information.		
Alternative event flow	4.1. The system notifies the missing fields.		
After-condition	no		

Table 2.7: Edit profile table

2.4 Non-functional requirement

2.4.1 Security requirements

- Each actor (guest, user) participating in the system is only allowed to request access to appropriate resources.
- Actor guest can only participate in the registration and login functions.
- Ensure server transparency for users' personal data.
- The user's password must be encrypted, do not save the password as "text".

2.4.2 Performance requirements

- Maximum system response time is 1 second (delay is greater than 1 seconds should have a loading state on the UI).
- Make sure the system can handle 4000 connections per minute.
- The notification features ensure real-time speed.

2.4.3 Interface requirements

- User-friendly interface.
- The interface displays well on the Desktop.

CHAPTER 3. ADOPTED TECHNOLOGIES AND FRAMEWORKS

3.1 Spring boot

Spring Boot is a powerful and innovative framework for building Java-based applications that has gained widespread popularity in the software development community. Developed by the Pivotal team, Spring Boot aims to simplify the process of creating production-ready, stand-alone, and web-based applications. It accomplishes this by providing a convention-over-configuration approach, enabling developers to focus on writing business logic rather than dealing with boilerplate code and complex configurations.

Spring Boot takes the hassle out of setting up and configuring a new Spring-based project. It offers a variety of auto-configuration features that automatically set up the application based on its dependencies. Developers can simply declare the required dependencies, and Spring Boot takes care of the rest, making development faster and more efficient.

Spring Boot promotes the concept of self-contained, stand-alone applications. It includes an embedded server (like Tomcat, Jetty, or Undertow) so that the application can run independently without requiring any external server setup. Additionally, Spring Boot follows the "opinionated" approach, which means it adopts sensible defaults and best practices, reducing the need for manual configuration.

Spring Boot is an excellent choice for building microservices and cloud-native applications. Its lightweight nature and minimal setup allow for faster development and deployment of microservices. Moreover, Spring Boot integrates seamlessly with other Spring projects, such as Spring Cloud, to support cloud-native architectures and microservices communication patterns.

Spring Boot provides an intelligent dependency management system. It offers a curated list of compatible dependencies and their versions, reducing the likelihood of version conflicts and ensuring a stable and reliable application.

Spring Boot has a vibrant and active community, making it easier for developers to find support, tutorials, and solutions to common problems. Additionally, it integrates seamlessly with a wide range of third-party libraries and frameworks, further expanding its ecosystem and flexibility.

3.2 ReactJs

ReactJS, commonly referred to as React, is a popular and powerful JavaScript library for building user interfaces. React's main goal is to create interactive and

dynamic user interfaces for web applications, making it the foundation for building modern, responsive, and user-friendly websites.

React introduces a declarative programming paradigm, where developers describe how the user interface should look and behave based on the application's current state. This approach allows for more straightforward and intuitive code, as developers no longer need to manage the manipulation of the DOM (Document Object Model) directly. Instead, they create reusable components that encapsulate their own logic and rendering, promoting modular and maintainable code.

One of the key innovations of React is its Virtual DOM. React creates an in-memory representation of the actual DOM, allowing it to efficiently compare and update only the necessary parts of the UI when the application's state changes. This Virtual DOM mechanism significantly improves the rendering performance, resulting in smoother and faster user interactions.

React's component-based architecture encourages code reusability. Developers can create encapsulated components that handle specific functionality or UI elements. These components can then be reused throughout the application, promoting a more efficient and organized development process.

3.3 RabbitMQ

RabbitMQ plays a crucial role in the context of microservices architecture, where it serves as a powerful and scalable messaging broker to facilitate communication and coordination among the different microservices. Its asynchronous messaging capabilities and message queuing system make it an ideal choice for handling the complexities of microservices-based applications.

In the RabbitMQ ecosystem, applications are categorized as "producers" and "consumers." Producers create and send messages to RabbitMQ, while consumers receive and process these messages. This separation of roles ensures that applications can work independently and asynchronously, making it ideal for building loosely coupled and distributed systems.

Microservices often need to handle tasks that take varying amounts of time to complete. RabbitMQ enables asynchronous communication, allowing services to send messages and continue their work without waiting for a response. This approach improves overall system responsiveness and resource utilization, as services can process other tasks while waiting for long-running operations to complete.

RabbitMQ offers message persistence, ensuring that messages are stored on disk to prevent data loss even if the broker or consumers experience failures. Durability

is vital for guaranteeing the reliable delivery of messages in scenarios where data integrity is crucial.

Conclusion

In this chapter, I have explained all the technologies and frameworks adopted in this project. The use of Spring Boot has streamlined the development process, while ReactJS has delivered an engaging and dynamic user interface. Meanwhile, RabbitMQ has enabled seamless communication and efficient message handling, supporting the application's performance and responsiveness. The synergy between these frameworks has resulted in a modern and scalable system, providing users with a user-friendly and efficient hotel booking experience.

CHAPTER 4. EXPERIMENT AND EVALUATION

4.1 Architecture design

4.1.1 Software architecture selection

Throughout the process of building the system, I followed the micro-service architecture model. A three-tier architecture is a software application architecture designed to organize an application, dividing it into tiers that run on its own infrastructure, each of which can be developed concurrently by a development team without affecting other layers. The three-tier architecture in Figure 4.1 includes the following components:

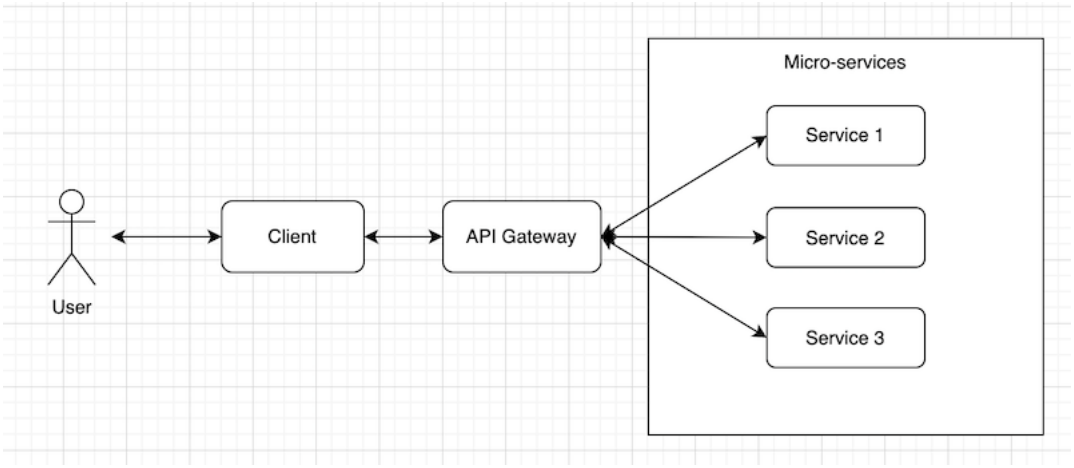

Figure 4.1: Micro-service architecture

Based on the three-tier architecture, Figure 4.2 depicts the system architecture in my project. Users interact with the system via the client UI. All requests from the client to the server has to go through API gateway and be verified by authentication service. The hotel service processes tasks associated with hotel entity such as get information, reserve hotel... the user service handle tasks about user information.

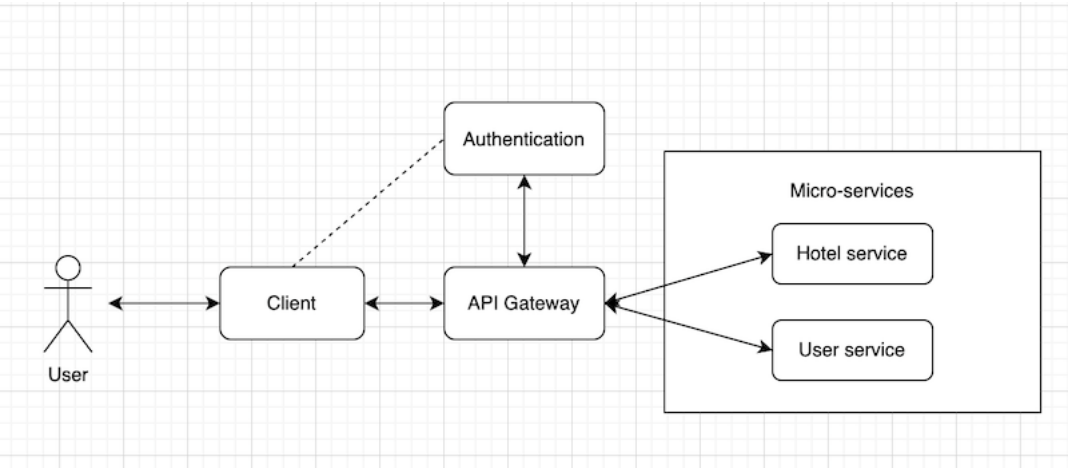

Figure 4.2: Application Architecture

4.1.2 Overall design

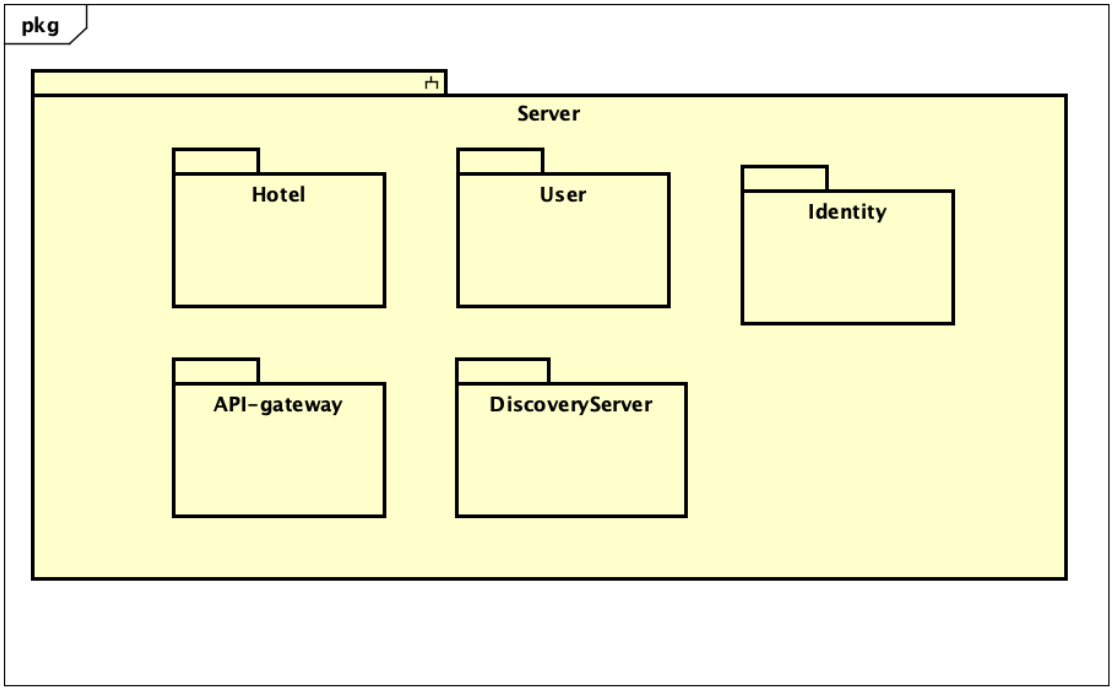

Figure 4.3: Package diagram

The system server has 5 packages which are API-gateway, DiscoveryServer, Indentity, Hotel and User. API-gateway takes responsibility of receiving request from user interface then routes the requests to the other packages. Discovery server is a critical component that facilitates communication and coordination between microservices. It acts as a central repository where microservices can register themselves and discover information about other available services within the system. Indentity packages helps the system authorize and authenticate users. Hotel and user package provides services associated to hotel and user profile respectively.

