HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Design and develop website for travel social network

TRẦN MẠNH HÙNG
hung.tm176775@sis.hust.edu.vn

Major: Information Technology
Specialization: Global ICT Program

Supervisor: Ph.D Trần Hải Anh _____

Signature

Department:

School: Information and Communications Technology

HANOI, 08/2022

ACKNOWLEDGMENTS

Firstly, I would like to extend my sincere thanks to all the teachers at the School of Information and Communication Technology - Hanoi University of Science and Technology, who taught and educated me. Thank you to the teachers who provided us all with support, guidance, and precious life lessons which will help me a lot in my future career path.

In particular, I would like to express my deepest appreciation to my instructor Tran Hai Anh. He gave me a lot of good advice on choosing the topic and building the system's functions. I appreciate his guidance that made it possible for me to complete this graduation thesis.

Lastly, I would like to express my gratitude to everyone who helped and supported me through my 5-year journey. Thank you parents for always caring, giving useful advice, and creating the best environment for my growth, especially for my academic career.

ABSTRACT

After the COVID epidemic was overcome worldwide, people's demand for travel increased much. This leads to the need for more information channels for people to share travel experiences or search for information related to the places they want to visit. Currently, there are many channels of tourism information, but they have not been specialized into a website that only shares and tracks tourism information. Nowadays most people can access the internet very easily. Therefore, I have researched and developed a travel social networking website that allows people to easily search for details about the hotel or tourist destination they need or follow other users with the same passion for traveling.

This social networking website will be designed according to the client-server model. In which the server side will use java spring boot and design according to microservice architecture to easily maintain and scale the application when the number of users increases. Docker will be used as the environment for development and deployment. More specifically, docker will be used to build an environment including a database, Kafka, Redis, and GUI tool to interact with DB, ... The client-side will use the ReactJS library developed by Facebook. The client-side will also be designed so that we can easily maintain or add a new feature.

Finally, we will get a system of social networking websites that are highly extensible, easy to change, and high performance. The website serves users in finding the right place for vacations, outings with friends, or storing memories and spreading joy and positive energy to everyone.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBRIVIATIONS

Abriviation	Full Expression
API	Application Programming Interface
CRUD	Create, read, update, delete
DTO	Data Transfer Object
ERD	Entity Relationship Diagram
GUI	Graphical User Interface
ON	Order number
UI	User Interface

CHAPTER 1. INTRODUCTION

1.1 Motivation

Along with the process of globalization and the development of information technology, the internet in the world and Vietnam has almost been covered. The participation of individuals on the internet is increasingly active and the need to share information and connect with friends is an essential need to promote the birth and development of social networks.

Currently, social networks are developing extremely brilliantly, and the number of people accessing and registering members in social networks is increasing. Typically, some social networking websites are Facebook or Twitter. Even so, user demand is still very high and specificity is even more necessary.

After the COVID pandemic is over, domestic tourism activities have witnessed a boom, represented by the strong growth of visitors over the months. According to the Vietnam National Administration of Tourism, in May and June 2022, the number of domestic tourists reached 12 million and 12.2 million respectively. This is the highest number of domestic visitors in a month in Vietnam in recent years. As a result, the number of domestic tourists in the first 6 months of 2022 reached 60.8 million (nearly 1.4 times higher than the same period in 2019), of which about 8.3 million guests stayed. Total revenue from tourists is estimated at 265,000 billion VND.

According to Digital's statistics, as of June 2021, the number of Internet users in Vietnam is nearly 70 million, an increase of 0.8% in the period 2020-2021 (accounting for more than 70% of the population); The number of social network users in Vietnam is nearly 76 million, an increase of nearly 10 million people within a year (equivalent to 73.7% of the population). Every day, Vietnamese users spend up to 7 hours participating in Internet-related activities.

If Reddit is a website specializing in updating social news, LinkedIn is a place to connect employers and candidates, then I aim to build a social networking site specializing in sharing photos or videos about tourist attractions, and user experiences. That's why I chose the topic "Design and develop a website for travel social network" for my graduation thesis, and I named this social network Vifrin.

1.2 Objectives and scope of the graduation thesis

The goal of the project is to build a social network to create a centralized environment with an intuitive interface that is easy to use, as well as can simplify

operations for users. In addition, it is also a tool for users to easily track favorite places, and share their experiences after interesting trips. The system also allows users to search for places of interest to get useful information about that place, as well as featured hotels in that location. In addition, the system can be easily further developed on mobile platforms (Android and IOS).

To meet the goals and requirements of the above problem, I would like to propose developing applications according to the client-server model. Specifically: (i) On the backend side, I suggest using Java Spring Boot to build RESTful APIs, (ii) On the frontend side, I suggest using ReactJs framework to build the user interface. In detail about the advantages and reasons for using these technologies in the project, I would like to present them more clearly in Chapter **??**

1.3 Tentative solution

Social networking site aims to create a friendly, reliable environment, making it easy for users to find useful information about the place they want to go. Users can rate the place through comments, all comments are real-time, helping the rating of the place to be updated as quickly as possible. What's more, tourist destinations also suggest featured hotels, making it easier for users to choose accommodation when starting a trip. Users can follow other users when they share an experience, enter reviews, to express their feelings about that experience. The system is suitable for many different audiences of many ages, as long as that person needs to travel.

To meet the goals of the solution, the system needs to have good performance and easily integrate many different technologies. Vifrin is a web product, built on the client-server model, which divides the application into two components. The server is a place to provide APIs that clients can use to interact with data. The advantages of this model are reflected in centralization, security, scalability, and accessibility. The server side will use the Java Spring Boot Framework designed according to the microservices architecture, with many outstanding features compared to mono service, increasing the ability to test, maintain, and scale the application. The client-side will use the ReactJS library to create the user interface, use redux, and redux-saga to manage the global state, increase the scale when adding new features to the system, easier to test execution. After the application development process, I completed the web version with all the features. The application has been deployed on Microsoft Azure servers. When developing and deploying, I use docker to help install the environment and run the project faster and more professionally.

1.4 Thesis organization

For the rest of this graduation thesis, it will be presented in the following sequence:

Chapter 2 Survey: In this chapter, I would like to show some reviews about other methods and some main functions of this software.

Chapter 3 Methodology: This chapter covers system design, structural modeling, behavior modeling, overview architecture design, class design, database design, and user interface design.

Chapter 4 System analysis and design: After researching user demand as well as doing some surveys, I started to design and implement the system.

Chapter 5 System deployment and evaluation: This chapter talks about test results and actual implementation status

Chapter 6 Solution and contribution: The main content of this chapter will be about proceeding to build the system and describing the results achieved.

Chapter 7 Conclusion: The final chapter presents the points that have worked and failed in the project, summarizes the development results, and analyzes new directions that allow improvement and upgrading of the system.

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS

2.1 Status survey

The tourism industry has a very large role, and the demand for users is high. Although the number of social networks in Vietnam is very high, there is still no specialized social networking site for tourism. We can know Facebook as the highest shared social network in the world, we can create groups for people to exchange, but this social network doesn't suggest prominent places. In addition, the amount of information on Facebook is very large and diverse. Therefore, it is difficult for users to find information about the tourist destination they want to find. There are also sites like TripAdvisor or Traveloka, which are quite famous travel review sites, but mainly focus on hotel room rentals and travel amenities, not on sharing the feeling of users. Therefore, Vifrin was born to create a social networking site focused on sharing experiences through trips, recommending outstanding places, or highly rated hotels in that place to help users make the right decisions, and get great experiences when making a trip.

2.2 Functional Overview

From the analysis of the above applications and platforms, I can draw the agents and functions according to the agent required in my system:

Table 2.1: Actors

Actor	Role
Guest	People who have not created an account, or have not logged in
User	A person who has successfully registered and logged in to the system and can use user functions.
Admin	As a user who logs in to the system with an administrator role, he has full control over the system

Overview of functions: Functions serve 3 agents.

- Guest: must register an account to use the system functions.
- User: when registering an account, the default permission is a user with functions such as login, logout, manage posts, rate, comment, update profile, and search.
- Admin: view, delete user accounts, manage destinations, hotels, view statistics about the total number of users, posts of the system.

2.2.1 General use case diagram

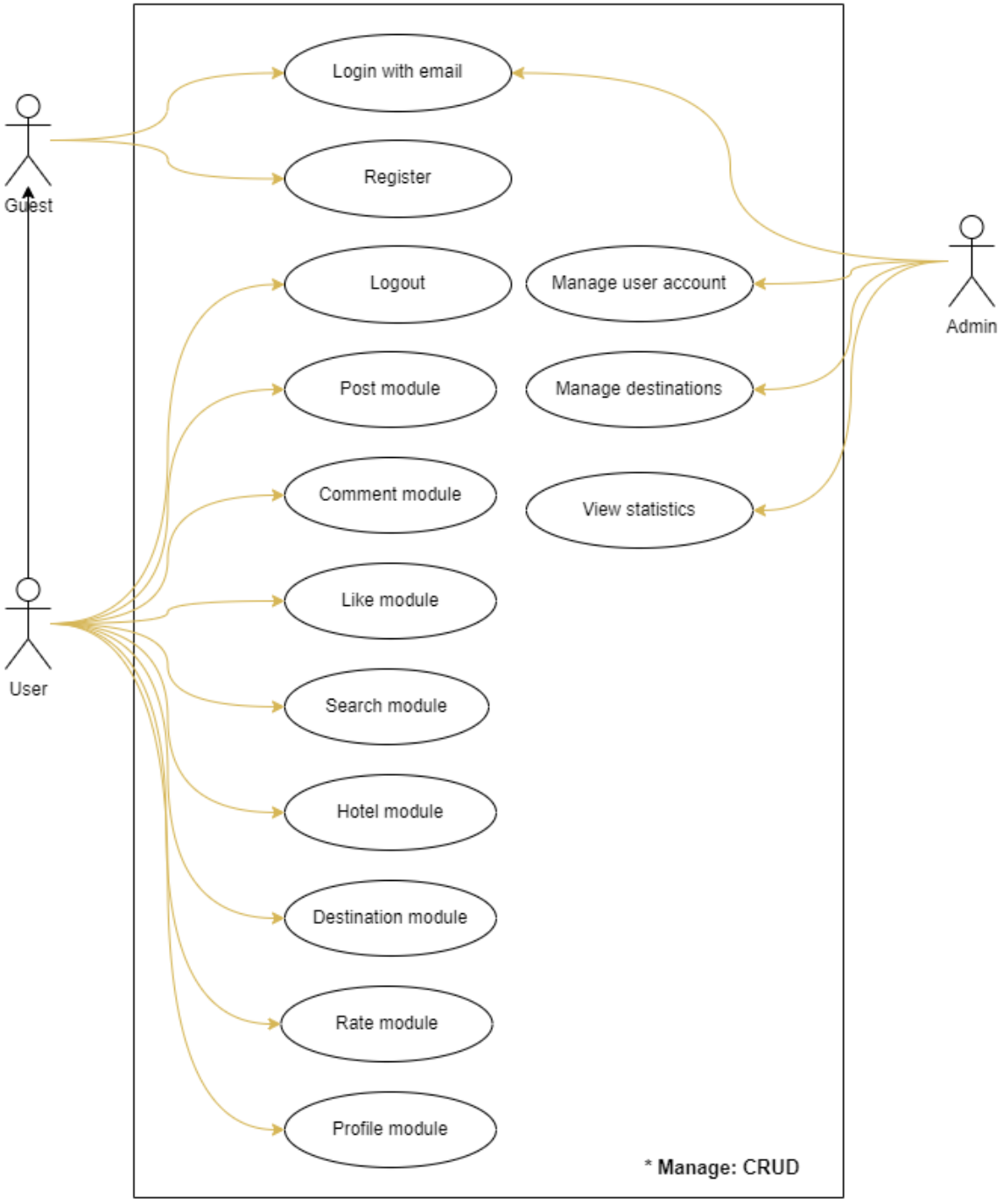

Figure 2.1: General usecase

2.2.2 Detailed use case diagram

a, Detailed use case diagram for "Post module"

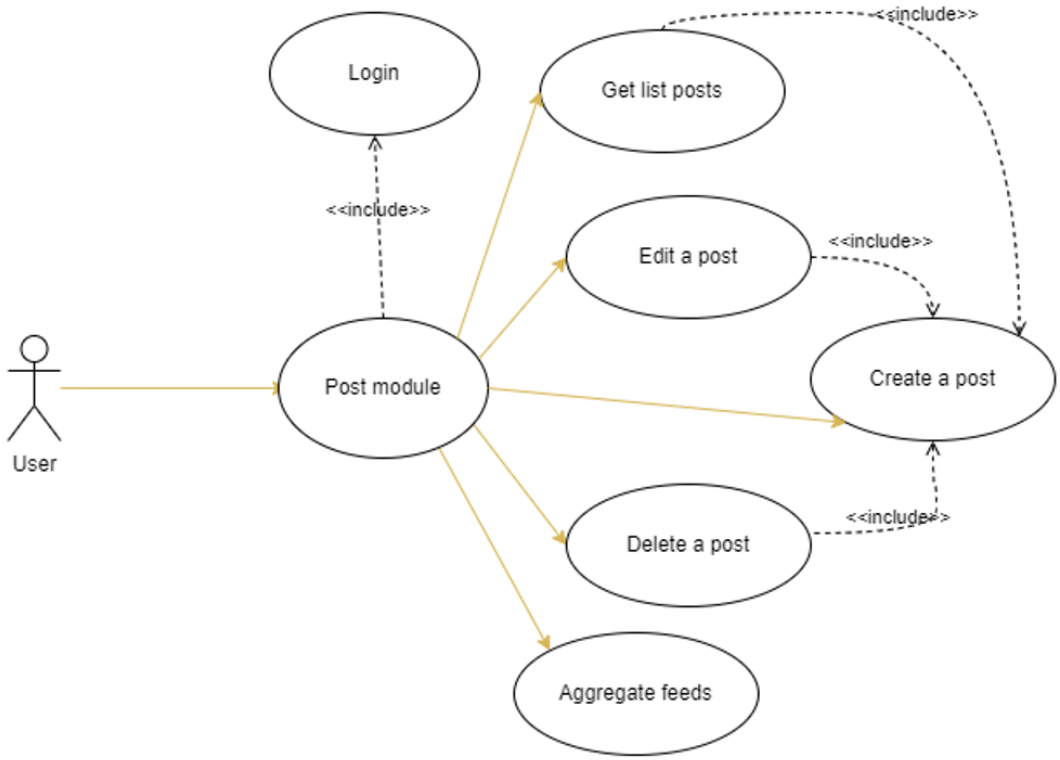

Figure 2.2: Post module

- Create a post: Users can share feelings, photos, videos about a place through posts
- Get lists post: Users get list personal posts or other user's post
- Edit a post: Users can edit personal posts
- Delete a post: Users can delete personal posts
- Aggregate feeds: Users can see the posts they are interested in by following other users or system can suggest posts.

b, Detailed use case diagram for "Comment module"