4.1.3 Detailed package design

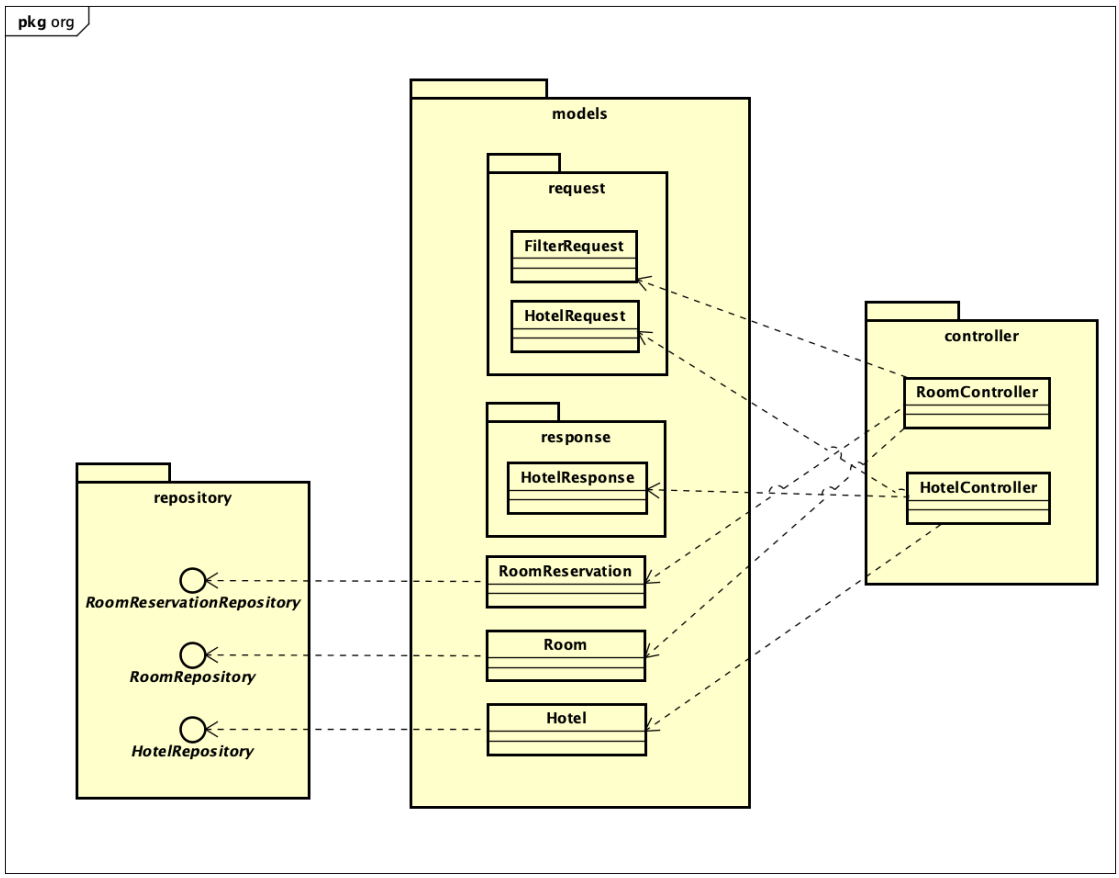

Figure 4.4: Hotel package diagram

The figure 4.4 describes detailed package design of the hotel package. This package contains 3 main packages which are controller, models and repository. The repositories are responsible for handling data access and storage operations. The models represents the application's data and business logic. They defines the structure of the data and includes methods to manipulate and access that data. In models package, we have request and response packages that contains classes that carry alternative data so that the service can work effectively with other services as well as user interface. Controllers act as intermediary between the user interface and the application's logic (model).

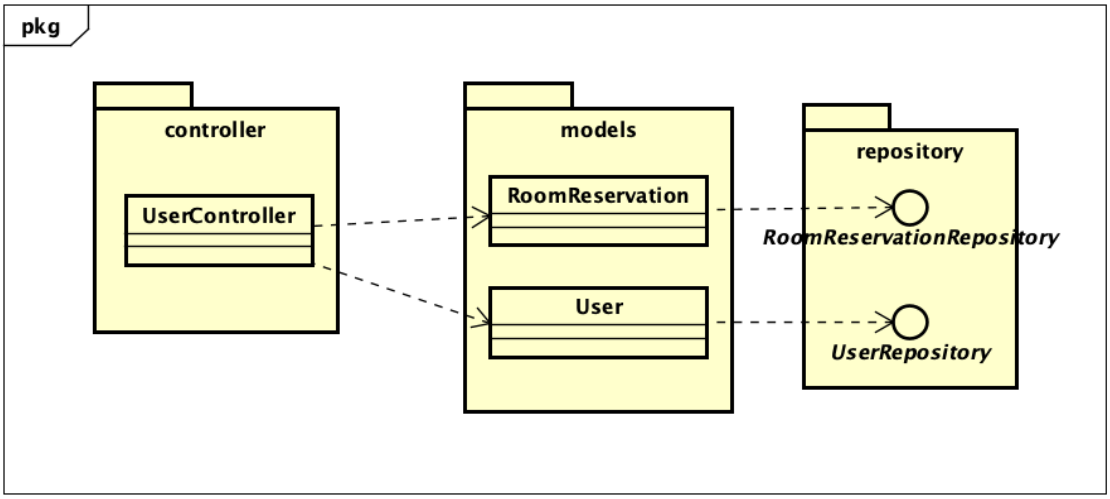

Figure 4.5: User package diagram

Similar to the hotel package, the user package also has three main packages controller, models and repository to handle tasks about user profile such as view and edit profile, view history...

4.2 Detailed design

4.2.1 User interface design

a, UI design "Login page"

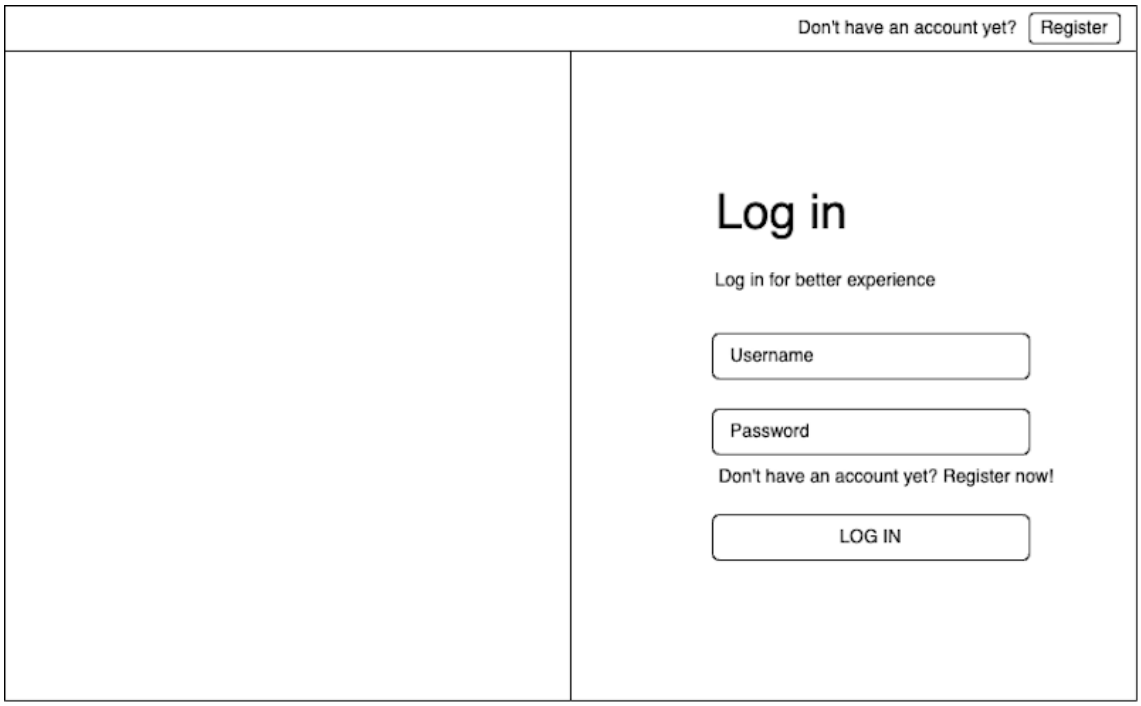

Figure 4.6: Login page UI design

Figure 4.6 showcases the initial design for the login page. There is a form for user to input and login. There are buttons to navigate to register page.

b, UI design "Login page"

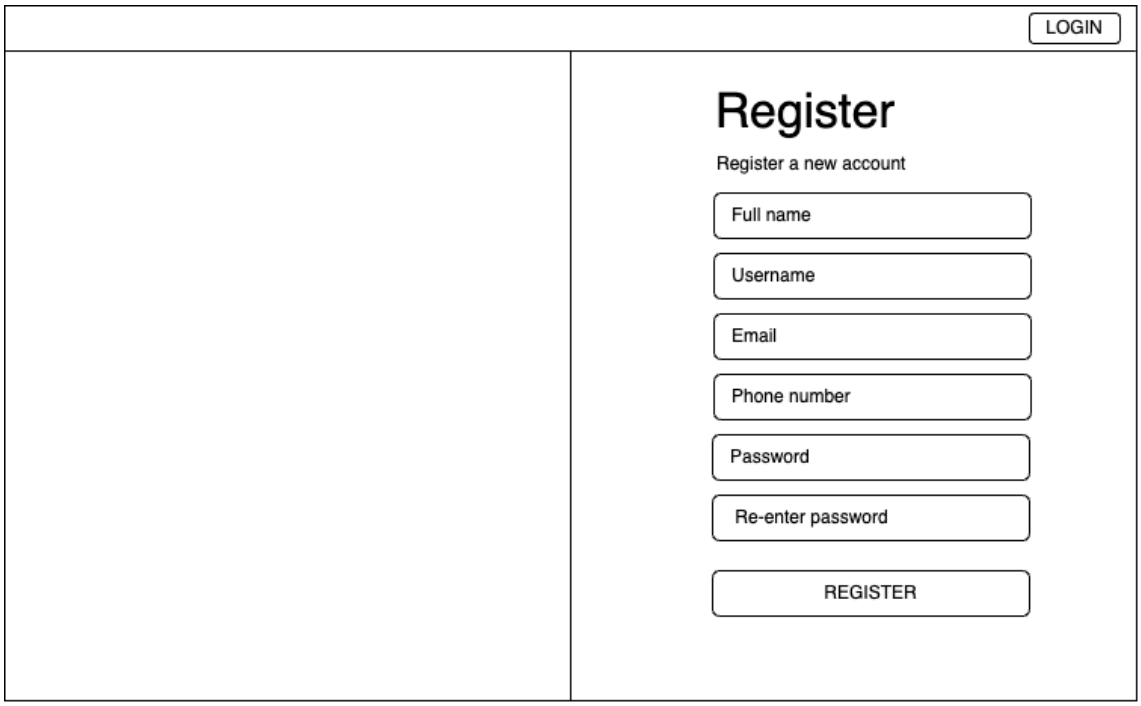

Figure 4.7: Register page UI design

Similarly to the Login UI, the register page UI is shown as in Figure 4.7.

c, UI design "Profile page"

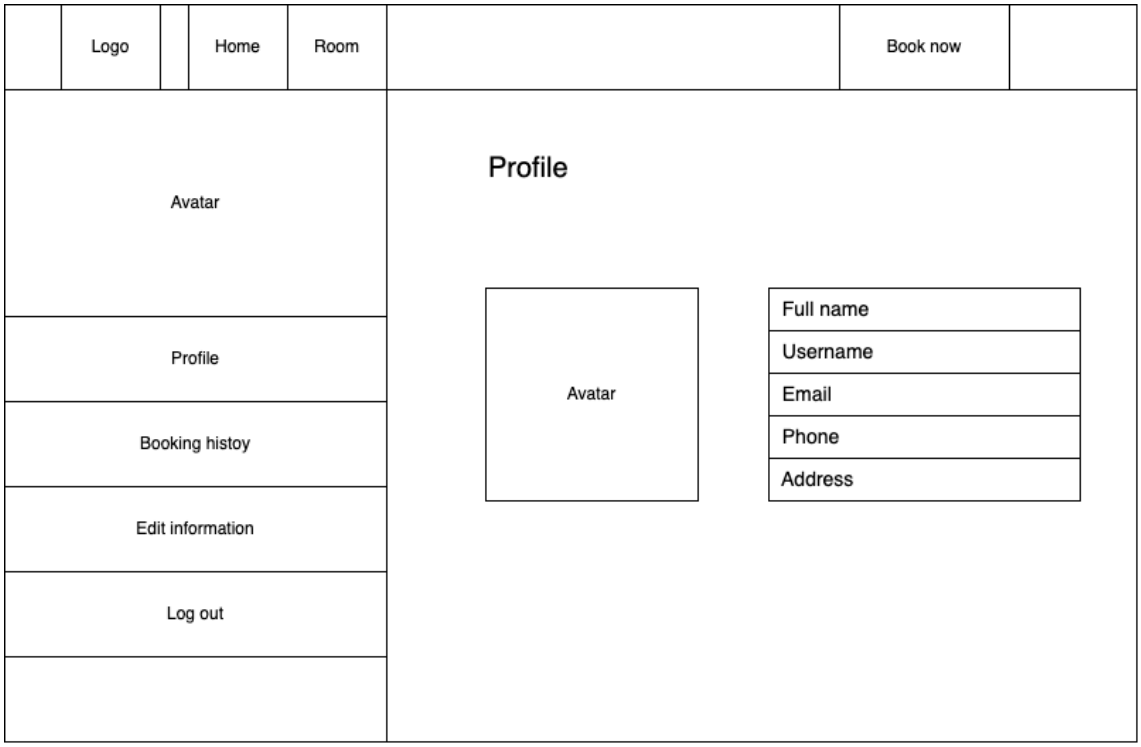

Figure 4.8: Profile UI design

Figure 4.8 presents the initial design for the Profile page. There is a header on top that contains buttons to navigate to home page or room list page. On the left of the page, there is a side bar with buttons to navigate to Booking history, edit information or log out. On the right is the information of user included full name, username, email, phone and address.

d, UI design "Booking history page"

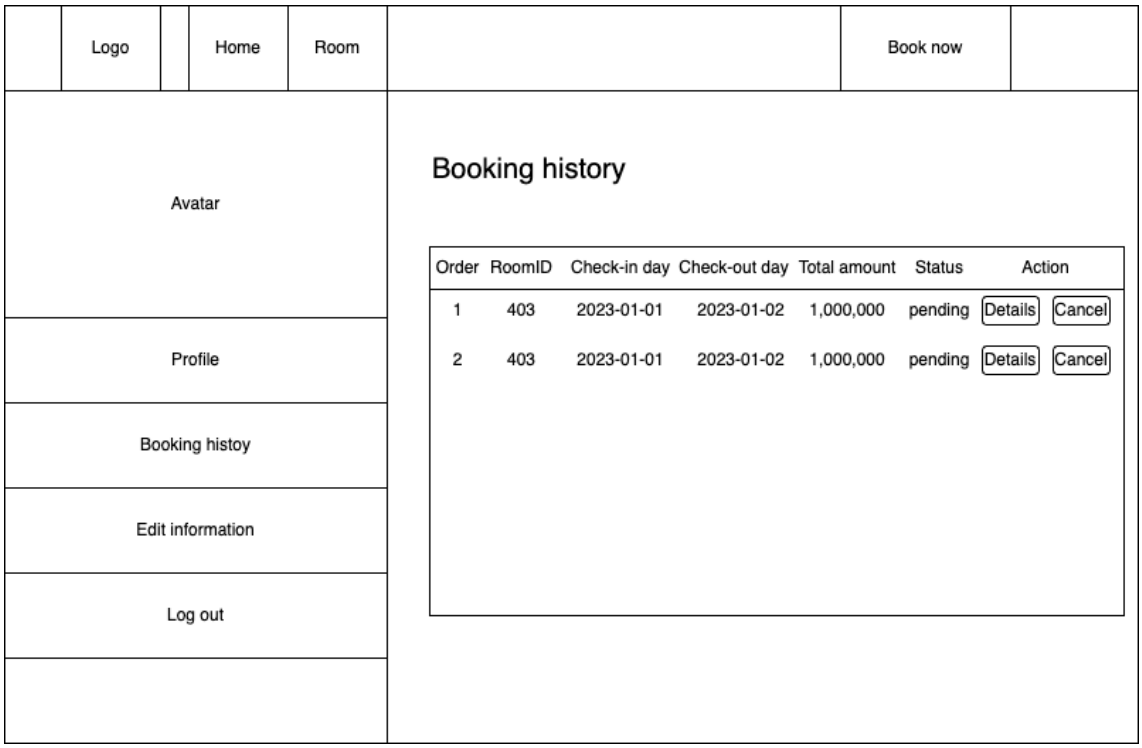

Figure 4.9: Booking history page UI design

Figure 4.9 displays the initial design for the history page. Besides the header and sidebar, this page has a table illustrate reservation history. Button detail to see detail information of owner.

e, UI design "Edit information page"

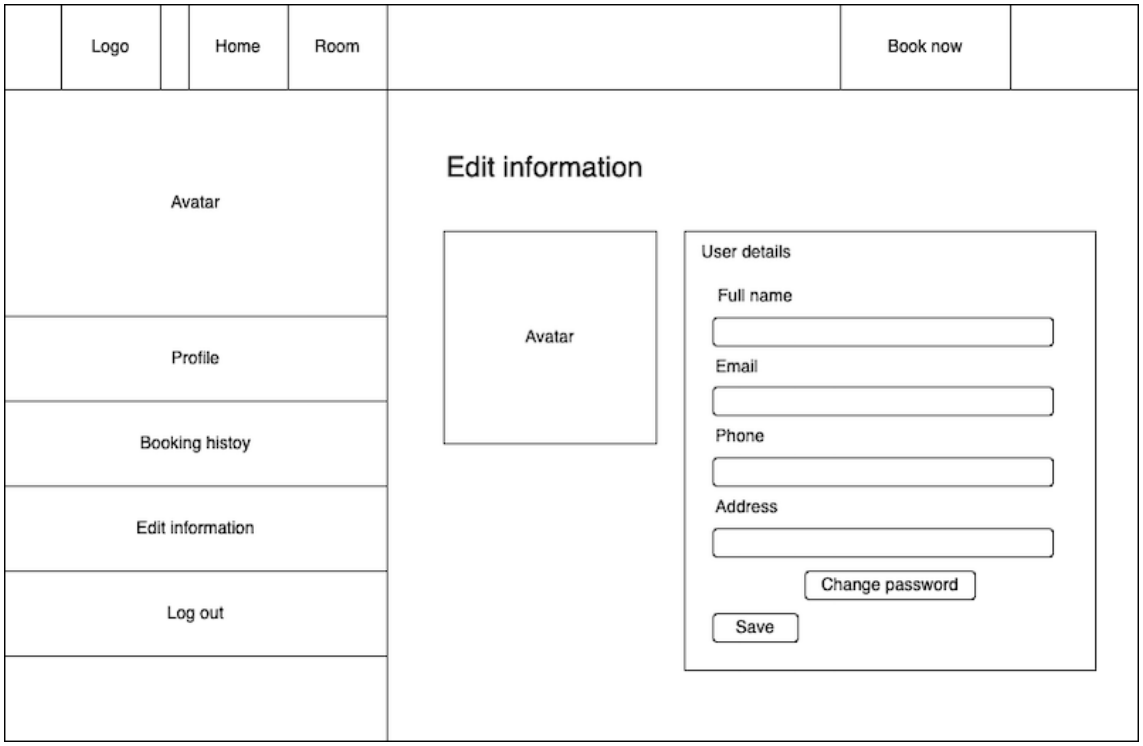

Figure 4.10: Edit information page UI design

Figure 4.10 displays the initial design for the edit information page. There is a form for user to input new information and a button to change password.

f, UI design "Search result page"

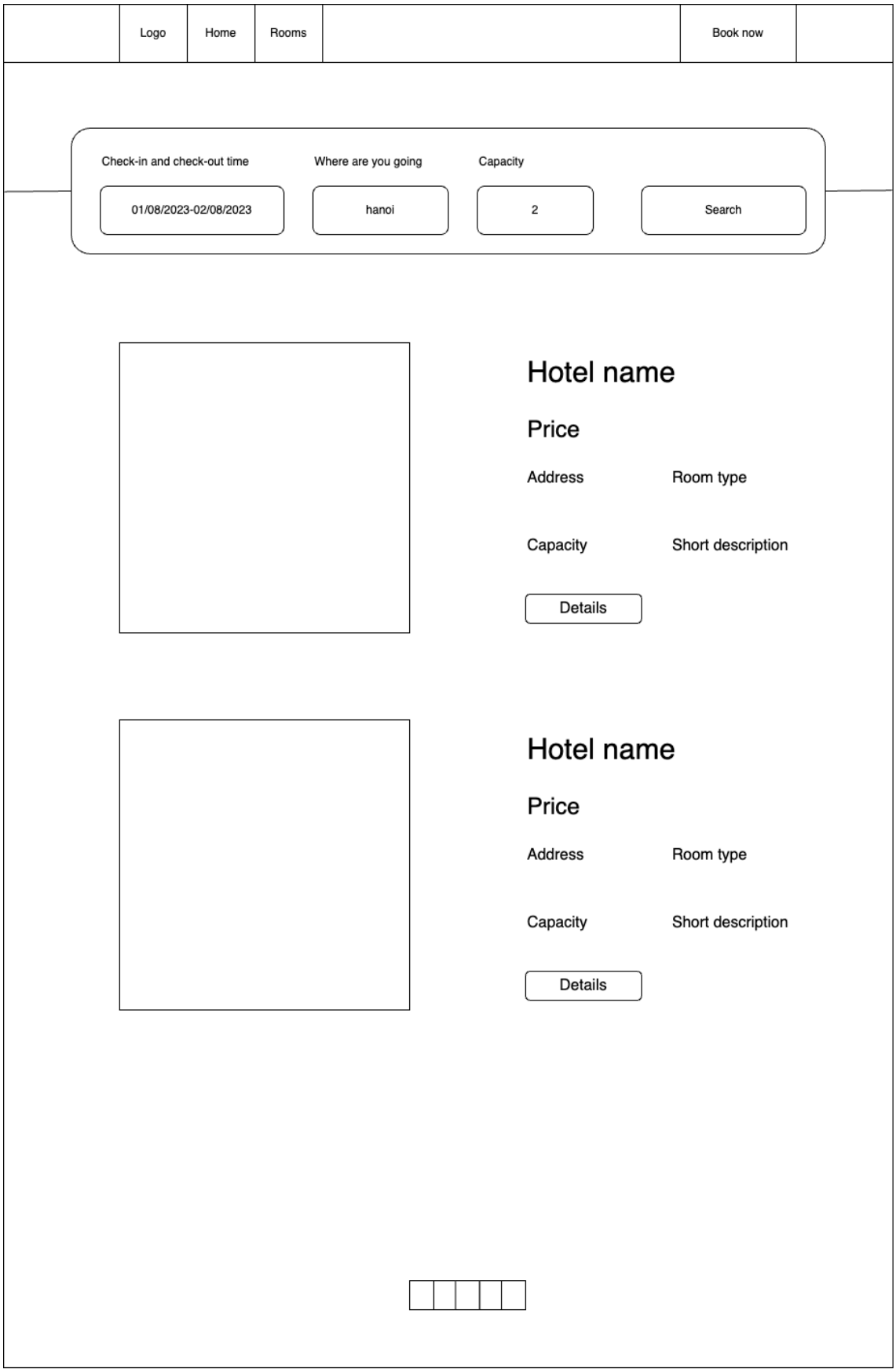

Figure 4.11: Search result page UI design

Figure 4.11 displays the initial design for the edit information page. This page tells user all details information of a room. There are button Book now to make a reservation, add to favorite to add to customer's favorite list. This page also recommend customers different options by list other rooms of the same hotel. There is a search form on the right of the page.