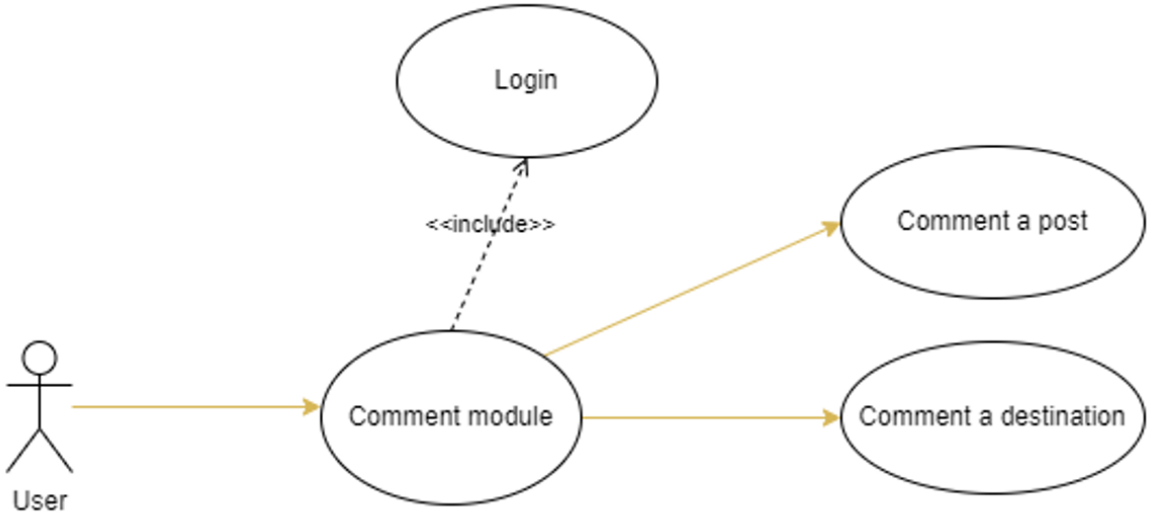

Figure 2.3: Comment module

- Comment a post: Users can comment on post
- Comment a destination: Users can comment on destination

c, Detailed use case diagram for "Like module"

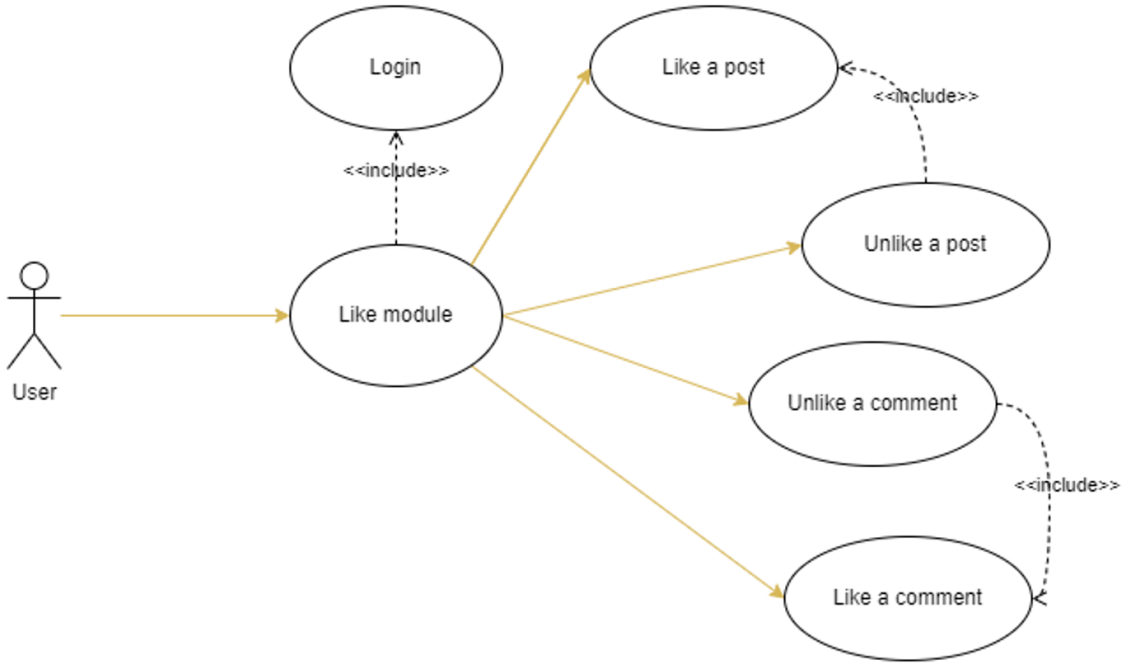

Figure 2.4: Like module

- Like a post: Users can like post
- Unlike a post: Users can unlike post

- Like a comment: Users can like comment
- Unlike a post: Users can unlike comment

d, Detailed use case diagram for "Rate module"

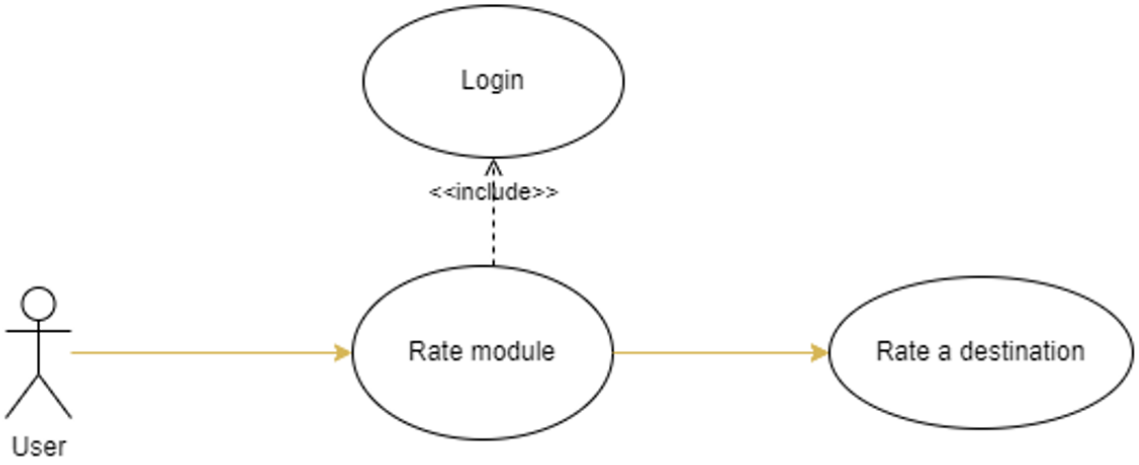

Figure 2.5: Rate module

- Rate a destination: Users can rate destination from 1 star to 5 star

e, Detailed use case diagram for "Search module"

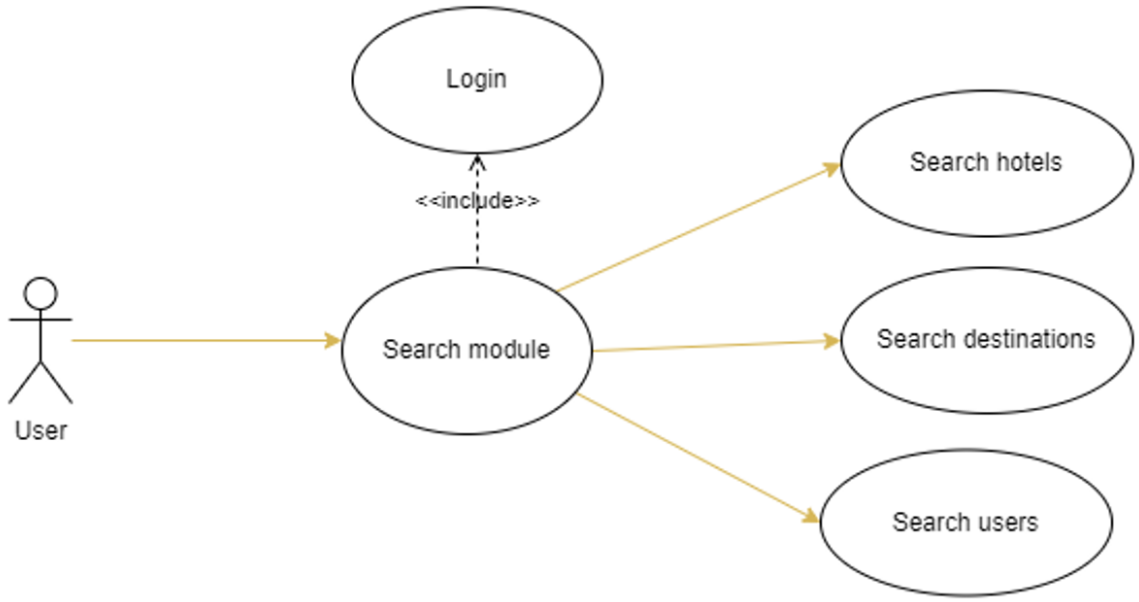

Figure 2.6: Search module

- Search hotels: Users can search hotels with keyword on the system
- Search destinations: Users can search destinations
- Search users: Users can search other users

f, Detailed use case diagram for "Destination module"

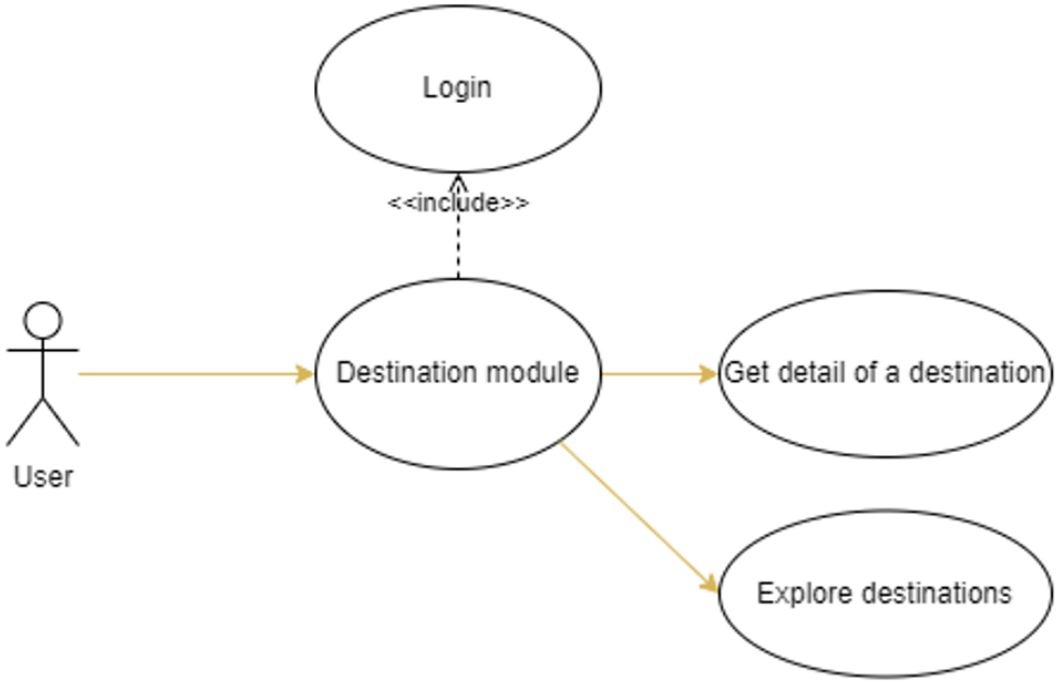

Figure 2.7: Destination module

- Get detail of a destination: Users can get information of destination: name, description, media, rating, comment.

g, Detailed use case diagram for "Hotel module"

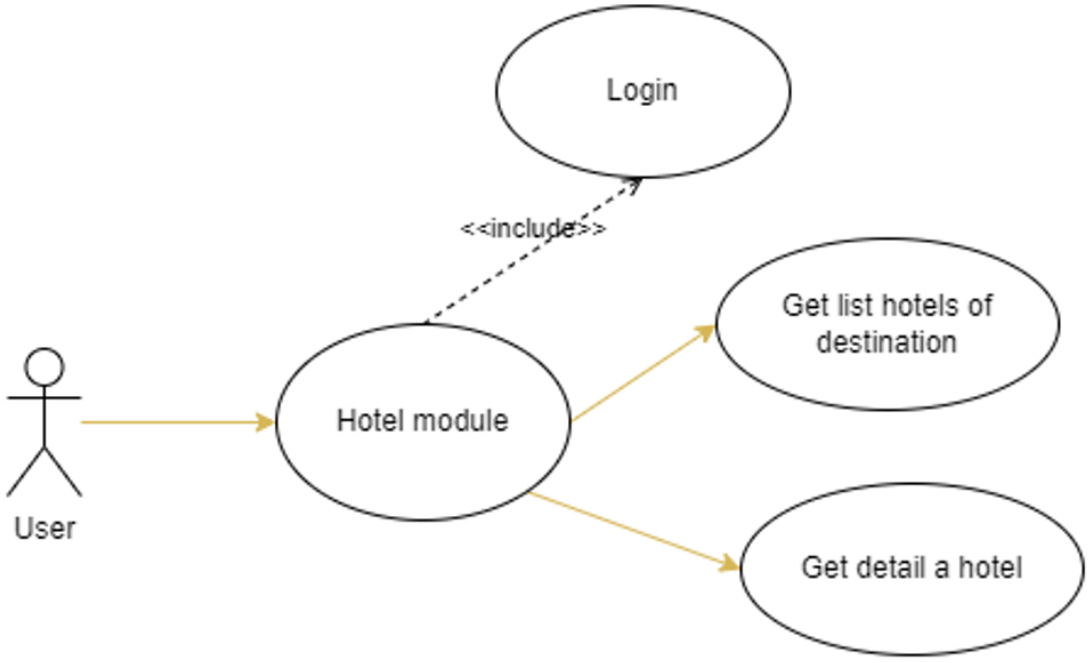

Figure 2.8: Hotel module

- Get list hotels of destination: Users can get list suggested hotels when see detail destination.
- Get detail a hotel: Users can get information of hotel: name, description, media, price, phone, address, utilities.

h, Detailed use case diagram for "Profile module"

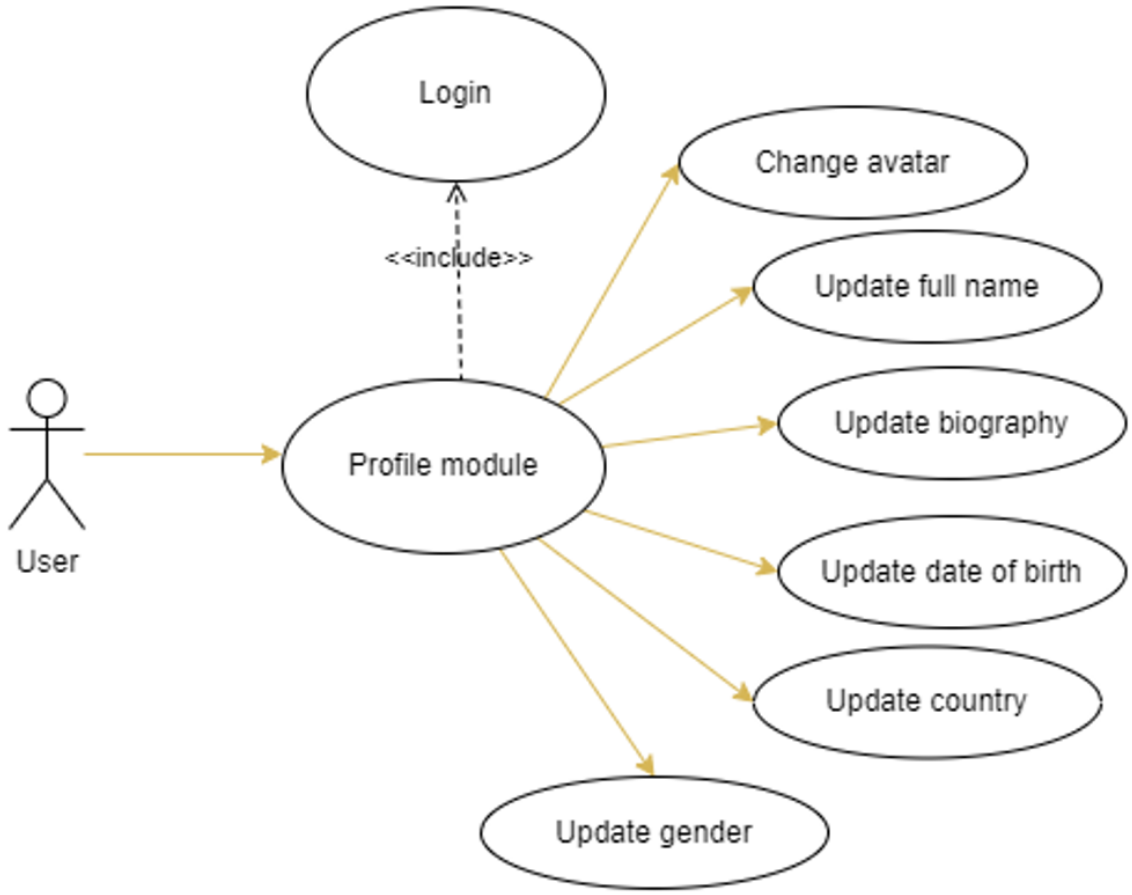

Figure 2.9: Profile module

- Change avatar: Users can change their avatar.
- Update full name: Users can update their full name.
- Update biography: Users can update their biography.
- Update date of birth: Users can update their date of birth.
- Update country: Users can update their country.
- Update gender: Users can update their gender.