g, UI design "Room details page"

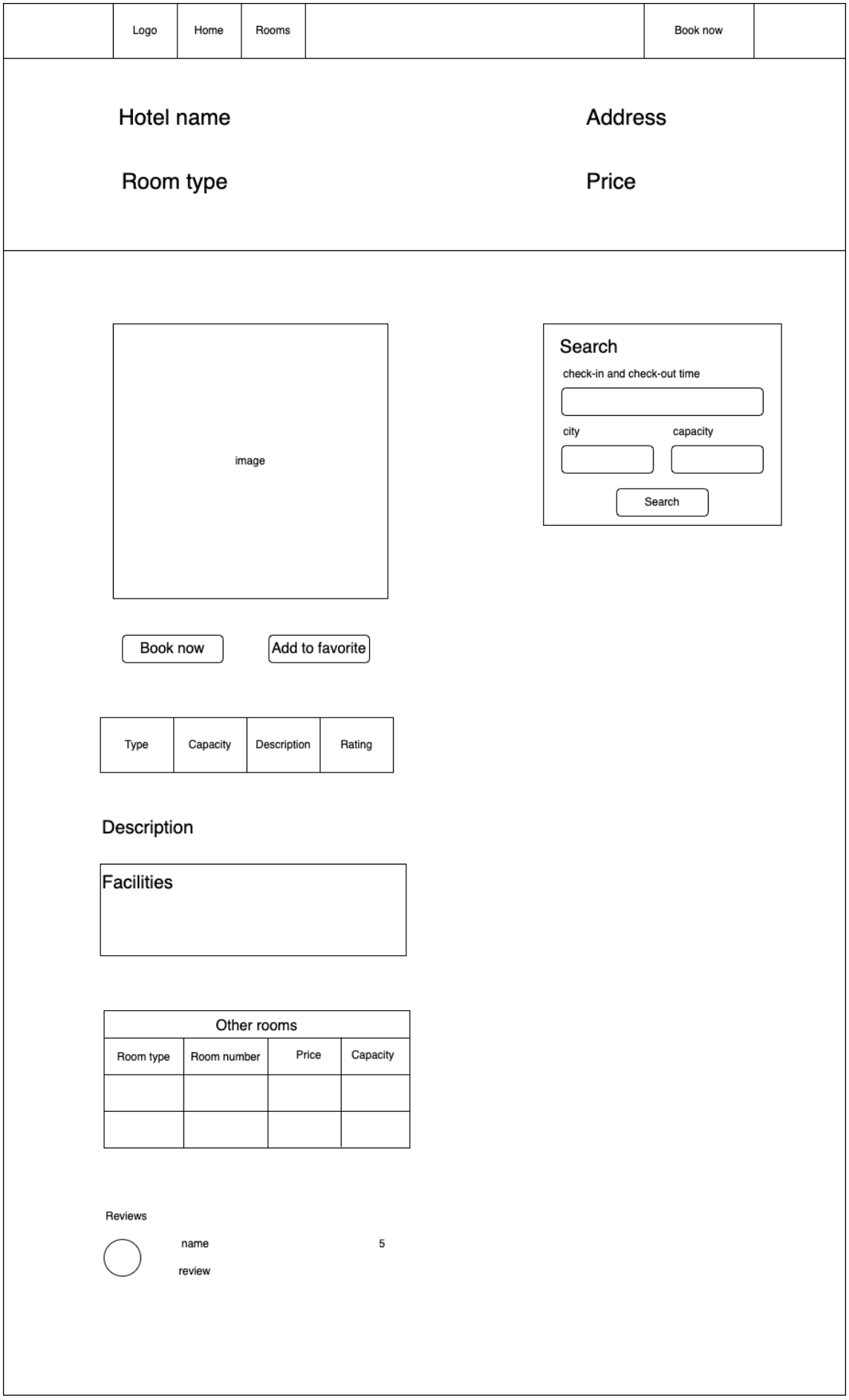

Figure 4.12: Room details page UI design

Figure 4.12 displays the initial design for the edit information page. On top of the page is the form to filter the hotel. The hotel list will be paginated and displayed 10 item per page. The interface will display basic information such as name, price, address, type, image, capacity, description. Details button to navigate to the room details page.

4.2.2 Layer design

a, Sequence diagram "Register"

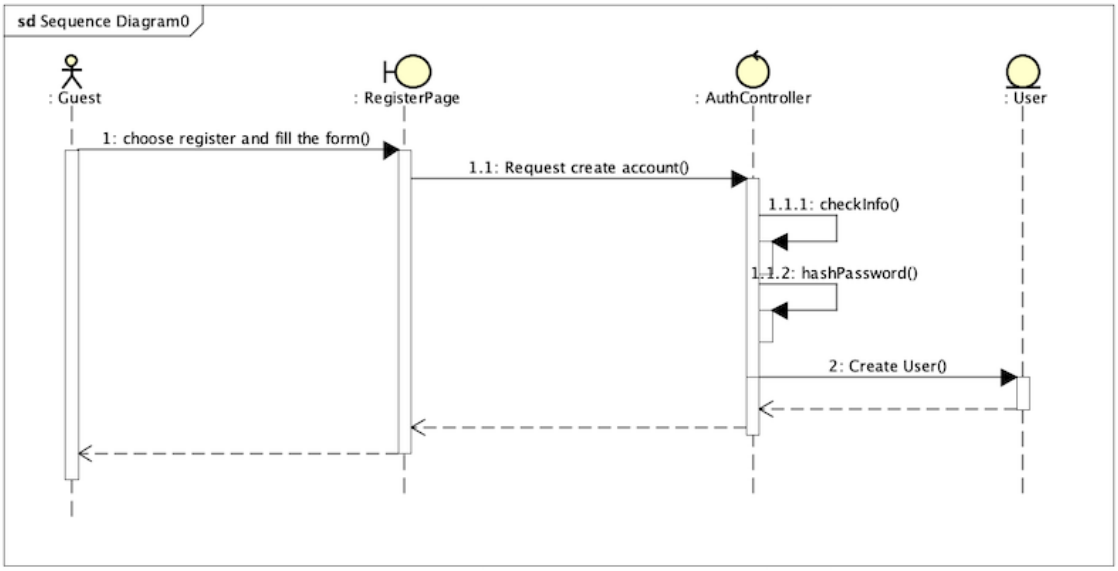

Figure 4.13: Register sequence diagram

The objects participating in the interaction are the actor(Guest), the registration page interface (RegisterPage), the authentication control(AuthController), user entity(User).

Event flow:

1. The Guest selects the registration function and fills out the registration form.
2. A request is sent to the Authentication control for information validation.
3. The Authentication control performs the following validation steps:
   a. Check for required information in the registration form.
   b. Check for duplicate accounts to avoid creating duplicate user entries.
   c. Hash the user's password for secure storage.
4. Once the validation is complete, the Authentication control sends a request to the user entity to create a new account.
5. If the account creation is successful, the user is redirected to the login page.

b, Sequence diagram "Login"

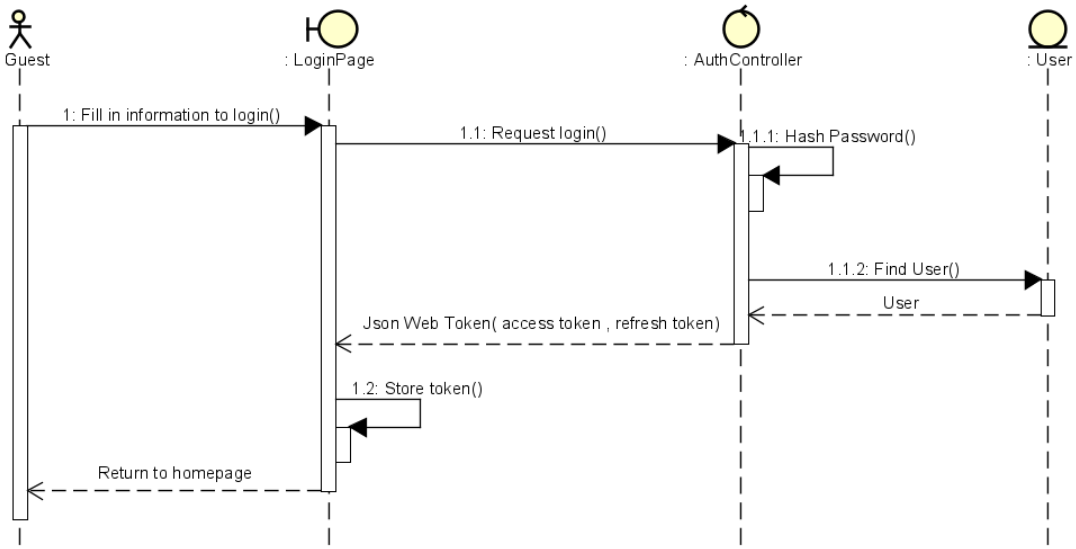

Figure 4.14: Login sequence diagram

The objects participating in the interaction are the actor(Guest), the login page interface (LoginPage), the authentication control (AuthController), user entity(User).

Event flow:

1. The Guest selects the login function and fills out the login form.
2. The login request is sent to the Authentication control, which performs information validation through the following steps:
   a. Hash the user's password.
   b. Verifying if the account meets any records in the system.
3. After successful validation, the control sends a request to the user entity to retrieve the account information.
4. Once the matching user information is found, the authentication control signs the user's authentication using JSON Web Tokens (JWT).
5. The signed JWT is passed to the login boundary layer and stored to use for getting resources.

c, Sequence diagram "Search Hotels"

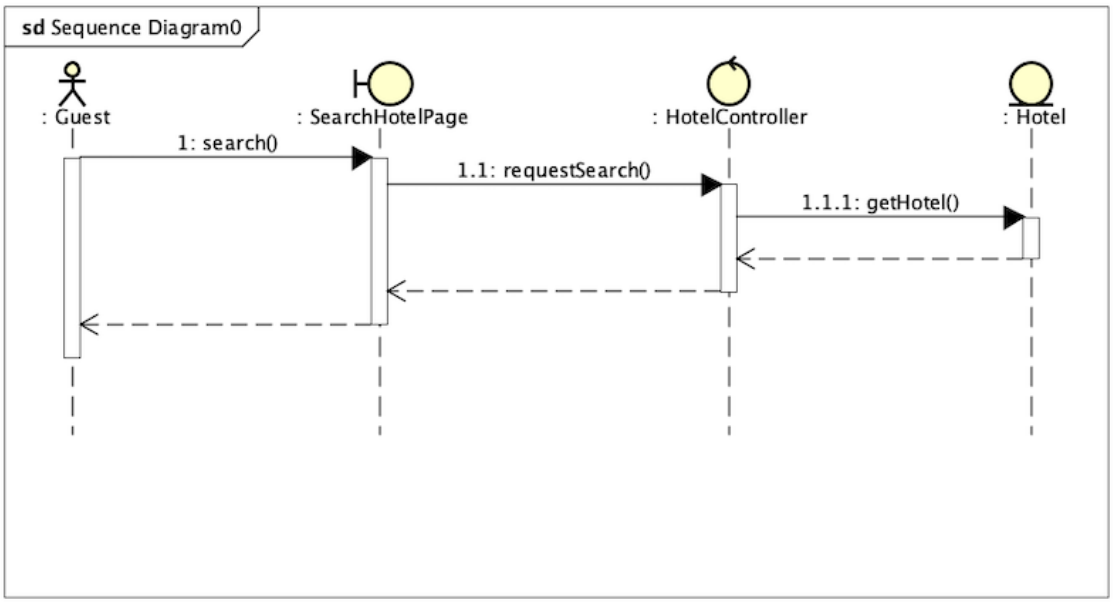

Figure 4.15: Search hotel sequence diagram

The objects participating in the interaction are the actor(Customer), the search hotel page interface (SearchHotelPage), the hotel control(HotelController), hotel entity(Hotel).

Event flow:

1. The Customer fills the search form and click search.
2. The request is sent to the hotel control to get the list of available hotels from the hotel entity.
3. Hotel entity returns result to the hotel control.
4. The hotel control return result to the search hotel page interface so that customer can see it.

d, Sequence diagram "Reserve hotels"

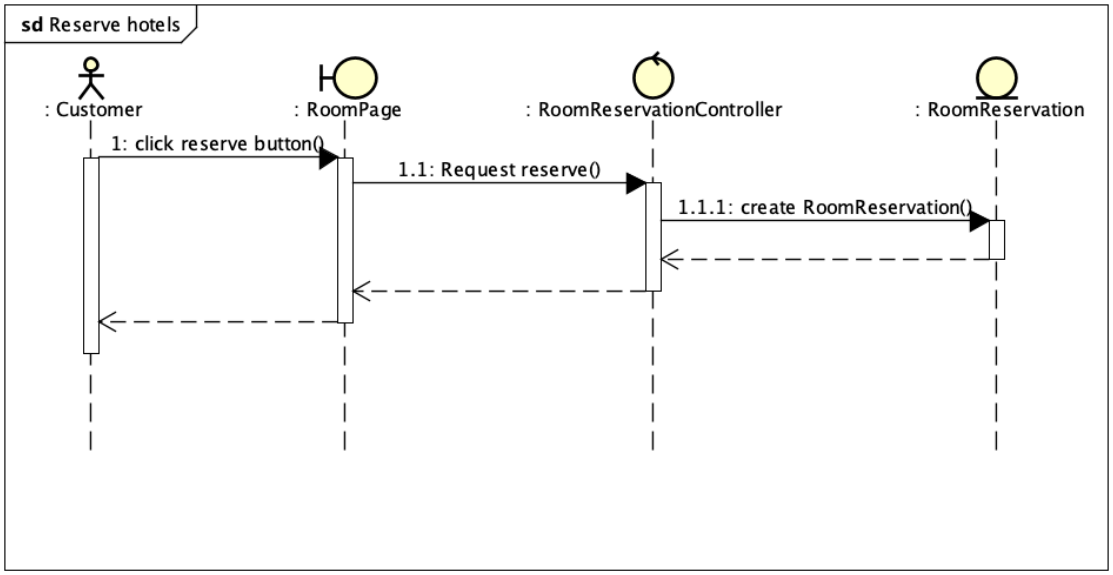

Figure 4.16: Reserve hotel sequence diagram

The objects participating in the interaction are the actor(Customer), the room detail interface (SearchHotelPage), the room reservation control(RoomReservationController), reservation entity(RoomReservation).

Event flow:

1. The Customer select check-in and check-out time from the room details interface and click reserve button.
2. The request is sent to the room reservation control.
3. The room reservation control send a request the room reservation entity to create a new record.
4. The reservation entity returns result to the room reservation control.
5. The hotel control return result to the search hotel page interface to notify customer successful request.

e, Sequence diagram "Cancel Reservation"

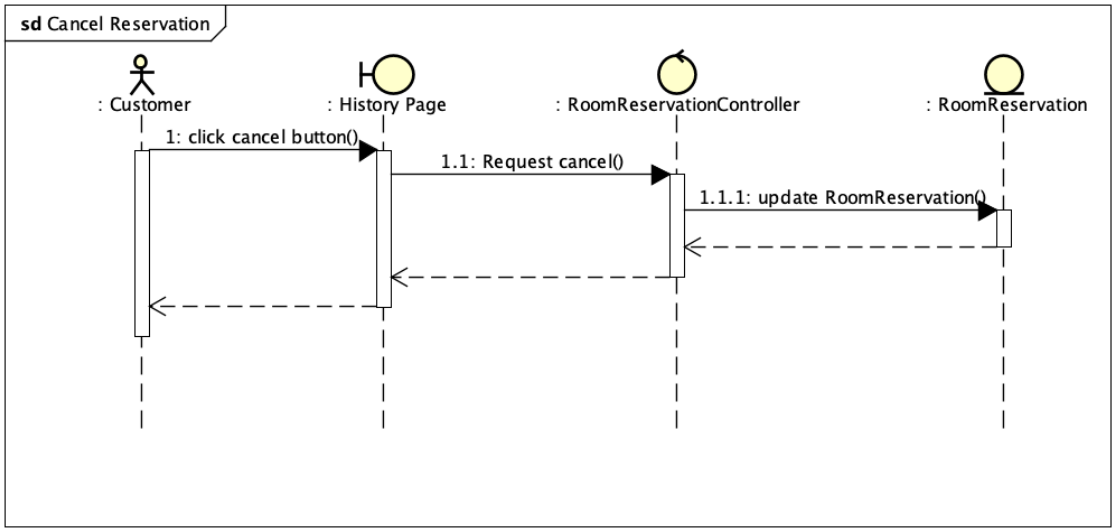

Figure 4.17: Cancel reservation sequence diagram

The objects participating in the interaction are the actor(Customer), the history interface (HistoryPage), the room reservation control(RoomReservationController), reservation entity(RoomReservation).

Event flow:

1. The Customer click cancel button on a reservation from the history interface.
2. The request is sent to the room control to update the reservation
3. The room reservation control send a request the room reservation entity to update the record.
4. The reservation entity returns result to the room reservation control.
5. The room reservation control return result to the search history page interface to update the status of the reservation.

f, Sequence diagram "Approve Reservation"

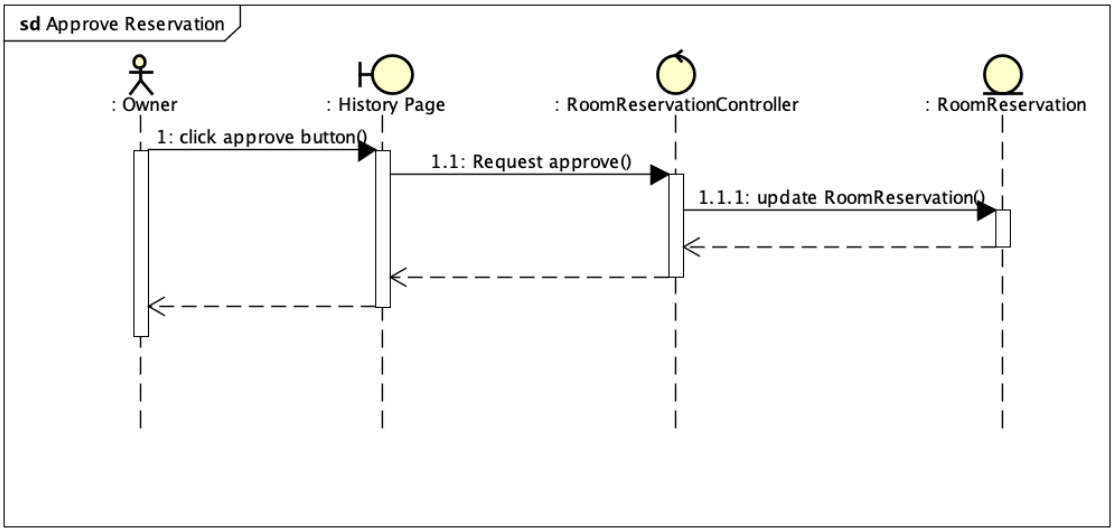

Figure 4.18: Approve reservation sequence diagram

The objects participating in the interaction are the actor(Owner), the history interface (HistoryPage), the room reservation control(RoomReservationController), reservation entity(RoomReservation).

Event flow:

1. The Owner click approve button on a reservation from the history interface.
2. The request is sent to the room control to update the reservation
3. The room reservation control send a request the room reservation entity to update the record.
4. The reservation entity returns result to the room reservation control.
5. The room reservation control return result to the search history page interface to update the status of the reservation.

g, Sequence diagram "Edit Profile"

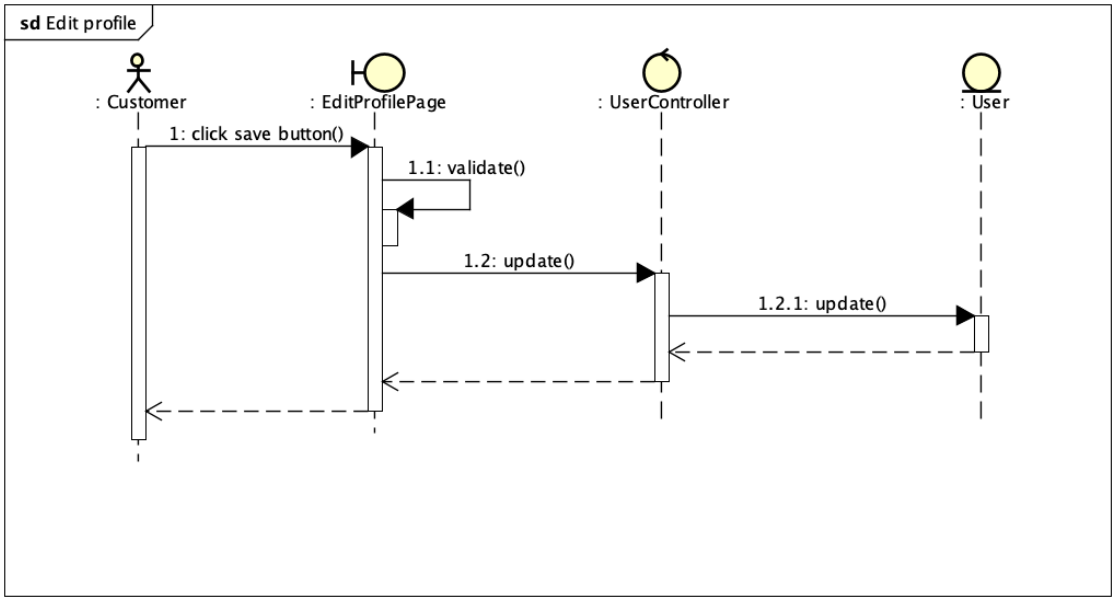

Figure 4.19: Edit profile sequence diagram

The objects participating in the interaction are the actor(Customer), the Edit profile interface (EditProfilePage), the User control(RoomReservationController), User entity(RoomReservation).

Event flow:

1. The actor fills in the profile form and click the save button on the Edit profile interface.
2. The Edit profile interface validate the form and notify invalid fields.
3. The Edit profile interface send a request User control to update the profile information.
4. The reservation entity returns result to the room reservation control.
5. The room reservation control return result to the Edit profile interface to update the information of the user.