2.3 Functional description

2.3.1 Description of use case "Register"

Table 2.2: Description of use case "Register"

Use case	Register		
Code	UC01		
Actor	Guest		
Description	A guest registers an account to be able to use system functions		
Pre-condition	None		
Post-condition	A guest has successfully registered for an account		
Main scenario (success)	**ON**	**Actors**	**Actions**
	1	Guest	Click to select the registration button
	2	System	Display account information registration interface
	3	Guest	Enter registration information, press the register button
	4	System	Check that the information entered is valid
	5	System	Announces successful registration and redirects to the login page
Extensions	None		
Alternative scenario	**ON**	**Actors**	**Actions**
	1	System	If input information is not valid, the system report an error asking the user to re-enter it

2.3.2 Description of use case "Login with email"

Table 2.3: Description of use case "Login with email"

Use case	Login with email		
Code	UC02		
Actor	Guest		
Description	the system allows guest to log in		
Pre-condition	A guest didn't log in and create an account on the system.		
Post-condition	An user has successfully logged in and could use the functions that provide by the system.		
Main scenario (success)	**ON**	**Actors**	**Actions**
	1	Guest	Chooses login function
	2	System	Display login form interface
	3	Guest	Enter login information, press the login button
	4	System	Check validity of information
	5	System	Announces successful registration and redirects to the home page
Extensions	None		
Alternative scenario	**ON**	**Actors**	**Actions**
	1	System	If input information is not valid, the system report an error asking the user to re-enter it

2.3.3 Description of use case "Create a post"

Table 2.4: Description of use case "Create a post"

Use case	Create a post		
Code	UC03		
Actor	User		
Description	An user post with photos, videos, and can add locations		
Pre-condition	An user has logged into the system.		
Post-condition	An user created post successfully.		
Main scenario (success)	**ON**	**Actors**	**Actions**
	1	User	Click the "Create post" button, and the
	2	System	Display create post form interface
	3	User	Enters the content of the post, uploads files, add destination and then click "Post" button.
	4	System	Announces post successfully, close popup, update list of user posts.
Extensions	None		
Alternative scenario	**ON**	**Actors**	**Actions**
	1	System	If the request fails, the system notifies the user.

2.3.4 Description of use case "Comment a post"

Table 2.5: Description of use case "Comment a post"

Use case	Comment a post		
Code	UC04		
Actor	User		
Description	An user comments on created post		
Pre-condition	An user has logged into the system.		
Post-condition	An user comments successfully.		
Main scenario (success)	**ON**	**Actors**	**Actions**
	1	User	Click on the text box "Enter comments..."
	2	System	Display create post form interface
	3	User	Enters the content of the post, uploads files, add destination and then click "Post" button.
	4	System	Announces post successfully, close popup, update list of user posts.
Extensions	None		
Alternative scenario	**ON**	**Actors**	**Actions**
	1	System	If the request fails, the system notifies the user.

2.3.5 Description of use case "Search"

Table 2.6: Description of use case "Search"

Use case	Search		
Code	UC05		
Actor	User		
Description	An user search destination, hotel or other user		
Pre-condition	An user has logged into the system.		
Post-condition	An user successfully searched for the necessary information		
Main scenario (success)	**ON**	**Actors**	**Actions**
	1	User	Click on the search input
	2	System	Focus input
	3	User	Enter keyword to search
	4	System	Show part of recent search results
	5	User	Click on the search results
	6	System	Redirect to detail page
Extensions	None		
Alternative scenario	**ON**	**Actors**	**Actions**
	1	User	Click "Show all result".
	2	System	The system redirects to all results page.

2.3.6 Description of use case "Rate a destination"

Table 2.7: Description of use case "Rate a destination"

Use case	Rate destination		
Code	UC05		
Actor	User		
Description	An user rates destination		
Pre-condition	An user has logged into the system.		
Post-condition	An user successfully rated for the destination		
Main scenario (success)	**ON**	**Actors**	**Actions**
	1	User	User go to detail destination page, and select rating input box
	2	System	Focus input
	3	User	Enter a comment and select a review score, then click post button
	4	System	Show comment at the top of list comments
Extensions	None		

2.3.7 Description of use case "Explore destinations"

Table 2.8: Description of use case "Explore destinations"

Use case	Explore destinations		
Code	UC05		
Actor	User		
Description	An user can see suggested destinations.		
Pre-condition	An user has logged into the system.		
Post-condition	An user successfully see suggested destinations.		
Main scenario (success)	**ON**	**Actors**	**Actions**
	1	User	Click "Explore" icon on navigation bar
	2	System	Show list suggested destinations
Extensions	None		

2.3.8 Description of use case "Like a post"

Table 2.9: Description of use case "Like a post"

Use case	Like a post		
Code	UC05		
Actor	User		
Description	An user can express their feelings with a post.		
Pre-condition	An user has logged into the system.		
Post-condition	An user successfully like a post.		
Main scenario (success)	**ON**	**Actors**	**Actions**
	1	User	Click "Like" icon at the bottom of a post
	2	System	Change color of "Like" icon
Extensions	None		

2.4 Non-functional requirement

- The system will work on any Web browser
- User-friendly interface, easy to use
- Fast data retrieval, good data storage capacity
- Quick and convenient search
- High security
- Easy to expand to add new features

CHAPTER 3. METHODOLOGY

In this chapter, I present about the technologies used in the project, the reasons for their use, their advantages and disadvantages.

3.1 Backend

3.1.1 Java Spring Boot

Java Spring Framework (Spring Framework) is a popular, open source, the enterprise-level framework for creating standalone, production-grade applications that run on the Java Virtual Machine (JVM) [1] Java Spring Boot (Spring Boot) is a tool that makes developing the web application and microservices with Spring Framework faster and easier through three core capabilities:

- Auto configuration.
- An opinionated approach to configuration.
- The ability to create standalone applications.

Spring Framework offers a dependency injection feature that lets objects define their dependencies that the Spring container later injects into them. This enables developers to create modular applications consisting of loosely coupled components that are ideal for microservices and distributed network applications.[2]

Figure 3.1: Spring boot (Source: Internet).

Spring Framework also offers built-in support for typical tasks that an application needs to perform, such as data binding, type conversion, validation, exception handling, resource and event management, internationalization, and more. It integrates with various Java EE technologies such as RMI (Remote Method Invocation), AMQP (Advanced Message Queuing Protocol), Java Web Services, and others. In sum, Spring Framework provides developers with all the tools and features they need to create loosely coupled, cross-platform Java EE applications that run in any environment.

3.1.2 Redis

Redis is an open-source key-value database. Data in a key-value database has two parts: the key and the value. Because Redis can accept keys in a wide range of formats, operations can be executed on the server and reduce the client's workload. Redis is often used for cache management and speeding up web applications.[3]

Figure 3.2: Redis (Source: Internet).

Advantage:

- Redis has excellent read-write performance, fast IO read-write speed from memory, and supports read-write frequencies of more than 100k + per second.
- Redis supports string, lists, hashes, sets, ordered sets and other data type operations.
- Redis supports data persistence, AOF and RDB
- Redis supports master-slave replication. The host will automatically synchronize the data to the slave, allowing read-write separation.
- All redis operations are atomic, and redis also supports atomic execution after full consolidation of several operations.

Disadvantages:

- Master-slave synchronization, data synchronization will be delayed. If the host goes down and some data is not synchronized to the slave before the shutdown, the data will be inconsistent.
- It is difficult to support online capacity expansion. When the cluster capacity reaches the upper limit, the online capacity expansion will become very complex. When the system goes online, enough space must be ensured, which causes a great waste of resources.

3.1.3 Cloudinary

Figure 3.3: Cloudinary (Source: Internet).

Cloudinary is an end-to-end image- and video management solution for websites and mobile apps, covering everything from image and video uploads, storage, manipulations, and optimizations to delivery. With Cloudinary, we can easily upload images and videos to the cloud and automate smart manipulations of those media without installing any other software. Cloudinary then seamlessly delivers your media through a fast content delivery network (CDN), optimized with the industry's best practices. Additionally, Cloudinary offers comprehensive APIs and administration capabilities, which you can easily integrate with your web and mobile apps. Cloudinary has only one limitation, that is we only have 25 GB of storage for the free version.

3.1.4 PostgreSQL

PostgreSQL is an open-source relational database management system (DBMS) developed by a worldwide team of volunteers. [4] PostgreSQL is not controlled by any corporation or other private entity and the source code is available free of charge. PostgreSQL outperforms MongoDB in almost all performance test cases. The PostgreSQL database management system (DBMS) measured between 4 and 15 times faster than MongoDB in transaction performance.

Advantages of PostgreSQL:

- PostgreSQL can run dynamic websites and web apps as a LAMP stack option.
- PostgreSQL's write-ahead logging makes it a highly fault-tolerant database.
- PostgreSQL source code is freely available under an open source license. This allows you the freedom to use, modify, and implement it as per your business needs.
- PostgreSQL supports geographic objects so you can use them for location-based services and geographic information systems.
- PostgreSQL supports geographic objects so it can be used as a geospatial data store for location-based services and geographic information systems.
- To learn Postgres, you don't need much training as it is easy to use.

- Low maintenance and administration for both embedded and enterprise use of PostgreSQL.

Figure 3.4: PostgreSQL (Source: Internet).

Disadvantages of PostgreSQL:

- Postgres is not owned by one organization. So, it has had trouble getting its name out there despite being fully featured and comparable to other DBMS systems
- Changes made for speed improvement require more work than MySQL as PostgreSQL focuses on compatibility
- Many open source apps support MySQL, but may not support PostgreSQL
- On performance metrics, it is slower than MySQL.

3.1.5 Kafka

Figure 3.5: Docker (Source: Internet).

Apache Kafka is a distributed publish-subscribe messaging system that receives data from disparate source systems and makes the data available to target systems in real-time. Kafka is written in Scala and Java and is often associated with real-time event stream processing for big data.[5]

There are many benefits when using kafka

- Kafka is highly scalable. Kafka is a distributed system, which can be scaled quickly and easily without incurring any downtime. Apache Kafka can handle many terabytes of data without incurring much at all in the way of overhead.
- Kafka is highly durable. Kafka persists the messages on the disks, which pro-

vides intra-cluster replication. This makes for a highly durable messaging system.

- Kafka is highly reliable. Kafka replicates data and can support multiple subscribers. Additionally, it automatically balances consumers in the event of failure. That means that it's more reliable than similar messaging services available.
- Kafka offers high performance. Kafka delivers high throughput for both publishings and subscribing, utilizing disk structures that are capable of offering constant levels of performance, even when dealing with many terabytes of stored messages.

3.1.6 Docker

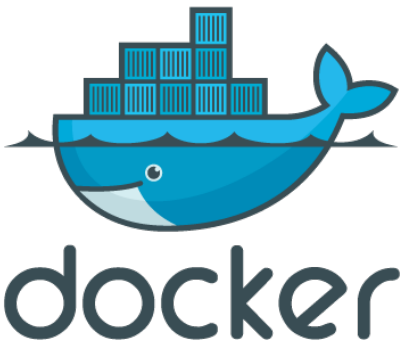

Figure 3.6: Docker (Source: Internet).

Docker is a virtual machine, but unlike virtual machines that create a completely separate operating system.[6] Docker allows the applications to use the Linux kernel of the same machine on which it is installed. By using this benefit, it can make the applications ready to ship to other machines running the same Linux OS with somewhat different configurations. Through Docker DevOps, developers can easily pack all parts of an application like libraries and other dependencies and ship it out as a single package.

3.2 Frontend

3.2.1 React

Figure 3.7: React (Source: Internet)

React.js,[7] more commonly known as React, is a free, open-source JavaScript library. It works best to build user interfaces by combining sections of code (components) into full websites. Originally built by Facebook, Meta and the open-source community now maintain it. One of the good things about React is that you can use it as much or as little as you want! For example, we can build your entire site in React or just use one single React component on one page.[8] There are many other famous FE libraries or frameworks such as Vuejs, AngularJs. So why would we choose React instead of Vue.js, or vice versa? The big bonus to React is that it's maintained by Meta – a tech giant. Such strong support from a main player in the tech world provides React the stability and long-term support that most libraries just don't have. This gives developers the confidence that React won't be deprecated in the near future, and developments will continue to improve it. Besides, React is better than Angular due to its virtual DOM implementation and rendering optimizations. Migrating between React's versions is quite easy, too; you don't need to install updates one by one, as in the case of Angular. Finally, with React, developers have myriads of existing solutions they can use

3.2.2 Redux saga

Figure 3.8: Redux saga (Source: Internet).

Redux Saga is a middleware library used to allow a Redux [9] store to interact with resources outside of itself asynchronously. This includes making HTTP requests to external services, accessing browser storage, and executing I/O operations. These operations are also known as side effects. Redux Saga helps to organize these side effects in a way that is easier to manage.

We can allocate three key benefits of sagas:

- Simplicity in organiziting difficult side effects sequences
- Declarative style
- Simplicity of testing

3.2.3 Antd

Figure 3.9: Antd (Source: Internet).

Antd is a collection of React components built according to the design standards of the Ant UED Team. Similar to the Material Design standard, Ant provides most of the common components in modern web applications, like Layout, Button, Icon, DatePicket, and more. . . Besides that, Ant also has its interesting components, like the LocaleProvider that allows you to change the language across the application. Ant currently has over 51k stars on Github. Ant Design for React can be considered as a collection of most React libraries. It meets most of your project's requirements without you having to install any additional libraries.

Antd is a FE-friendly library of pre-provided components that make it easy for programmers to customize without spending much time and effort while still ensuring quality.