4.2.3 Database design

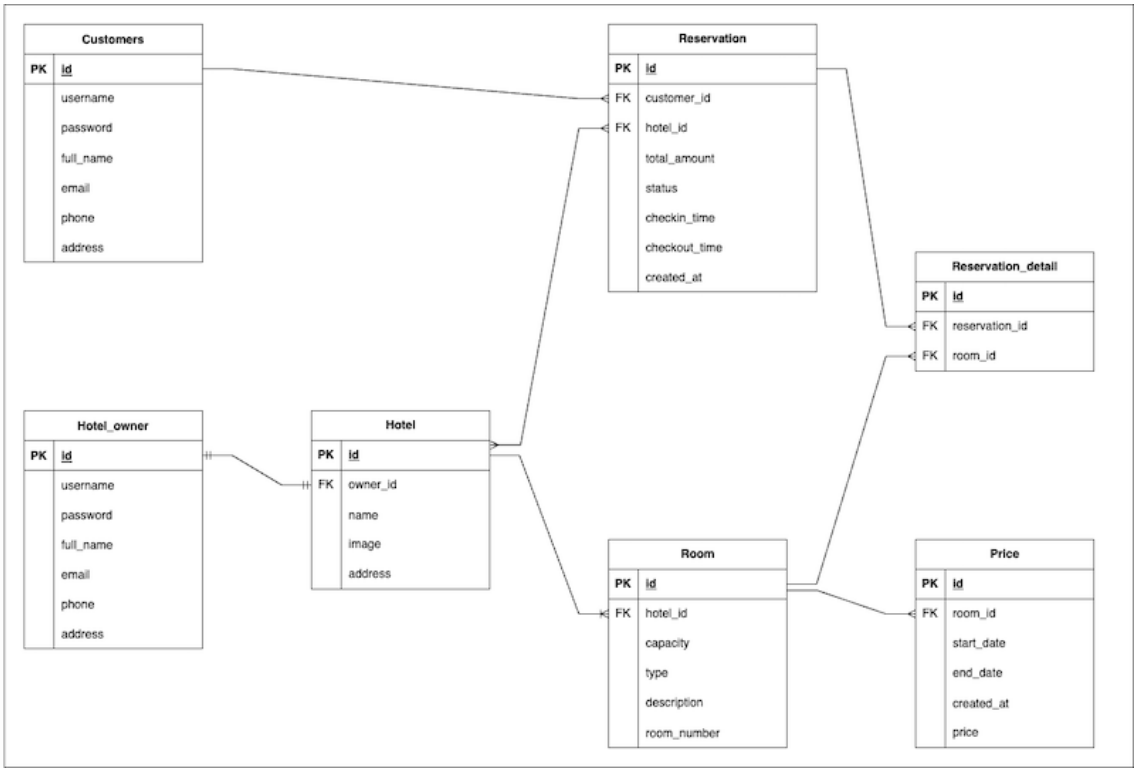

Figure 4.20: ERD Diagram

The entity relationship diagram of the system in Figure 4.20 shows the relationships of entities in the system's database. I build a database including entities: users, videos, comments and notifications.

Description of the entity relationship:

- One hotel have to have one owner and one owner have to have one hotel.
- One hotel can be reserved in many reservations.
- One customer can make many reservations.
- One reservation can contain many rooms and one room can be in many reservations, as showed in Reservation_detail.
- One room can have many price at different periods of time.
- One hotel can have many rooms.

a, Customer table design

Order	Fields	Type	Description
1	id	ObjectId	User identity
2	full_name	String	User full name
3	username	String	User name
4	password	String	User password
5	phone	String	User avatar url
6	email	String	User email
7	address	String	User address

Table 4.1: Users table

b, Hotel_owner table design

Order	Fields	Type	Description
1	id	ObjectId	Owner identity
2	full_name	String	Owner full name
3	username	String	Owner name
4	password	String	Owner password
5	phone	String	Owner avatar url
6	email	String	Owner email
7	address	String	Owner address

Table 4.2: Hotel_Owner table

c, Hotel table design

Order	Fields	Type	Description
1	id	ObjectId	Video identity
2	user_id	ObjectId	Id of the hotel owner
3	name	String	Hotel name
4	image	String	Hotel image url
5	address	String	Hotel address

Table 4.3: Hotel table

d, Room table design

Order	Fields	Type	Description
1	id	ObjectId	Room identity
2	hotel_id	ObjectId	Id of the hotel that the room belongs to
4	capacity	Number	The number of persons can stay in room
5	type	String	Room type
6	roomNumber	String	Room number
7	description	String	Room description

Table 4.4: Room table

e, Reservation table design

Order	Fields	Type	Description
1	id	ObjectId	Reservation identity
2	room_id	ObjectId	Id of the room of that reservation
3	user_id	ObjectId	Id of the user of that makes the reservation
4	total_amount	Number	Total cost
5	status	Number	Reservation status
7	checkin_time	Date	Check-in time
8	checkout_time	Date	Check-out time
8	created_at	Date	Created time

Table 4.5: Reservation table

f, Price table design

Order	Fields	Type	Description
1	id	ObjectId	Reservation identity
2	room_id	ObjectId	Id of the room of that reservation
4	price	Number	Price
5	start_date	Date	Start date price applied
7	end_date	Date	End date price applied
8	created_at	Date	Time price created

Table 4.6: Reservation table

4.3 Application Building

4.3.1 Libraries and Tools

The list of tools I used in the project is listed in Table 4.7 as follows:

Purpose	Tools	URL
IDE	Visual Studio Code	https://code.visualstudio.com/
IDE	IntelliJ IDEA	https://www.jetbrains.com/idea/
Database management system	MySQL	https://www.mysql.com/
API testing	Postman	https://www.postman.com/

Table 4.7: List of tools

4.3.2 Achievement

Statistical information about packages in the application as follow:

Package	Size
Client	1.1MB
Server	2.2MB

Table 4.8: Statistical information about packages

4.3.3 Illustration of main functions

The following are images illustrating the functions in the application and the description of the interface.

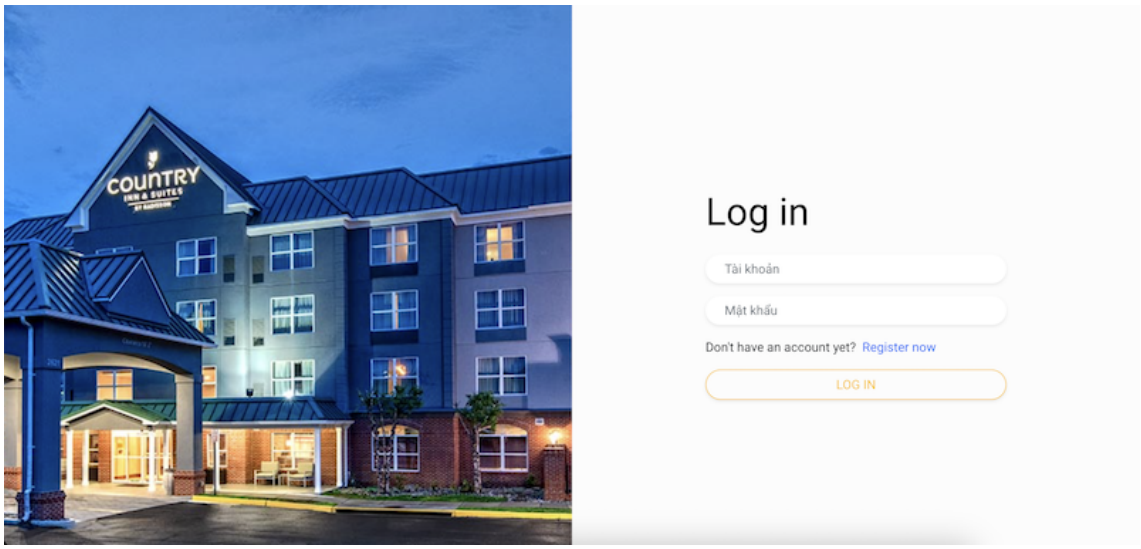

Figure 4.21: Login page

The login page allows users to enter their email and password. Upon successful login, it navigates to the home page.

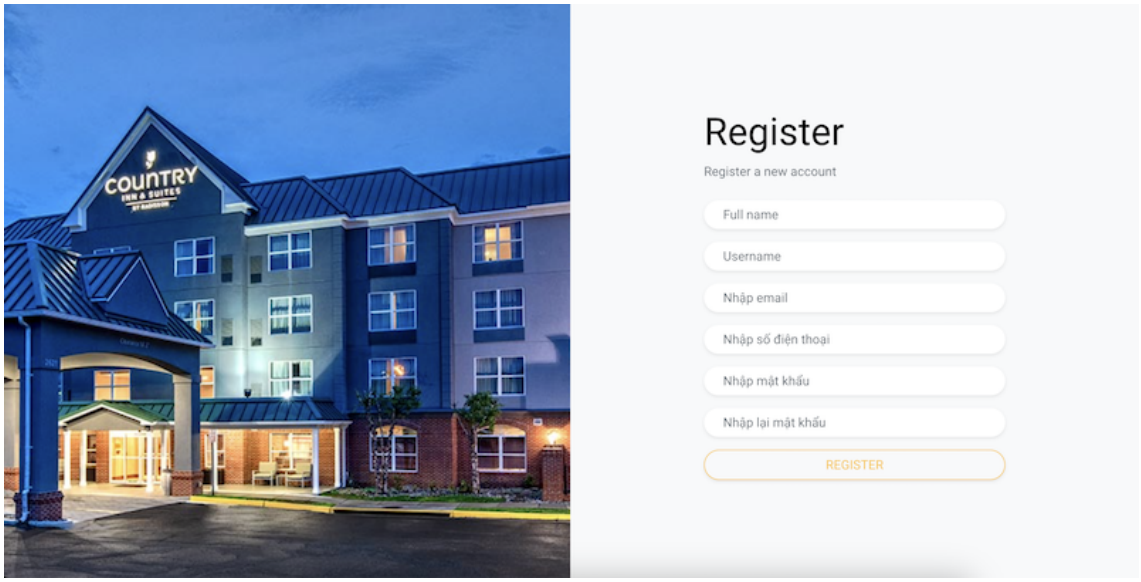

Figure 4.22: Register page

The register page allows users to enter their user name, email, password, and avatar. Upon successful registration, it navigates to the login page.

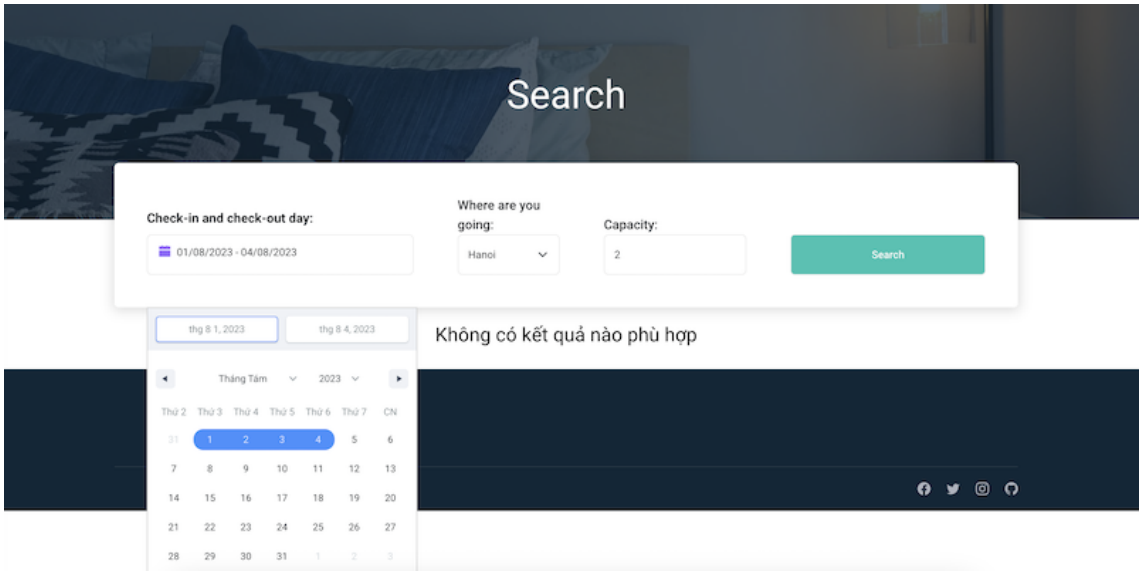

Figure 4.23: Search hotel page

The search hotel page allows users to search hotel by choosing check-in and check-out time, destination and number of persons. The system will return result as shown in Figure 4.24.