CHAPTER 4. SYSTEM ANALYSIS AND DESIGN

4.1 Architecture design

4.1.1 Software architecture selection

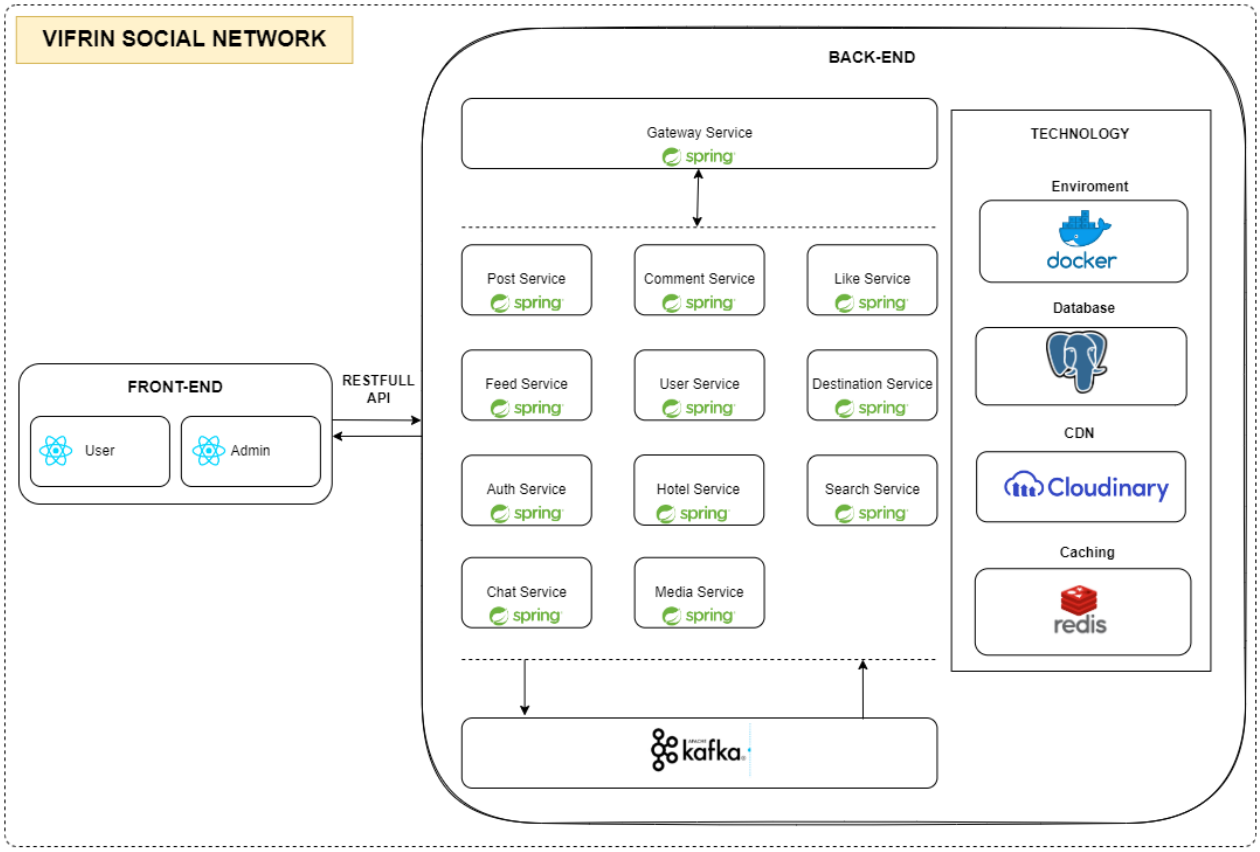

Figure 4.1: Overall architecture of the system

The figure above depicts the architecture of the system that will include frontend and backend. The frontend has 2 parts, an admin page and a user page which use ReactJs to build UI/UX components and interact with the backend through Rest APIs. The global state in ReactJS is managed by the redux-saga, which makes components easier to test, and the application architecture is cleaner. Components are injected only when used, resulting in better performance. I have developed a tool to make code faster, and easier to interact with API. This will be explained in more detail in chapter 4. The backend uses a microservice system [10], divided into 11 services. Each service is implemented separately, to ensure that they can work independently of each other. They are all connected to the database as PostgreSQL, each service will be attached to manipulate a table in the database. All requests from the front end using restful API will be handled by the backend gateway service. Gateway service is supported by java spring boot in configuring which

request streams should be handled by which service. Services communicate with each other by listening to Kafka queues. That means those services will subscribe to a queue to listen to and handle events sent by other services. In addition, the system uses Redis to store cache queries. To create an environment for development and deploy, I used docker and docker-compose to manage the config and version of environment.[11]

+ Details of available services in the backend system:

- **Gateway Service**: all APIs request go through this gateway, which is then redirected to other services.
- **Auth Service**: implement the registration, login API
- **User Service**: update user information, follow other users
- **Post Service**: add, edit, delete, view, save posts
- **Comment Service**: take care of APIs related to adding, editing, deleting comments
- **Like Service**: like the post or like the comment
- **Destination Service**: responsible for tourist destination functions
- **Hotel Service**: responsible for hotel functions
- **Feed Service**: aggregate feeds for users
- **Media Service**: interact with CDN to save media
- **Search Service**: responsible for the function of returning search results
- **Chat Service**: manage chats between users
- **Docker**: run services like database, Kafka and deploy service into containers
- **Cloudinary**: storage of multimedia files (photos, videos)
- **Redis**: save information on cache, serve to save some important information and retrieve information quickly
- **Kafka**: assume the role of communication between services, handle asynchronous events.

+ With both admin and user sites, I use ReactJS to build screens. I use redux-saga to handle asynchronous logic. This layer also handles the business logic, we won't handle too much logic in screens/components(views). The structure of the front-end is designed with folders and functions as shown:

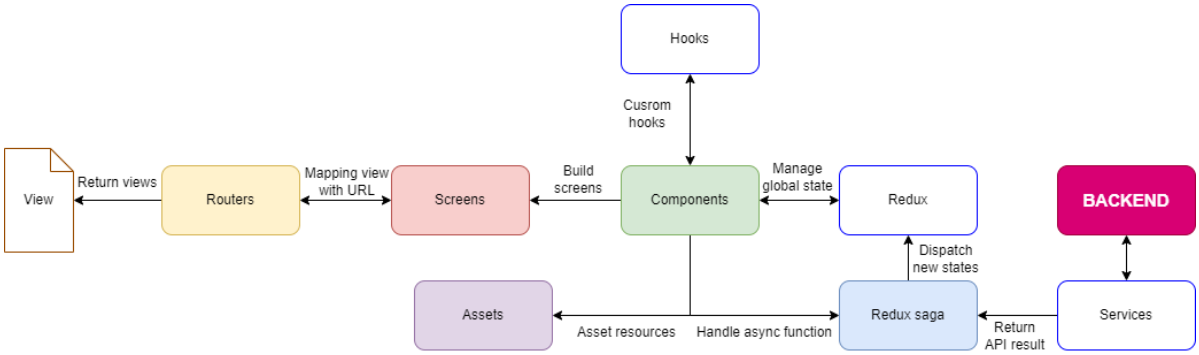

Figure 4.2: Front-end overview design

4.1.2 Overall design

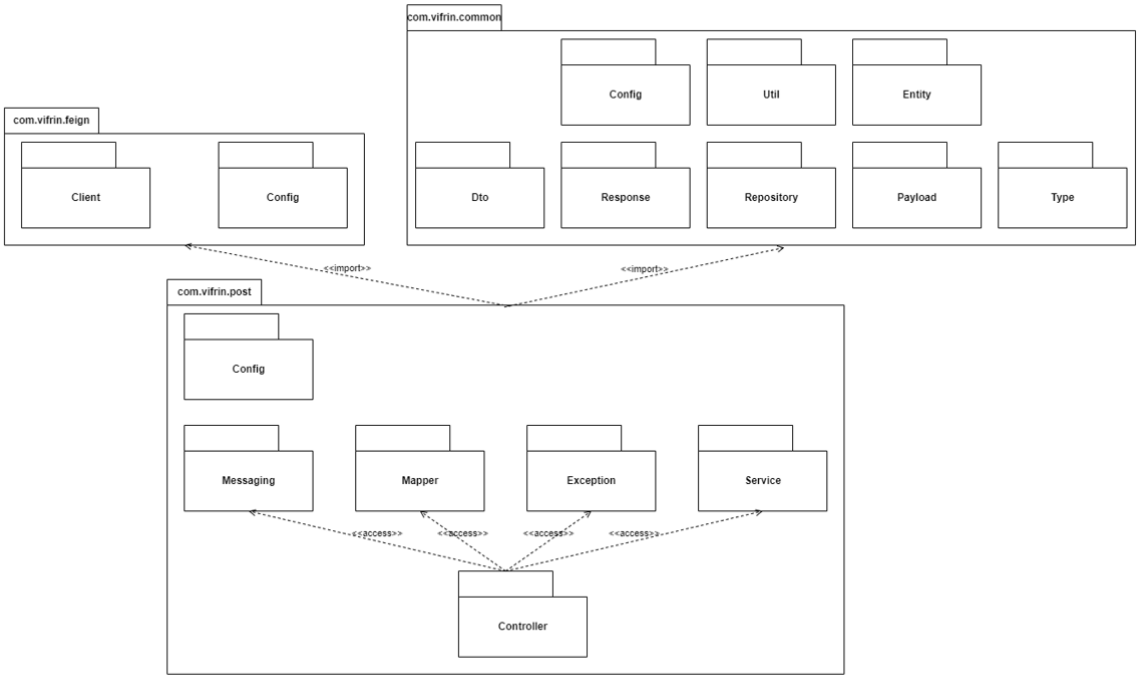

Figure 4.3: Package diagram of post service

The packages of service will have the same structure as a post-service (above figure). Post-service will use 2 packages (i) **com.vifrin.common**: where the system's shared data types are stored (eg entity, response structure definition, system's shared constant functions ...). These data types are not divided into each service, but centralized management in one place will have certain benefits. For example, when we change a field in another database, the definition of the entity changes/calls the result of this service from service if the management is distributed in each service, then just change the classes in the package. in this package, (ii) **com.virgin.feign**: used to call API from 1 service from another service, must finish calling, get results, then the program will continue to run. This is different

from using Kafka in that if you fire an event on Kafka, an event is an asynchronous event, there is no need to wait for the result of the action of firing the event on Kafka. Next, we will clarify the function of the sub-packages in each service.

+ com.vifrin.feign:

- config:
- client: define API to call between services. For example, we can using @GetMapping("/users"), an anotation imported by org.springframework.cloud.// openfeign.FeignClient to call API of user service from another service.

+ com.vifrin.common:

- config: contain common config of web (page size of list, default page number,..), redis, operator with data base(create, read, update, delete).
- util: common utils(operations with redis,..)
- dto: This only contain data, not business logic. In this system, when we get result from DB, data will tranfer to store in DTO.
- response: define common format template
- repository: using org.springframework.stereotype.Repository package to create function to get data from database.
- payload: define payload of request which sent from client side
- type: common type of system(action type, notification type,...)
- entity: define field property and relationship between them in database

+ com.vifrin.post:

- config: custom sercurity config, for example check token before handle request
- message: define message to send to kafka
- mapper: convert results when query from database to dto
- exception: contains error configurations that will be returned to the user when the program encounters an error
- service: define the functions for the business logic, using the help of the repository

4.1.3 Detailed package design

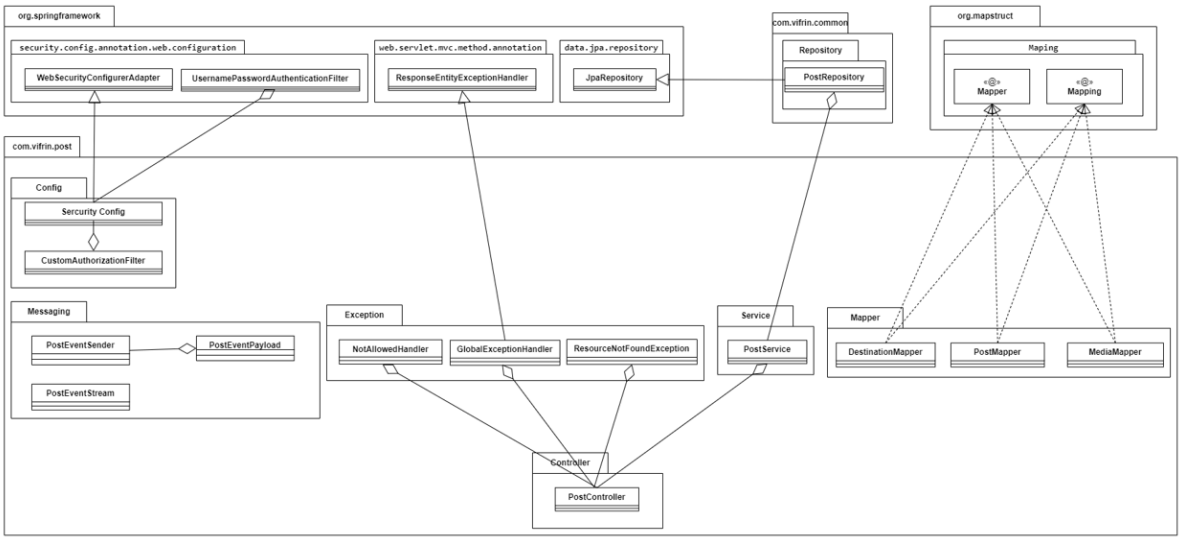

Figure 4.4: Detail package diagram of post service

The main feature of this design is dependency injection.[12] This is a very famous technique in software design. Spring Boot is built on top of the Spring Framework, it uses XML and annotation to create stand-alone, production-grade Spring-based applications more easily. Dependency Injection is evident in the architecture of spring boots. As shown in the figure, class **Sercurity Config** is similar to a middleware, used to check the requirements of the request. Class **Sercurity Config** extends from WebSecurityConfigurerAdapter, which is supported by spring framework. All requests must pass the check of this class before going to the next handler. This class injects filters: **CustomAuthorizationFilter** (for decode token in header of request). I will take another example. Class **Post Repository** is injected into class **Post Service**. Of course, we can inject it with other repositories. All repositories extend from class **JpaRepository**, which supports writing very concise queries. We'll talk more about it in 6.

4.2 Detailed design

4.2.1 User interface design

The system interface is designed with a 16: 9 ratio, 1920x1080 resolution. Application support screen size for laptops, tablets, and mobile. The layout includes a header, and content components arranged as shown below.

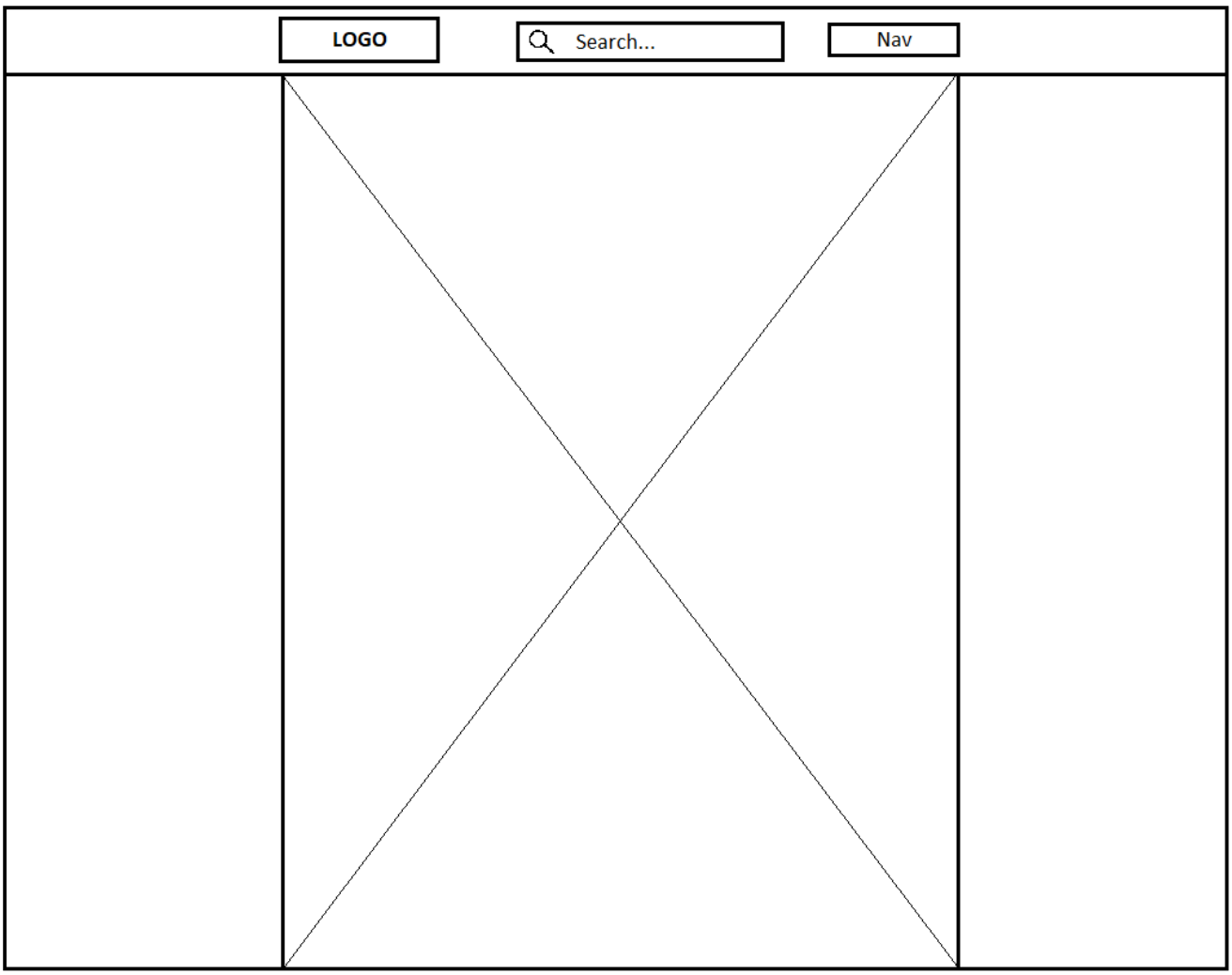

Figure 4.5: General system interface wireframe

The header will include a logo, search box, and navigation bar (home, explore, profile). The colors will be mixed according to the main color tone to increase the consistency of the system. The table below will present the uniform information of the interface.

Table 4.1: Interface property information list table

Attribute name	**Attribute configuration**
Color	Primary Color : #03A9F4, Other color : #1976D2
Text	Font: VFSans-Regular, sans-serif; Text color: #333
Button	Normal button: #1976D2, Delete button: #FF0A03
Modal position	The center of screen
Language	Vietnamese, English
Notification	Show when there was condition triggered

Below are wireframes of some other screens:

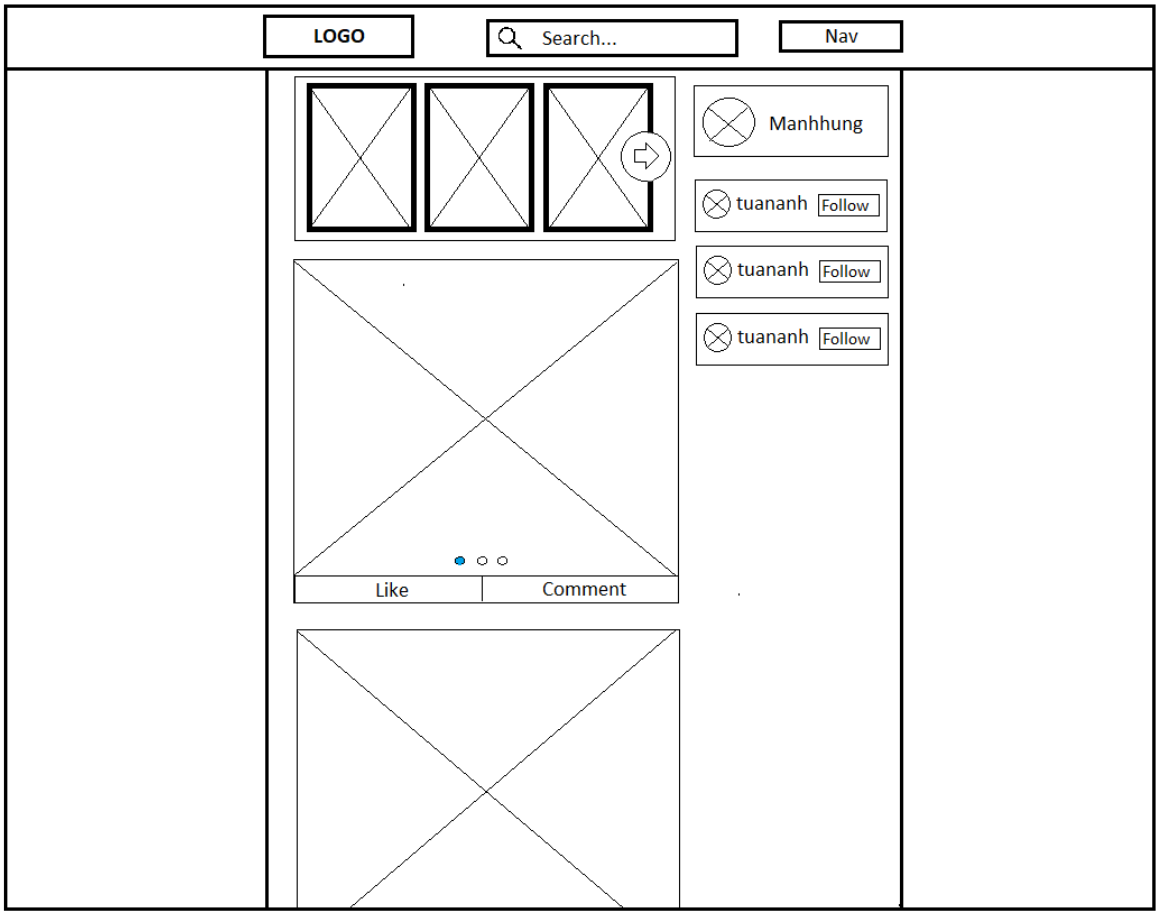

Figure 4.6: Wireframe of homepage

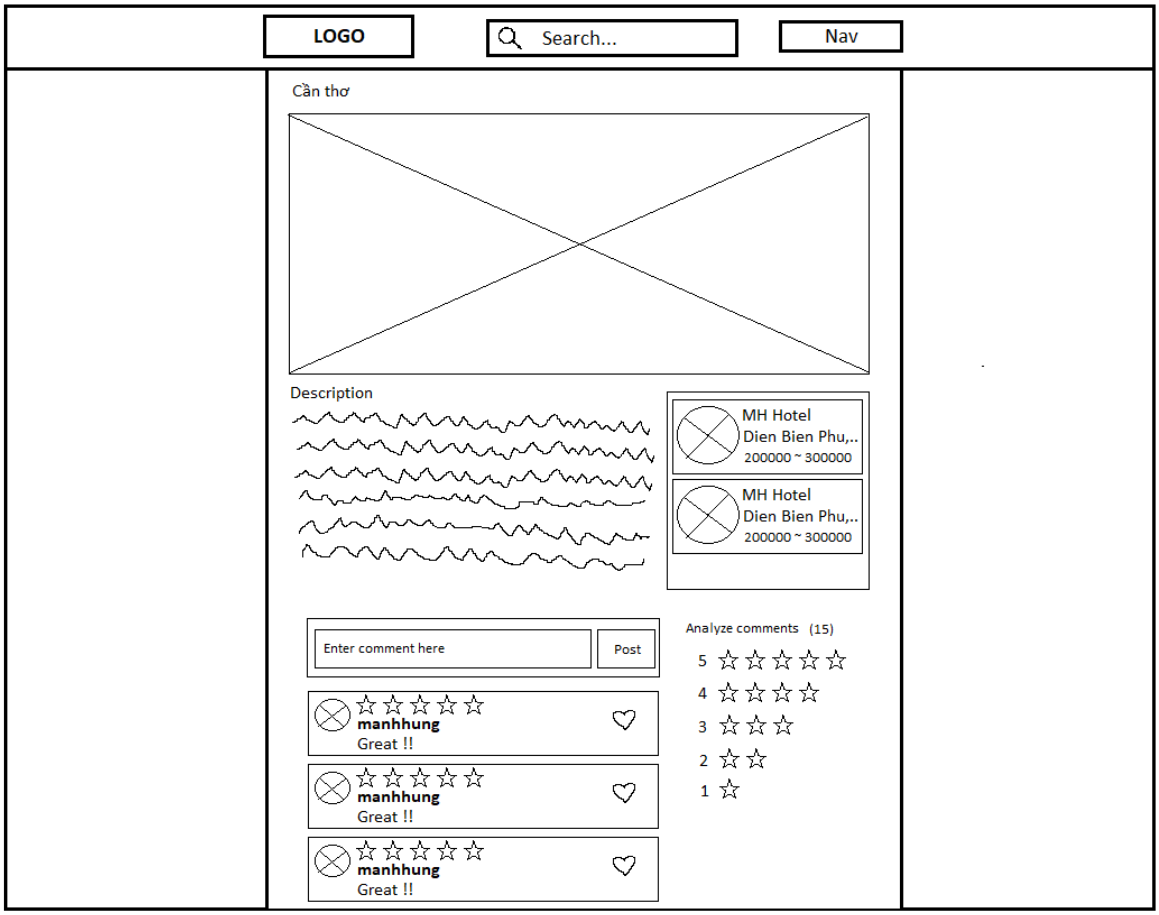

Figure 4.7: Wireframe of destination

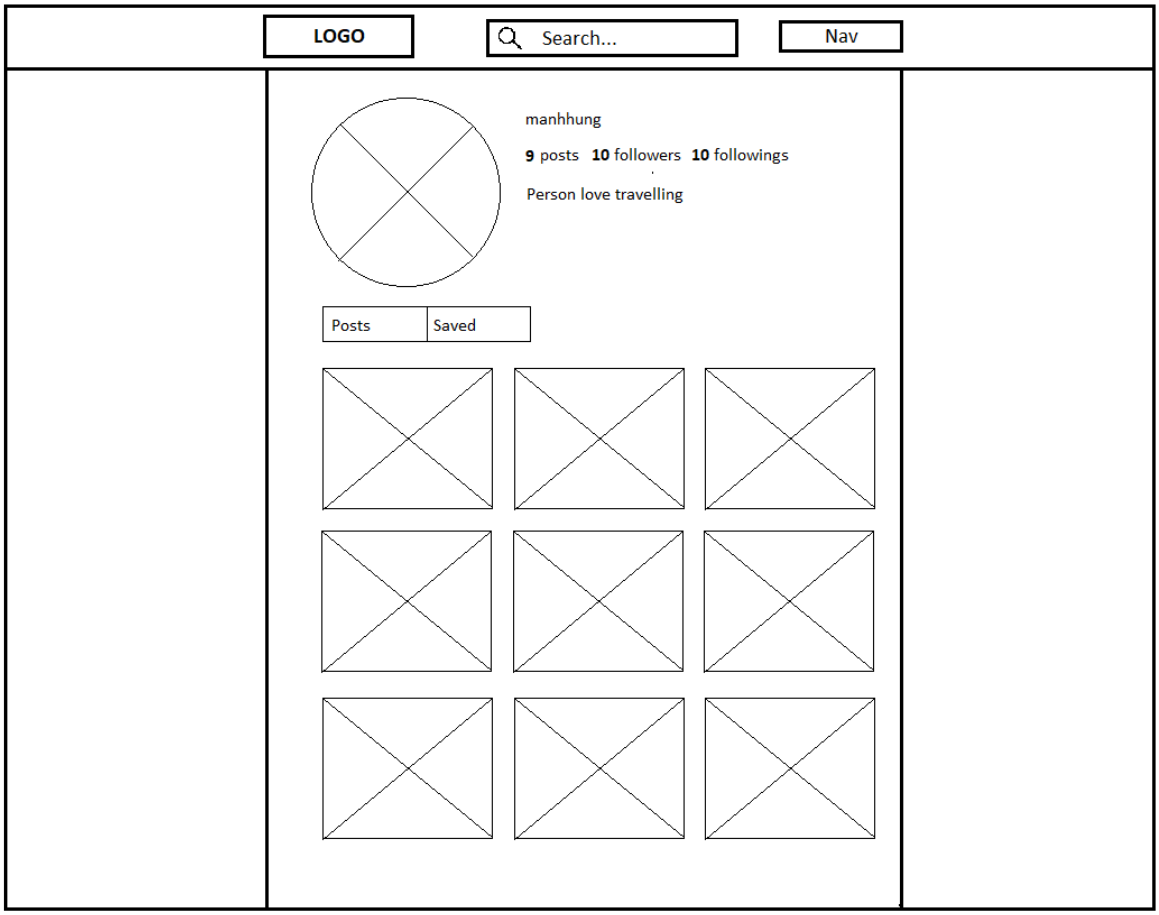

Figure 4.8: Wireframe of profile

4.2.2 Layer design

Detailed design of properties and methods for some of the most important classes of post service are displayed in figure:

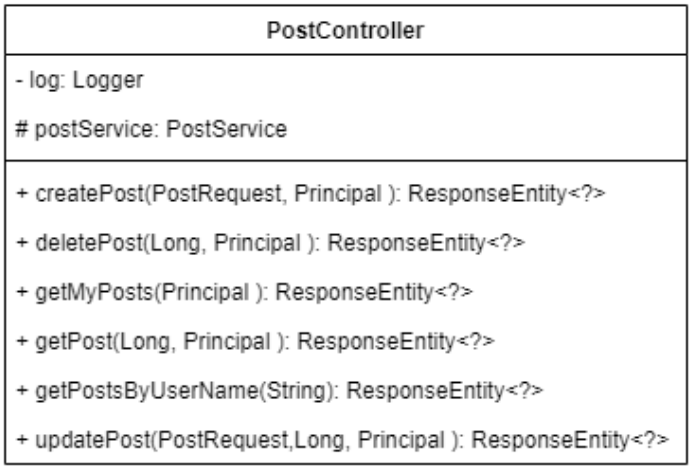

Figure 4.9: Class diagram of PostController.java

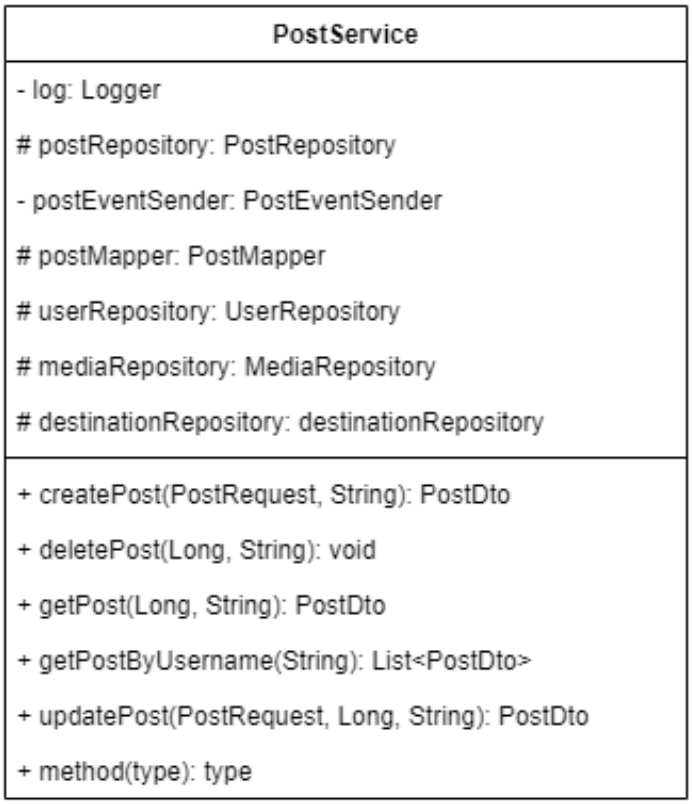

Figure 4.10: Class diagram of PostService.java

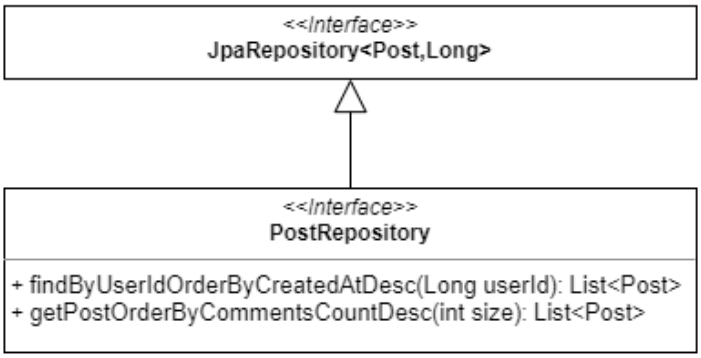

Figure 4.11: Class diagram of PostRepository.java

4.2.3 Sequence diagrams

4.2.4 Sequence diagram of "Login" feature

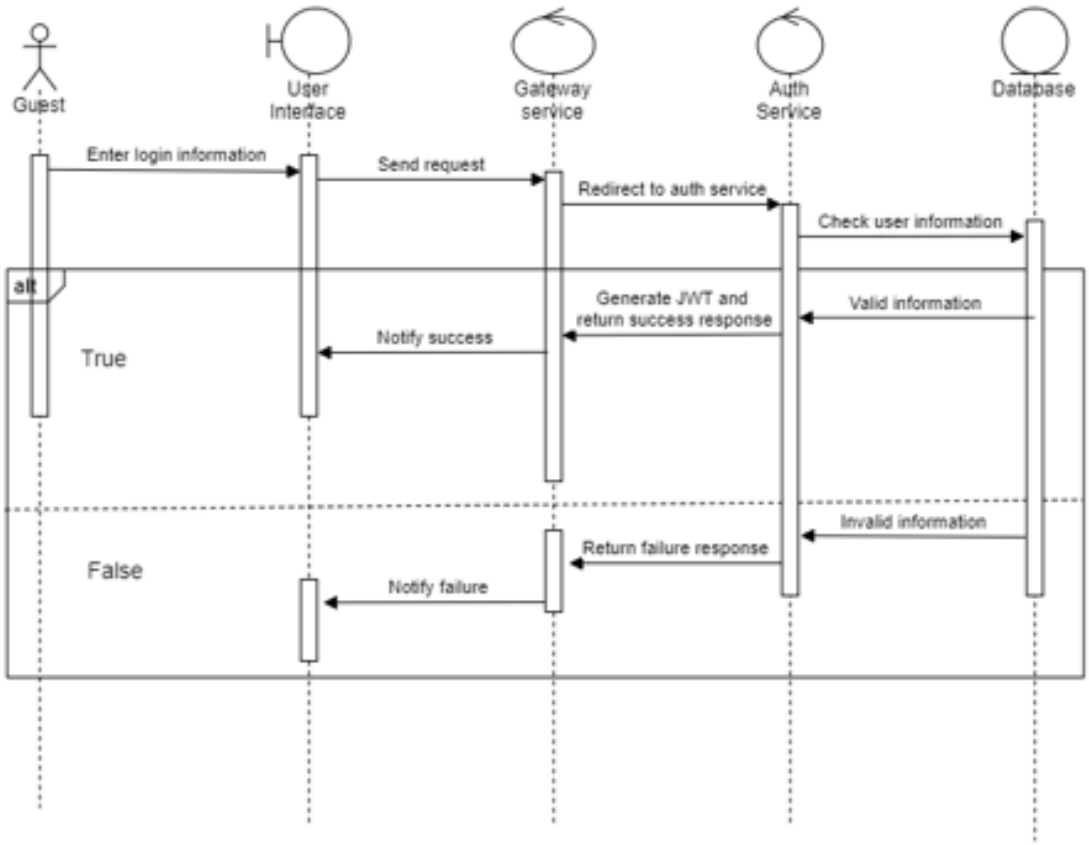

Figure 4.12: Sequence diagram of "Login" feature

4.2.5 Sequence diagram of "Search" feature

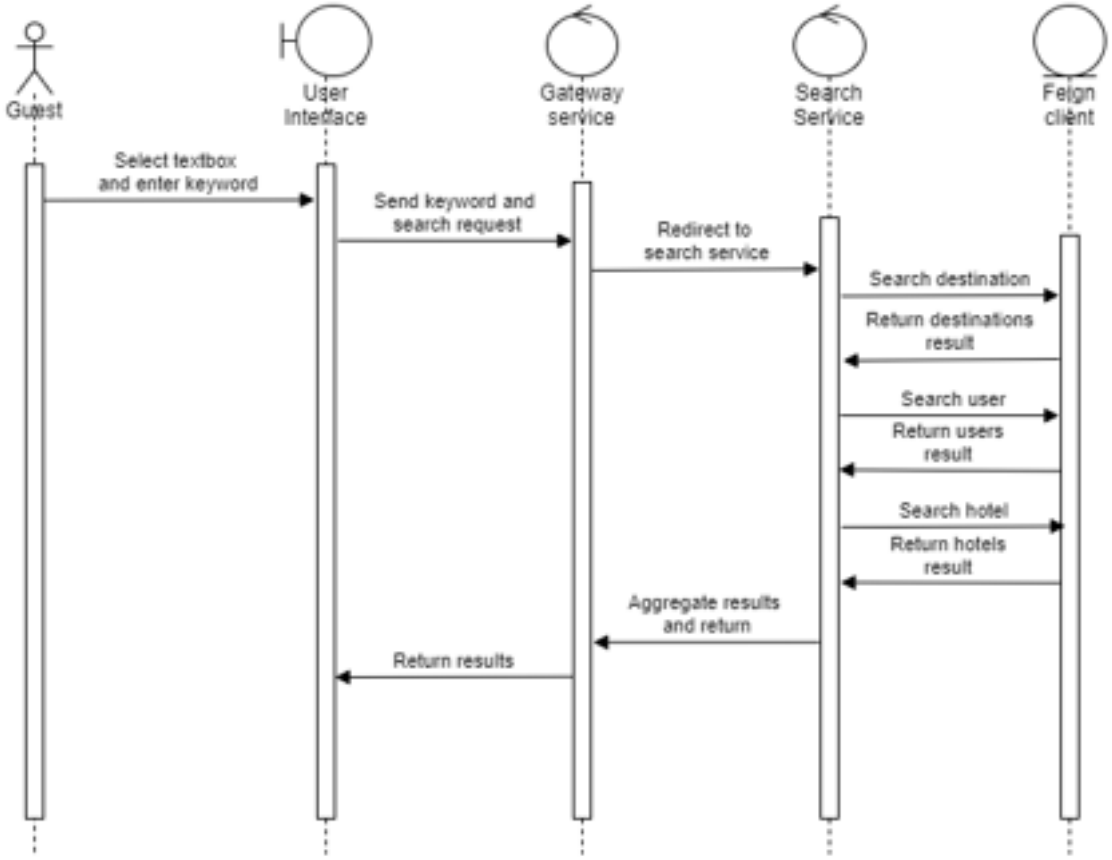

Figure 4.13: Sequence diagram of "Search" featur

4.2.6 Sequence diagram of "Create post" feature

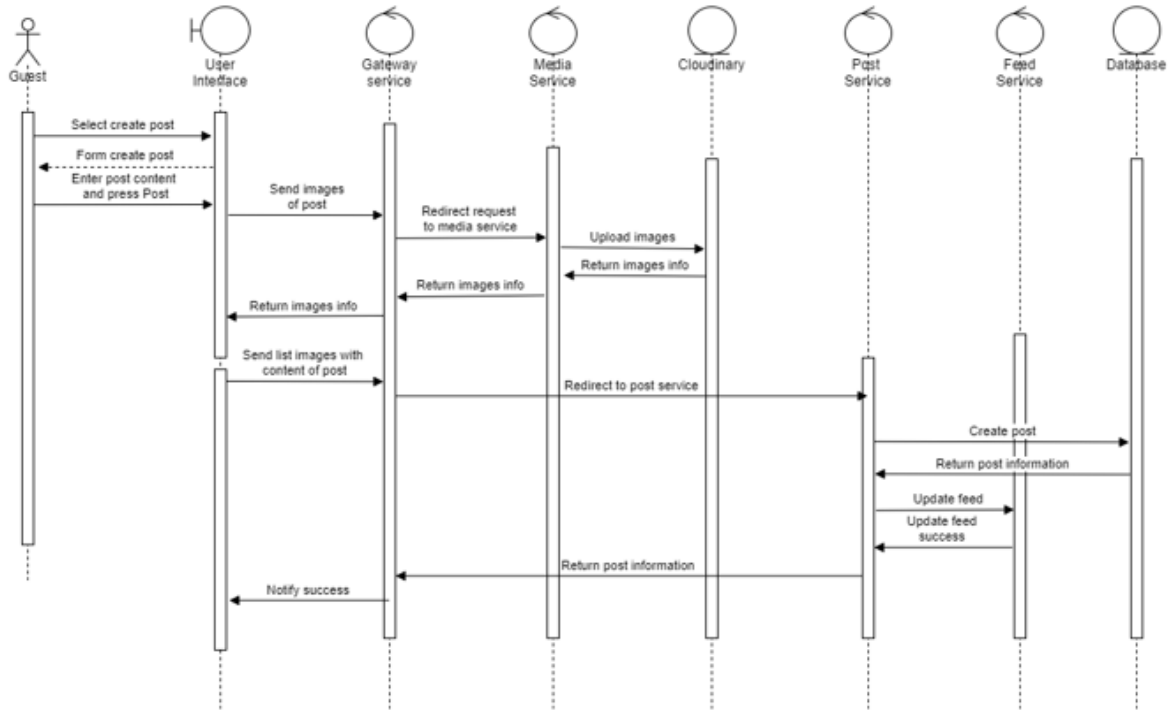

Figure 4.14: Sequence diagram of "Create post" featur

4.2.7 Database design

a, Entity relationship diagram

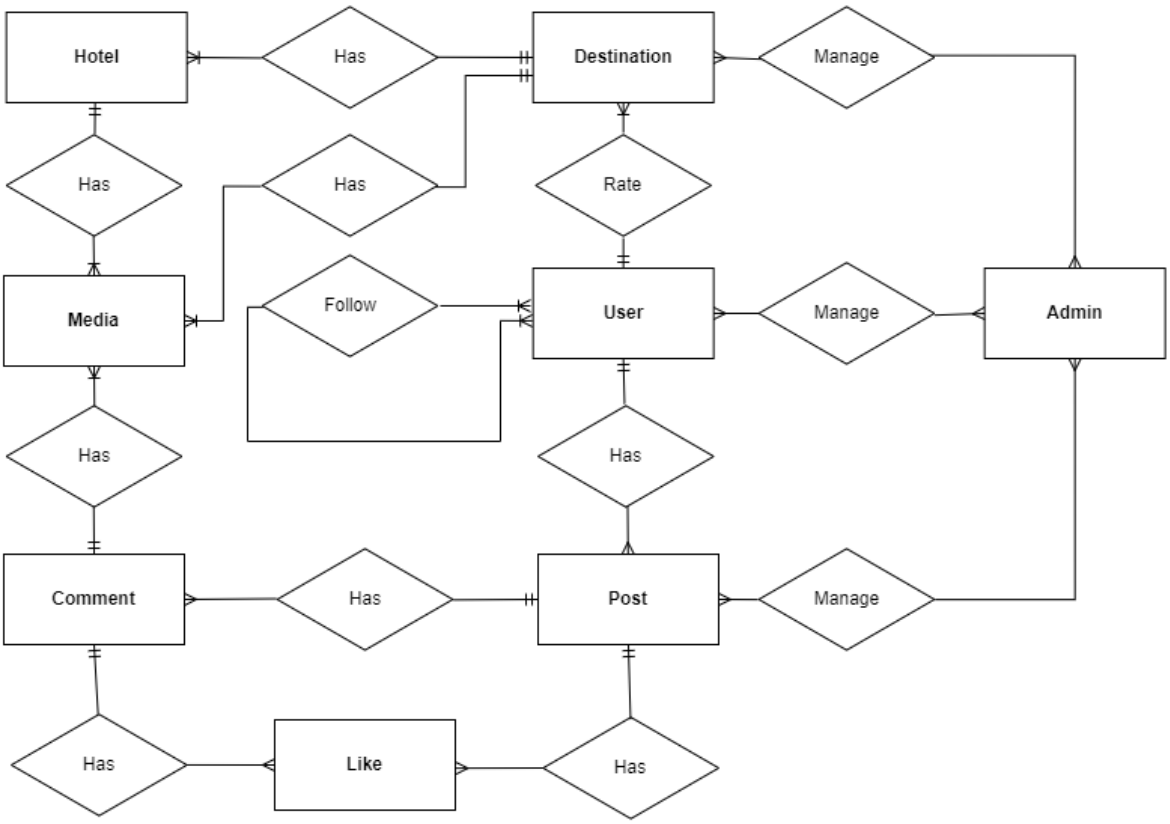

Figure 4.15: Entity relationship diagram

About the entity relationship diagram including entities: Admin, User, Destination, Hotel, Post, Comment, Media. Users are the main target of the system, need to register for an account before using the functions. After signing up for an account, users can follow other users to see other users' latest posts on the feed. Users can also create multiple posts, each post can have many photos and many comments. Each comment can have multiple photos. Posts and comments can be liked. The system has many destinations, each destination can have many hotels. Users can rate that destination through review scores or comments. The administrator is the person who manages the normal users, the destination, and the hotels of that destination.

b, Database design

The social network system uses a relational database (SQL) with the advantages mentioned in the previous section. Database using PostgreSQL technology.

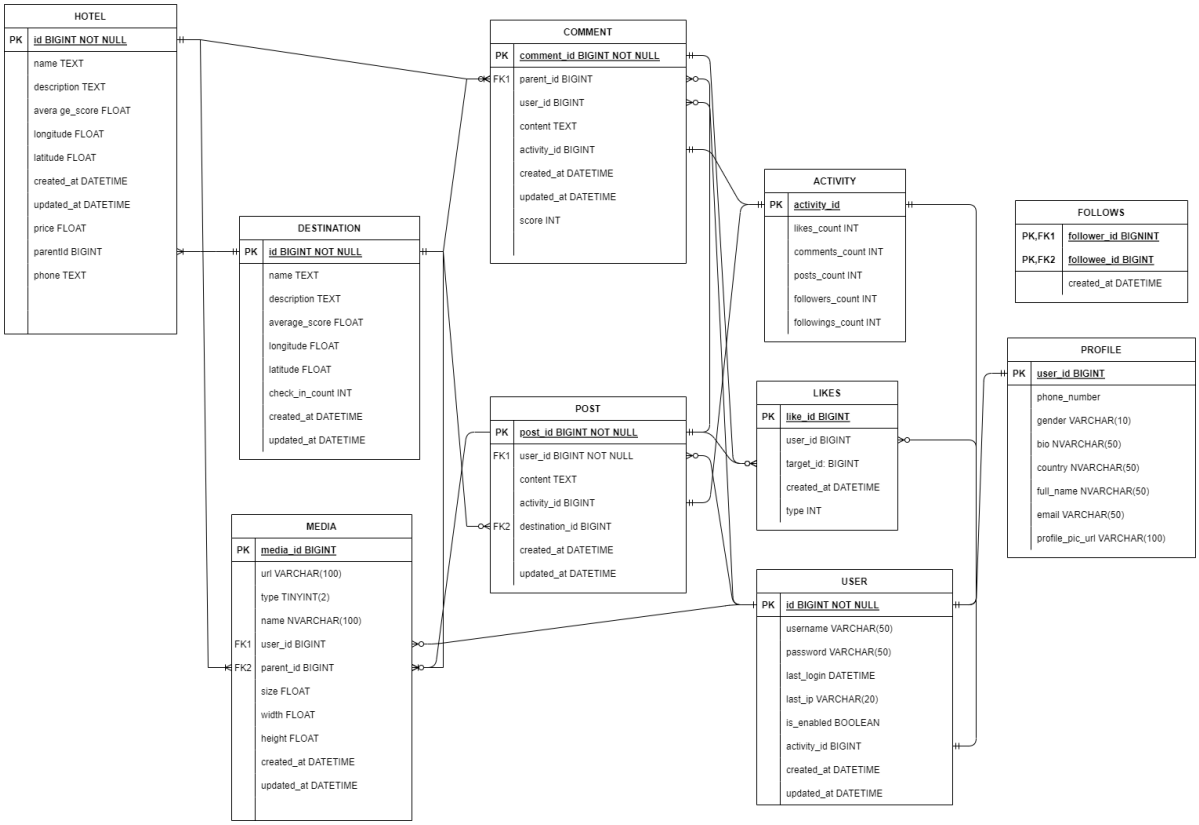

Figure 4.16: General system interface wireframe

Description of the tables in the database

- **Users**
  Store information about users participating in the system, including name, username, avatar url,..
- **Profile**
  Store profile of users participating in the system, including phone, gender, bio, full name,..
- **Likes**
  Store list people who liked the post
- **Activity**
  Store user activities such as number of comment, follower, posts,...
- **Post**
  Store posts that user shares
- **Comment**
  Store comments of destination, post
- **Media**
  Store information of media such as url, width, height,...