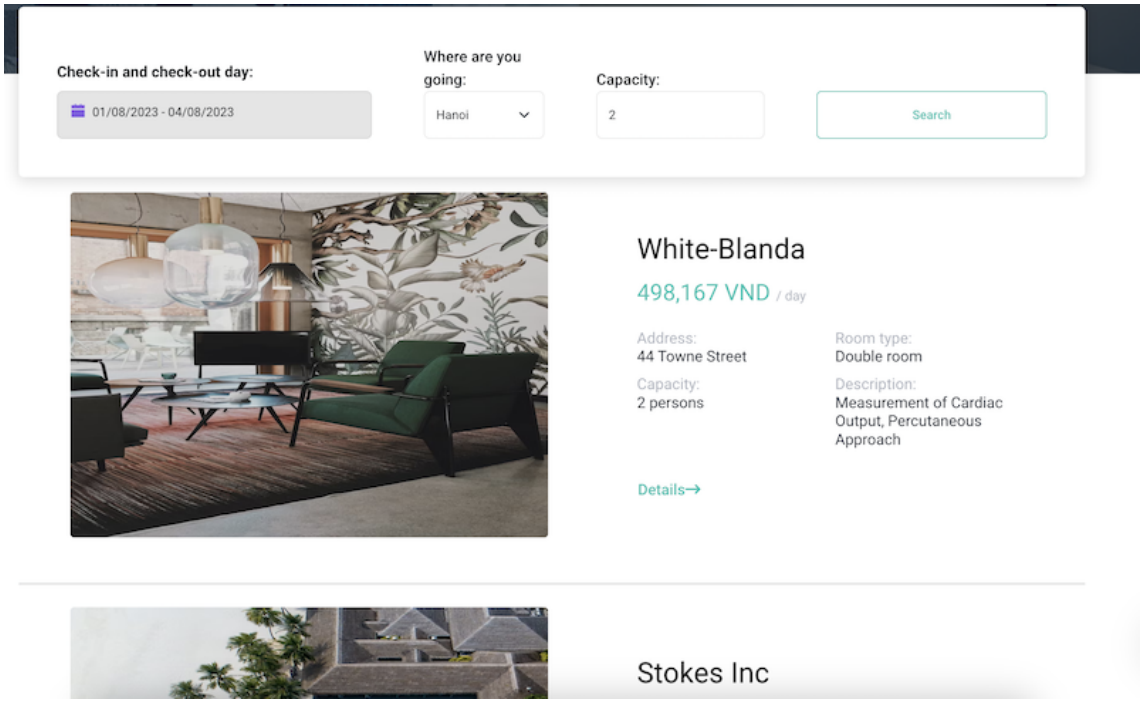

Figure 4.24: Search result page

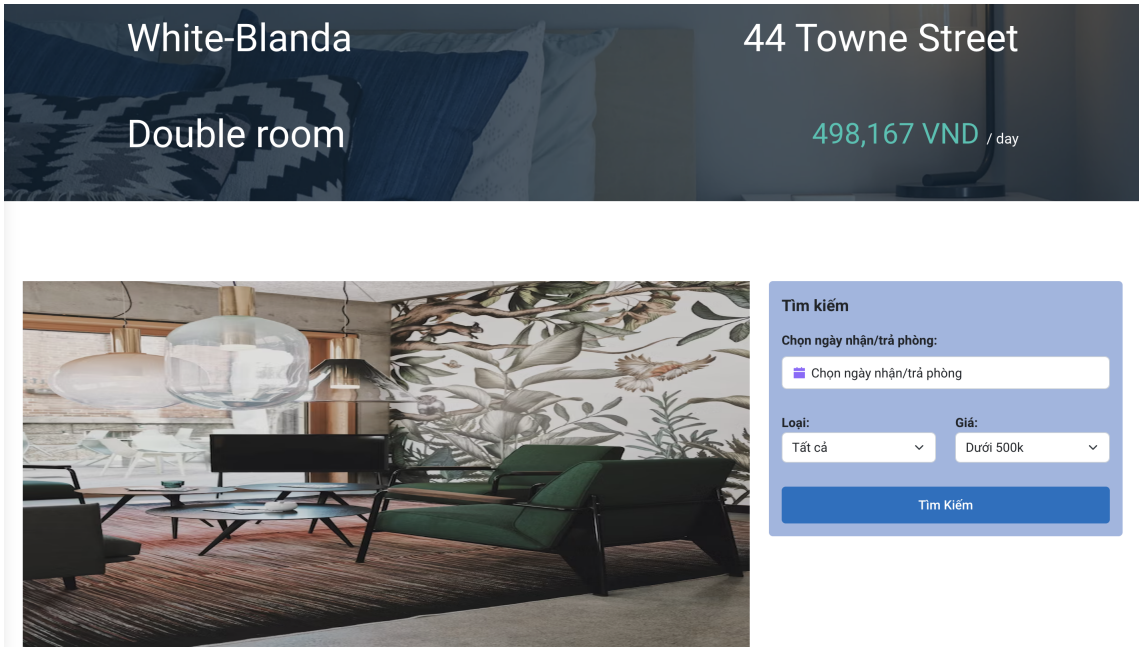

Figure 4.25: Room detail page

The room detail page shows information of the room included hotel name, address, room type, price, the list of suggested rooms in the hotel as shown in Figure 4.26. Users are also able to make a reservation from this page by clicking the booking button

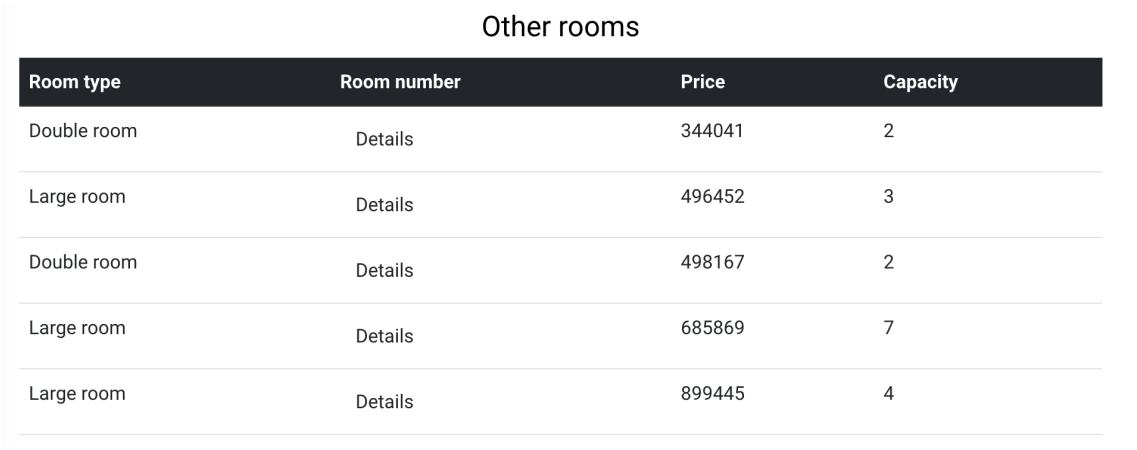

Figure 4.26: Suggested room

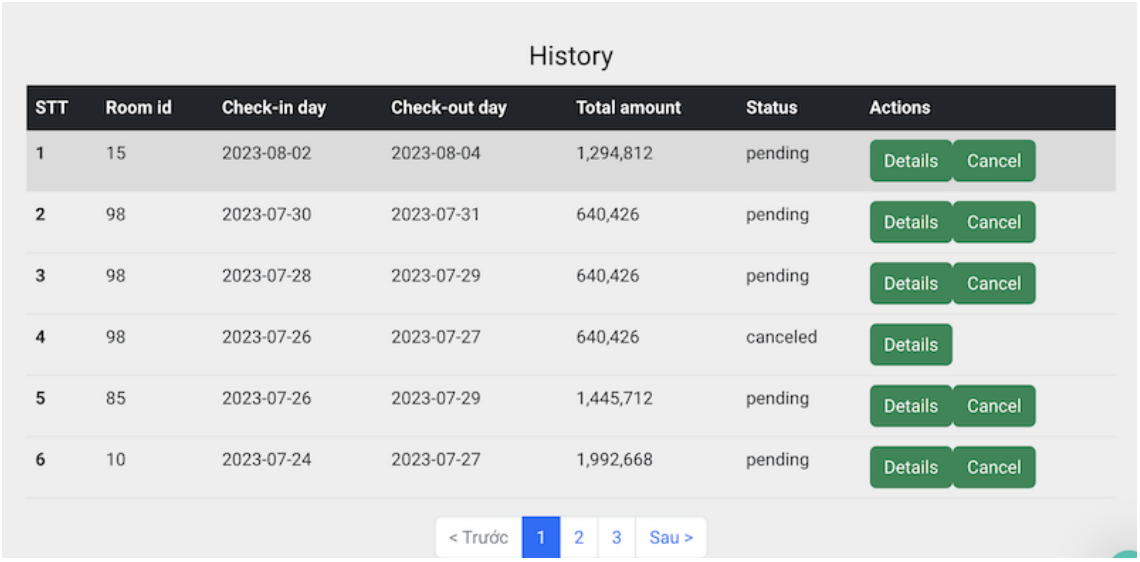

Figure 4.27: History page

The room detail page shows information of the reservation history of users. User can see the reservation detail by clicking Details button or cancel the reservation that have status pending. There will be a confirmation popup as Figure 4.17.

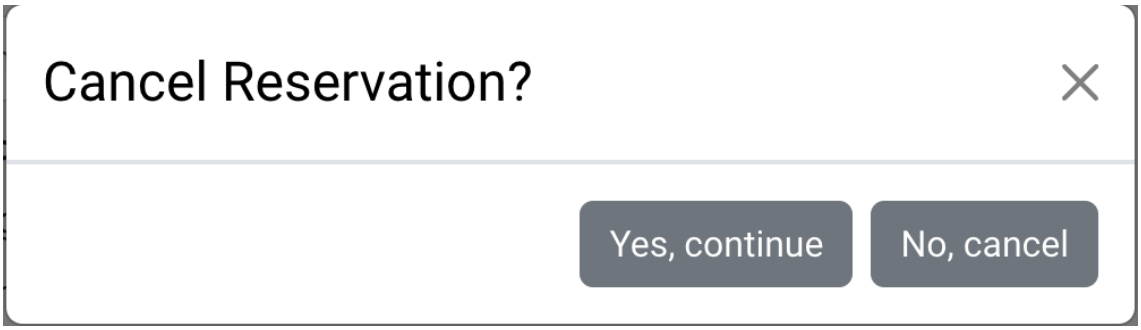

Figure 4.28: Confirmation popup

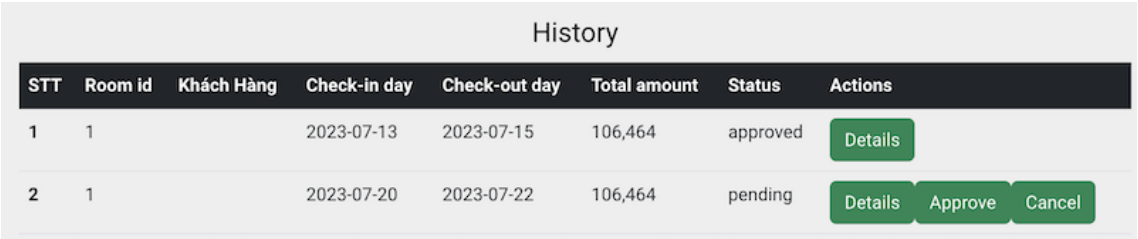

Figure 4.29: Owner history page

If the user role is hotel owner this page will show reservation history of the hotel that belongs to him or her. The owner can apprprove or cancel the pending orders.