- **Destination**

  Store destinations information

- **Hotel**

  Store hotels information

c, Detail database design

+ Users

Table 4.2: User table

ON	Field name	Field type	Primary key	Foreign key	Description
1	id	long	x		User code
2	username	nchar(100)			Username
3	password	nchar(100)			Password
4	avatar_url	nchar(500)			Avatar url
5	is_enabled	boolean			Account status
6	last_login	DateTime			Last time login
7	role	nchar(100)			Account role
8	created_at	DateTime			Created time
9	updated_at	DateTime			Updated time

+ Profile

Table 4.3: Profile table

ON	Field name	Field type	Primary key	Foreign key	Description
1	user_id	long	x		User code
2	email	nchar(100)			Email
3	phone_number	nchar(100)			Phone number
4	bio	nchar(100)			Biography
5	full_name	nchar(100)			Full name
6	gender	nchar(100)			Gender
7	date_of_birth	DateTime			Date of birth
8	country	nchar(100)			Country

+ Destination

Table 4.4: Destination table

ON	Field name	Field type	Primary key	Foreign key	Description
1	id	long	x		Destination code
2	name	text			Destination name
3	description	text			Description
4	average_score	float			Average score
5	longitude	float			Longitude
6	latitude	float			Latitude
7	check_in_count	int			Check in count
8	created_at	DateTime			Created time
9	updated_at	DateTime			Updated time

+ Hotel

Table 4.5: Hotel table

ON	Field name	Field type	Primary key	Foreign key	Description
1	id	long	x		Destination code
2	name	text			Destination name
3	description	text			Description
4	average_score	float			Average score
5	longitude	float			Longitude
6	latitude	float			Latitude
7	address	text			Latitude
8	price	float			Price
9	sales_price	float			Sales price
10	phone	text			Phone number
11	has_air_conditioner	boolean			Hotel has air conditioner or not
12	has_bathroom	boolean			Hotel has bathroom or not
13	has_elevator	boolean			Hotel has elevator or not
14	has_parking	boolean			Hotel has parking or not
15	has_restaurant	boolean			Hotel has restaurant or not
16	has_swimming_pool	boolean			Hotel has swimming pool or not
17	has_wifi	boolean			Hotel has wifi or not
18	created_at	DateTime			Created time
19	updated_at	DateTime			Updated time
20	destination_id	long			Destination code

+ Post

Table 4.6: Post table

ON	Field name	Field type	Primary key	Foreign key	Description
1	id	long	x		Post code
2	user_id	long		x	Creator code
3	activity_id	long		x	Activity code
4	destination_id	long		x	Destination code
5	content	text			Content of post
6	config	float			Config: public or private

+ Media

Table 4.7: Media table

ON	Field name	Field type	Primary key	Foreign key	Description
1	id	long			Media code
2	url	nchar(500)			Media url
3	mime	nchar(500)			Mime of media
4	name	nchar(500)			Name
5	width	float			Width
6	height	float			Height
7	size	float			Size
8	created_at	DateTime			Created time
9	updated_at	DateTime			Updated time
10	post_id	long		x	Post code
11	user_id	long		x	User code
12	comment_id	long		x	Comment code
13	destination_id	long		x	Destination code

+ Comment

Table 4.8: Comment table

ON	Field name	Field type	Primary key	Foreign key	Description
1	id	long			Media code
2	content	text			Media url
3	star	int			Name
4	created_at	DateTime			Created time
5	updated_at	DateTime			Updated time
6	post_id	long		x	Post code
7	user_id	long		x	User code
8	destination_id	long		x	Destination code

+ Likes

Table 4.9: Likes table

ON	Field name	Field type	Primary key	Foreign key	Description
1	id	long			Media code
2	type	text			Media url
3	created_at	DateTime			Created time
4	post_id	long		x	Post code
5	user_id	long		x	User code

+ Activity

Table 4.10: Activity table

ON	Field name	Field type	Primary key	Foreign key	Description
1	activity_id	long			Activity code
2	likes_count	text			Number of likes
3	comments_count	DateTime			Number of comments
4	post_count	long		x	Number of posts
5	followers_count	long		x	Number of followers

+ Follow

Table 4.11: Follow table

ON	Field name	Field type	Primary key	Foreign key	Description
1	following_id	long			User following code
2	follower_count	long			User follower code
3	created_at	DateTime			Created time

CHAPTER 5. SYSTEM DEPLOYMENT AND EVALULATION

5.1 Application Building

5.1.1 Libraries and Tools

Purpose	Tools	URL	Version
IDE	Visual studio code	https://code.visualstudio.com/	1.69
IDE	Intellij IDEA Community Edition	https://www.jetbrains.com/	2022.1.2
APIs testing	Postman	https://www.postman.com/	v9.25.2
Package Manager	NPM	https://www.npmjs.com/	8.1.0
Database manager	pgAdmin	https://www.pgadmin.org/	4.0.0
Redux Flow Monitor	Redux Dev Tools	Chrome Extensions Store	4.25.0
Browser Debugging Tool	Chrome Dev Tools	https://developer.chrome.com	v9.25.2

Table 5.1: List of libraries and tools used

5.1.2 Achievement

After the research and implementation process, I have developed a tourism social network system that can be easily deployed to put into practice. The packaged product includes backend and frontend source code, and a docker-compose file to easily package services into containers on the server environment.