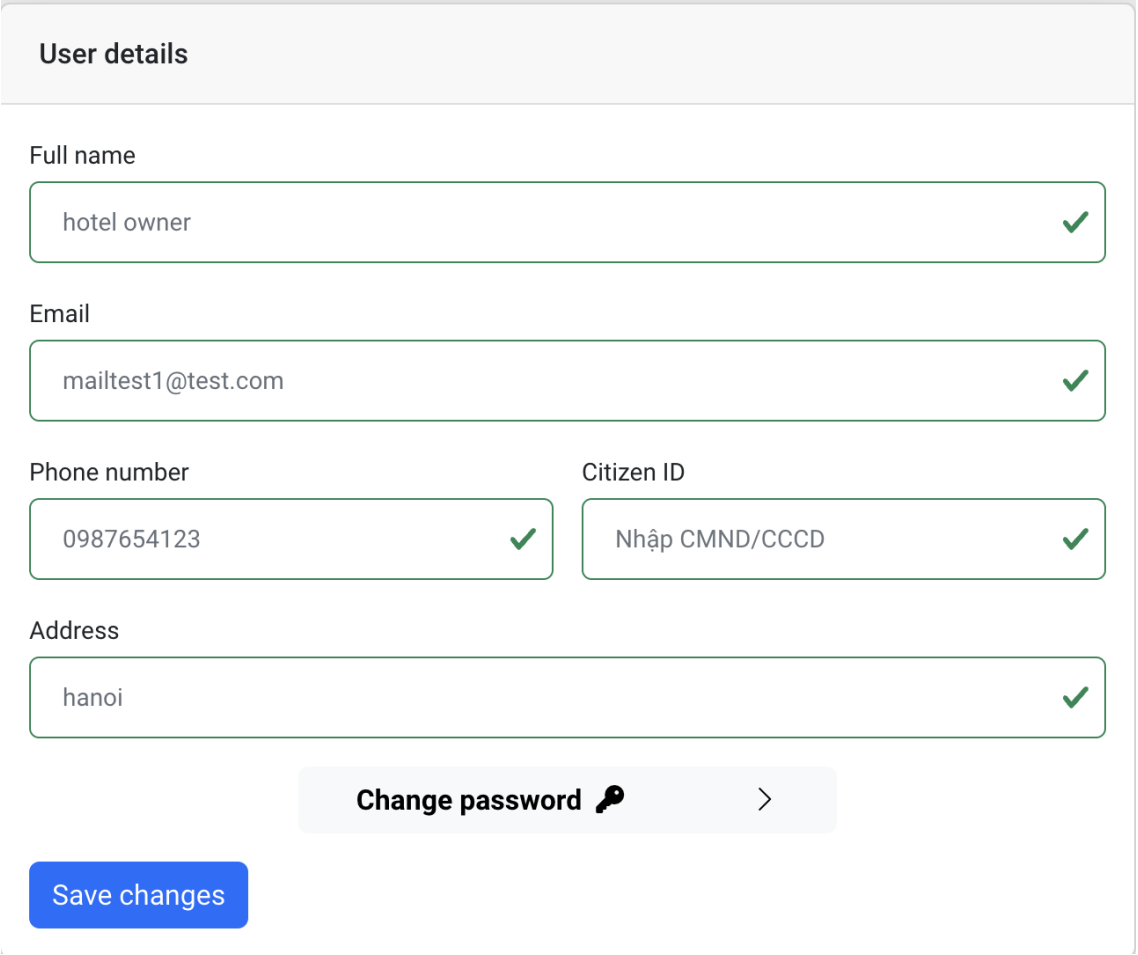

Figure 4.30: Edit information page

User can edit their information included their full name, email, phone number, citizen ID, address as well as account password in the edit information page.

4.4 Testing

The method I apply testing in my graduation project is manual testing. Following is the list of system test cases:

4.4.1 Authentication test

- Register

  Test case 1: Enter one of the fields missing or invalid (name, email, password).

  Expected result: Warning that invalid fields.

  Result: Pass.

  Test case 2: Enter existed email user.

  Expected result: Warning existed email fields.

  Result: Pass.

- Login

  Test case 1: Enter one of the fields missing or invalid (email, password).

  Expected result: Warning that invalid fields.

  Result: Pass.

  Test case 2: Enter wrong email or password user.

  Expected result: Warning wrong email or password.

  Result: Pass.

4.4.2 User test

- Search hotel.

  Test case 1: Input suitable information.

  Expected result: Display hotel list.

  Result: Pass.

  Test case 2: Input information that meets no result.

  Expected result: Notify "No result.

  Result: Pass.

- Edit information

  Test case: Enter one of the fields missing or invalid (name, email, password).

  Expected result: Warning that invalid fields.

  Result: Pass.

- Resev

  Test case 1: Enter one of the suitable check-in and check-out time.

  Expected result: Reserve successfully.

  Result: Pass.

  Test case 2: Enter one of the unsuitable check-in and check-out time().

  Expected result: Warning that the room has been reserved by other one.

  Result: Pass.

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

During the course of the thesis, the development of our web application using the micro-services architectural pattern has proven to be a successful approach in creating a robust and maintainable software system. Microservices architecture provides an effective solution to address the challenges encountered in systems with a large number of users. In traditional monolithic systems, an increase in users can lead to performance bottlenecks and scalability issues. However, with microservices, the application is divided into smaller, autonomous services, each responsible for a specific functionality. This modular approach allows for horizontal scaling, where individual services can be replicated and deployed independently to handle increasing user loads. As a result, microservices ensure better resource utilization and improved system performance, even during high user traffic. Additionally, microservices allow for easier maintenance and updates, as changes in one service do not require redeploying the entire application. By offering enhanced scalability and agility, microservices enable organizations to cater to a growing user base while maintaining a responsive and efficient system.

Howerver, the cooperation between different services of this architecture may leads to the negative user experience.One of the scenarios that the problem occurs is when a service A(Provider) send requests to a service B(Consumer) but the Consumer cannot be able to response all of the requests from the former one. At this time, users using other works from the service A have to spend unnecessary time on waiting the above process. One of the effective technologies to deal with this problem is a message queue and RabbitMQ is our selection.

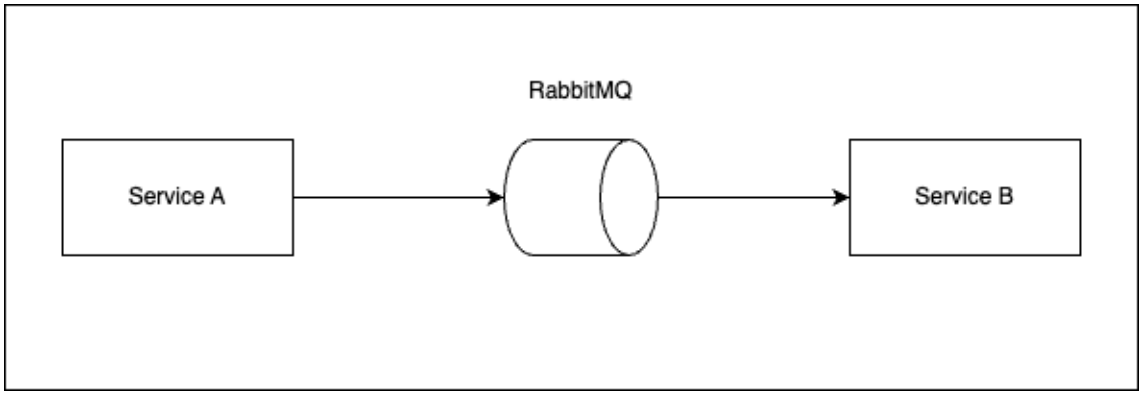

Figure 5.1: Message queue

The figure 5.1 illustrates the mechanism of the message broker RabbitMQ. The

message queue mechanism is a core component of message-oriented middleware, which enables communication between different software components or services in a distributed system. It follows the publisher-subscriber pattern, where the sender of a message (Provider) does not directly interact with the recipient (Consumer) but instead sends messages to a central intermediary called the message queue.

The message queue acts as a buffer, storing messages until they are consumed by the intended subscribers. This decoupling of communication allows for asynchronous and reliable message exchange between different components, even if they operate at different speeds or experience intermittent availability.

By applying this technology, the user experience is enhanced significantly since the RabbitMQ helps users to wait for responses from the Consumer.

We have successfully built an instance of RabbitMQ to solve the problem and manage the message queue as in the figure 5.2

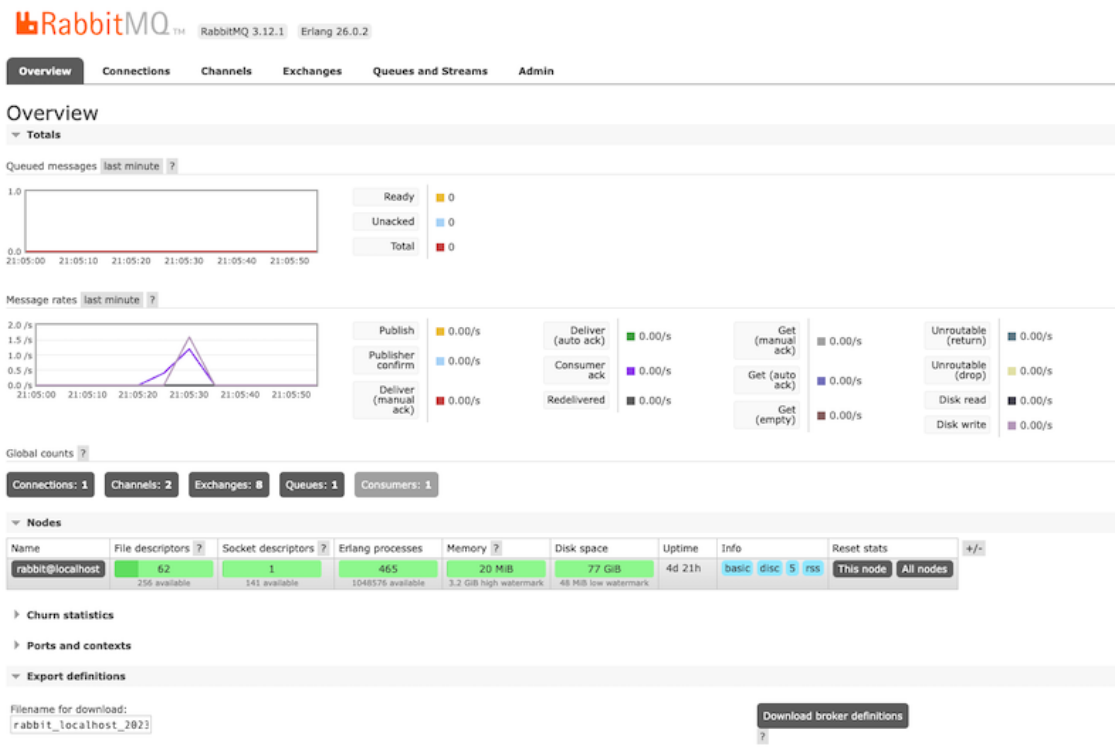

Figure 5.2: RabbitMQ management

Engaging in this project has been a highly enriching experience that has equipped me with invaluable knowledge and skills to propel my future career forward. Throughout the project, I delved into various aspects of web development, honing my technical abilities in programming, database management, and frontend design. Besides that, I developed problem-solving acumen, learning to navigate complexities and find innovative solutions.

5.2 Future work

While the current implementation of our web application has met its initial objectives, there are several areas of future work that can further enhance its capabilities and user experience:

1. User Interface Enhancements: Continuously improving the user interface with modern design trends, responsiveness, and accessibility will keep the application visually appealing and user-friendly.
2. Internationalization and Localization: Adding support for multiple languages and regions will make the application accessible to a global audience.
3. Automated Testing: Implementing automated testing suites will ensure the application's reliability and minimize the risk of introducing regressions during updates.
4. Mobile App Development: Exploring the development of mobile applications using technologies like React Native will extend the application's reach and accessibility.

SHORT NOTICES ON REFERENCE

[1] Spring Boot tutorials. *[Online]. Available:* `https://spring.io/guides/gs/spring-boot/`.

[2] ReactJS tutorials. *[Online]. Available:* `https://legacy.reactjs.org/tutorial/tutorial.html`.

[3] Microservices in Spring Boot *[Online]. Available:* `https://spring.io/microservices`.

[4] RabbitMQ tutorial *[Online]. Available:* `https://www.rabbitmq.com/#getstarted`.