5.1.3 Illustration of main functions

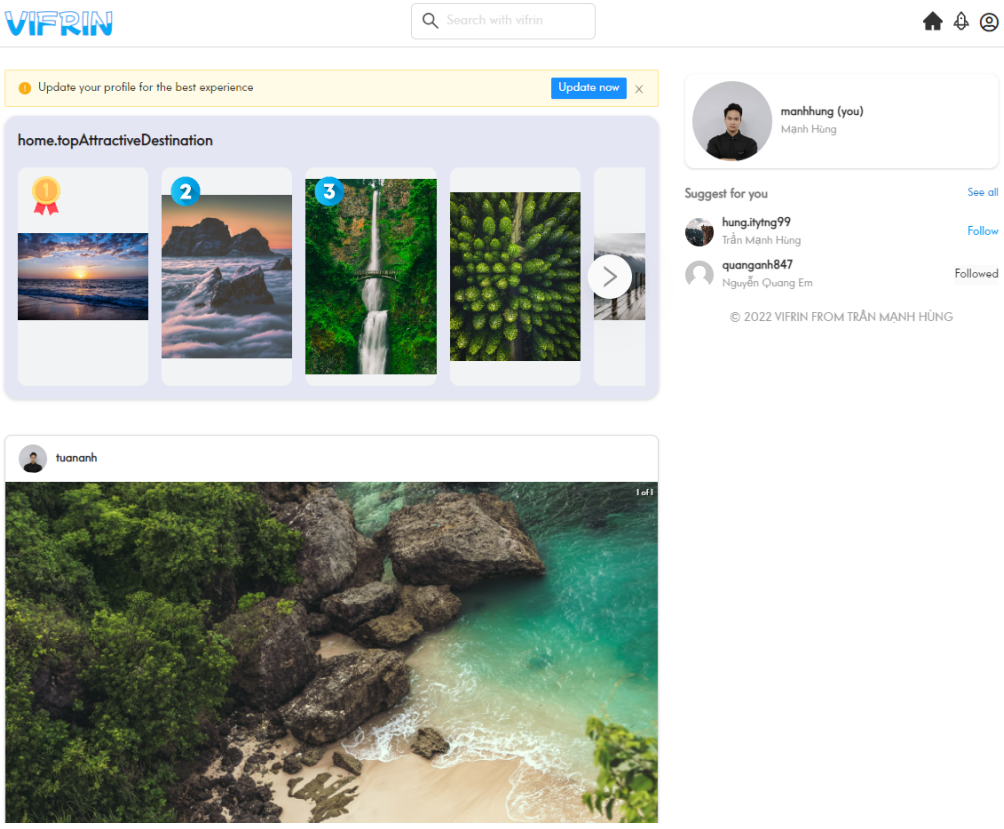

Figure 5.1: Home page

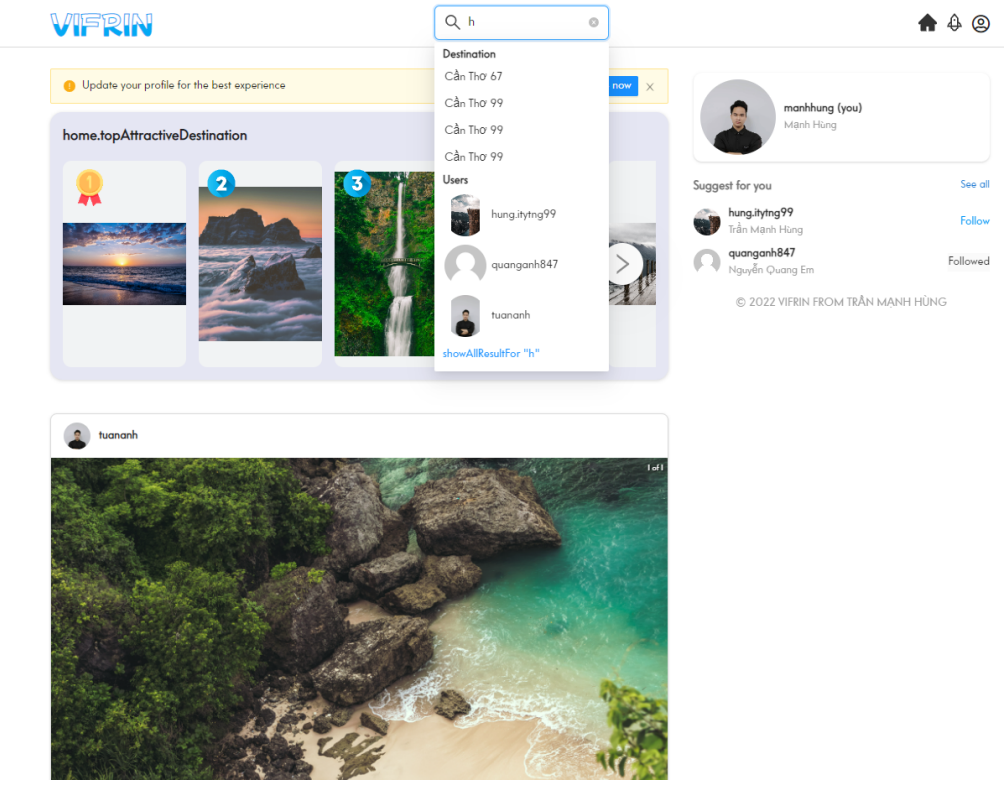

Figure 5.2: Search on header

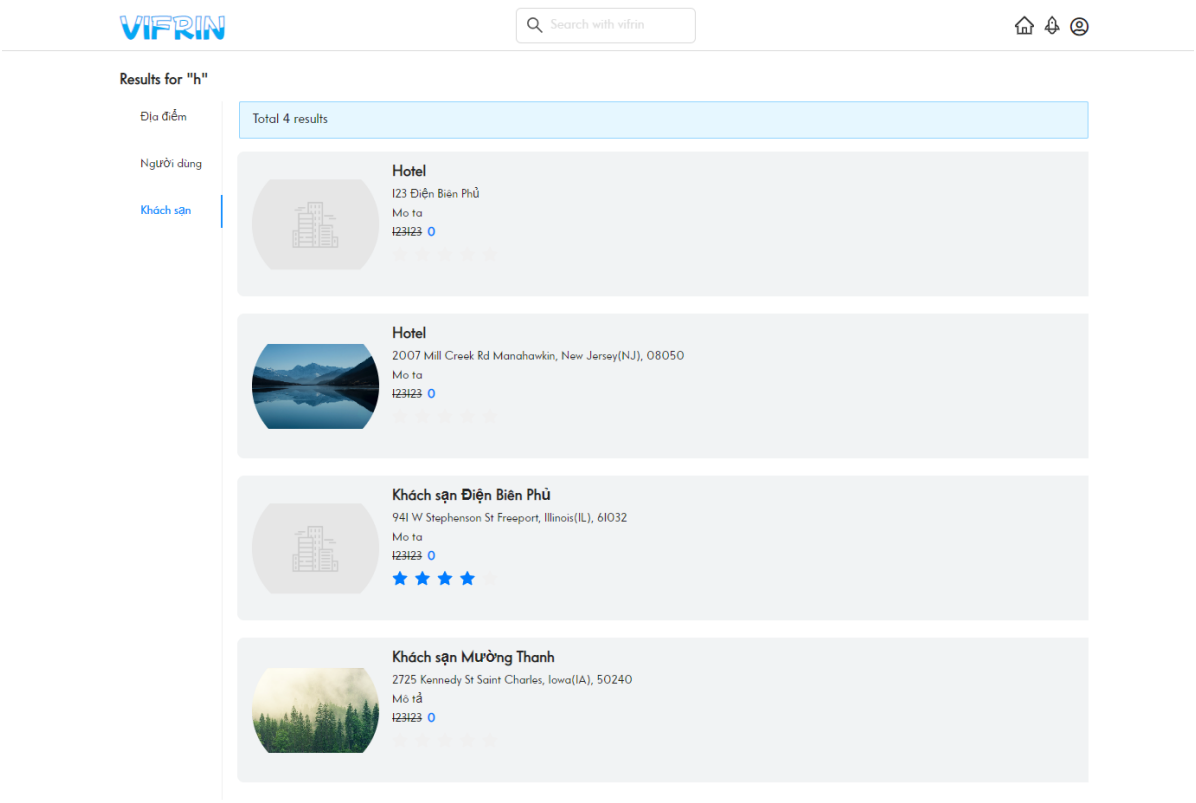

Figure 5.3: All search results page

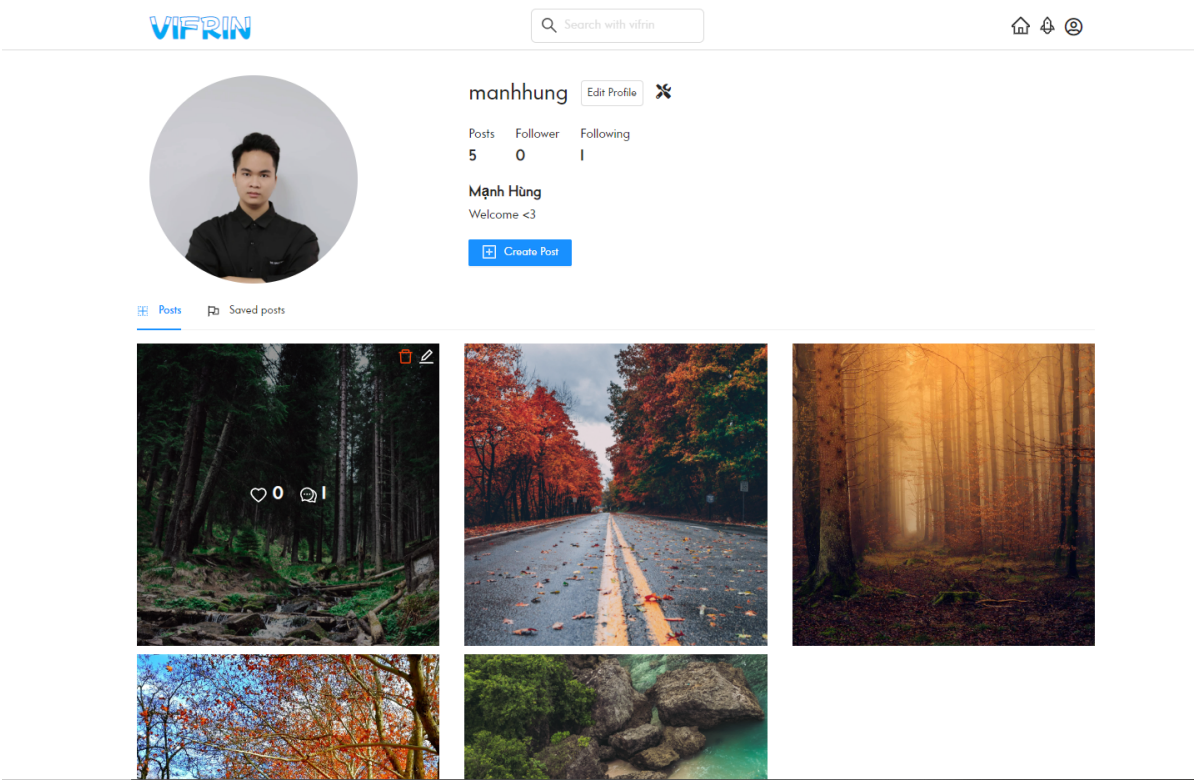

Figure 5.4: Personal profile page

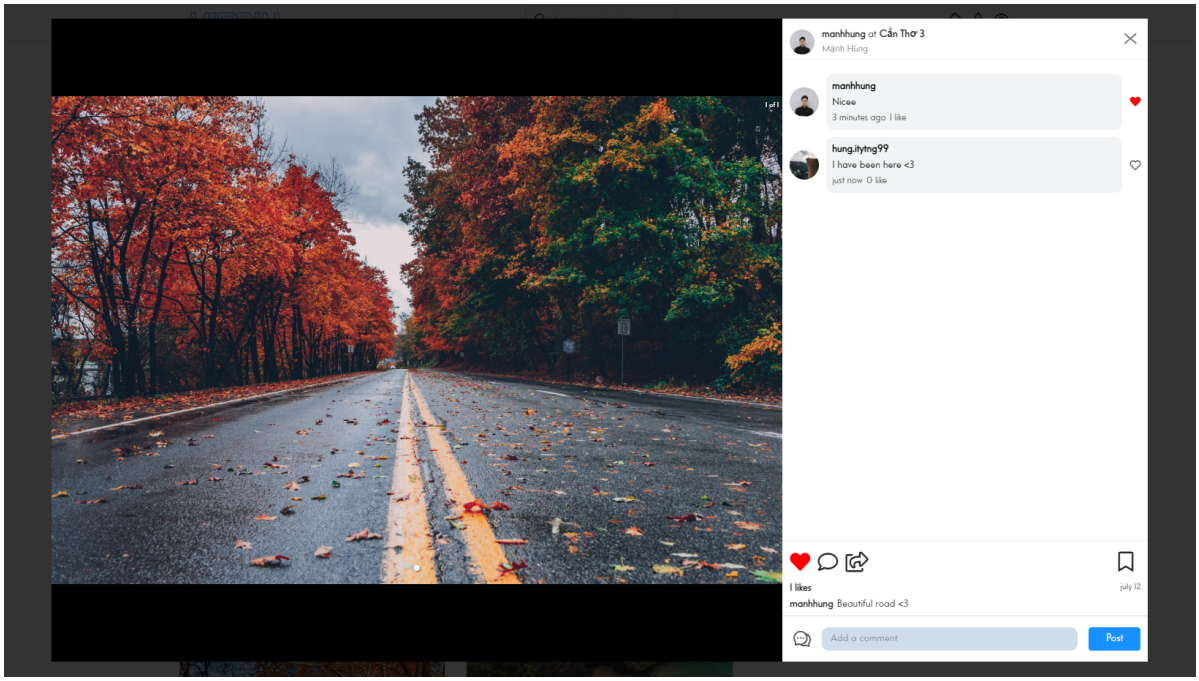

Figure 5.5: Detail post

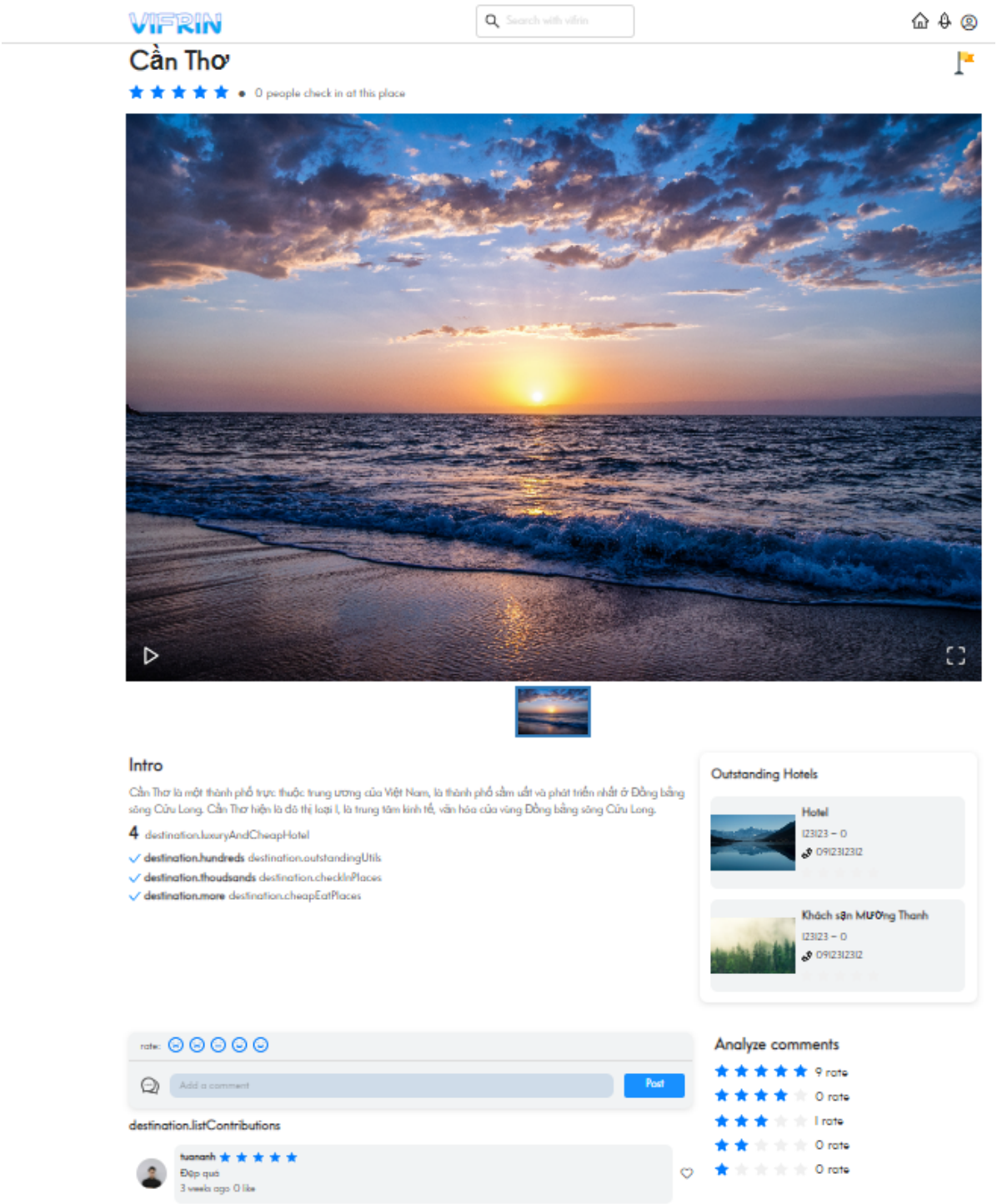

Figure 5.6: Detail destination

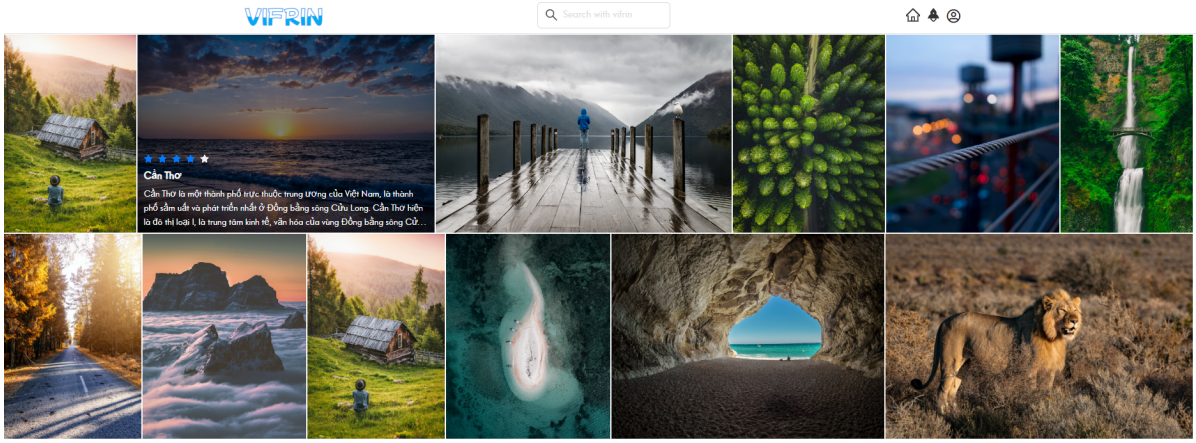

Figure 5.7: Explore destinations page

5.2 Testing

5.2.1 Testing scenarios

The table 5.2 shows some testing scenarios for some of the most important features of the application:

Function	Scenario	Expected Result
Register	1. Input user information (username, password, retype password,...) 2. Click "Register" button	1. If the input is correct, navigate to login page. 2. If the input is incorrect, an error will appear, request the user to try again.
Login	1. Input username and password 2. Click the “Log in” button	1. If the input is correct, the main page is shown to the user. 2. If the input is incorrect, an error message will appear, request the user to try again.
Create post and update feeds	1. Upload image, enter content and select destination 2. Click the “Post” button	1. If the input is correct, notify success and update list post of user. The system will update the feeds of the users, who follow current user. 2. If the input is incorrect,notify error and request re-entry.
Edit post	1. Hover on created post, select edit 2. Edit post, click "Edit button"	1. If the input is correct, notify success and update post. 2. If the input is incorrect, notify error and request re-entry.
Delete post	1. Hover on created post, select delete 2. An popup confirm appear, click "Yes" or "No"	1. If click "Yes", notify success and update list posts of user. If click "No", nothing happen 2. If error, notify error.
Search destinations, users or hotels	1. Click search input in nav bar 2. Enter keyword	1. An dropdown component appear, show recent search results
Change language	1. Click user dropdown on right navbar. 2. Select "Change language" button.	1. Switch website language

Table 5.2: Testing scenarios

5.2.2 Testing results

Here are the results of table 5.3 Test result

Scenario	Chrome Browser	Microsoft Edge
Register	Pass	Pass
Login	Pass	Pass
Create post and update feeds	Pass	Pass
Edit post	Pass	Pass
Delete post	Pass	Pass
Search destinations, users or hotels	Pass	Pass
Change language	Pass	Pass

Table 5.3: Test result

5.3 Deployment

The system will be deployed according to the client-server model. Because the product is being tested, I do not have clear statistics on the number of users, the number of hits, or the server's load capacity. I use the Microsoft Azure server with configs as shown:

Property	Value
Operating System	Linux 20.04
RAM	16GB
CPU core	4 cores
Environment	Docker, maven

Table 5.4: Server deployment configuration

CHAPTER 6. SOLUTION AND CONTRIBUTION

6.1 Build micro-service system with java spring boot

6.1.1 Problem

Monolithic architecture is considered to be a traditional way of building applications. A monolithic application is built as a single and indivisible unit. Usually, such a solution comprises a client-side user interface, a server side-application, and a database. It is unified and all the functions are managed and served in one place. Normally, monolithic applications have one large code base and lack modularity. If developers want to update or change something, they access the same code base. So, they make changes in the whole stack at once. There are some weaknesses of the monolithic architecture:

- **Understanding:** When a monolithic application scales up, it becomes too complicated to understand. Also, a complex system of code within one application is hard to manage.
- **Making changes:** It is harder to implement changes in such a large and complex application with highly tight coupling. Any code change affects the whole system so it has to be thoroughly coordinated. This makes the overall development process much longer.
- **Scalability:** You cannot scale components independently, only the whole application.
- **New technology barriers:** It is extremely problematic to apply new technology in a monolithic application because then the entire application has to be rewritten.

6.1.2 Solution overview

Instead of using monolithic architecture, I use micro-service architecture. A micro-services architecture breaks application down into a collection of smaller independent units. These units carry out every application process as a separate service. So all the services have their logic and database as well as perform specific functions. Within a micro-services architecture, the entire functionality is split up into independently deployable modules which communicate with each other through defined methods called APIs (Application Programming Interfaces). Each service covers its scope and can be updated, deployed, and scaled independently. Strengths of the micro-service architecture:

- **Independent components:** All the services can be deployed and updated

independently, which gives more flexibility. Secondly, a bug in one micro-service has an impact only on a particular service and does not influence the entire application. Also, it is much easier to add new features to a micro-service application than a monolithic one.

- **Easier understanding:** All the services can be deployed and updated independently, which gives more flexibility. Secondly, a bug in one micro-service has an impact only on a particular service and does not influence the entire application. Also, it is much easier to add new features to a micro-service application than a monolithic one.
- **Better scalability:** Another advantage of the micro-services approach is that each element can be scaled independently. So the entire process is more cost- and time-effective than with monoliths when the whole application has to be scaled even if there is no need for it. In addition, every monolith has limits in terms of scalability, so the more users you acquire, the more problems you have with your monolith. Therefore, many companies, end up rebuilding their monolithic architectures.

6.1.3 Results and future development directions

After research and implementation, I have built a micro-service system with the Java Spring Boot framework. Spring Boot makes it very easy to deploy micro-services. Each micro-service system must have a gateway, which is a software application between a client and a set of back-end micro-services. I use cloud-gateway, a package of maven that helps easily to create the configuration of the gateway.

```
- id: POST-SERVICE
  uri: lb://POST-SERVICE
  predicates:
    - Path=/posts/**
  filters:
    - name: CircuitBreaker
      args:
        name: POST-SERVICE
        fallbackuri: forward:/postServiceFallBack
```

Figure 6.1: Post service configuration of gateway

Besides, I use **JpaRepository** to create an interaction with the database very quickly and conveniently. All CRUD operations with the entity have been written, which will save software development time. **JpaRepository** also supports writing quick custom queries. For example, when defining the method of **List<User> find-**

ByEmailAddressAndLastname(String emailAddress, String lastname);. In the repository interface, Spring Boot will automatically create the query, specifically in this example is **Select u From User u Where u.emailAddress = ?1 and u.lastname = ?2**. Here are some figures for **Post Repository**

```
@Repository
public interface PostRepository extends JpaRepository<Post, Long> {
    1 usage  trantuananh1
    List<Post> findByUserIdOrderByCreatedAtDesc(Long userId);

    1 usage  trantuananh1
    @Query(value = "select * from posts p order by p.comments_count DESC LIMIT ?1", nativeQuery = true)
    List<Post> getPostOrderByCommentsCountDesc(int size);


}
```

Figure 6.2: Post repository method

In addition, all services will be monitored by Eureka, which is an application that holds information about all client-service applications. Every micro-service will register into the Eureka server and the Eureka server knows all the client applications running on each port and IP address. We can easily track the status of the service through this application.

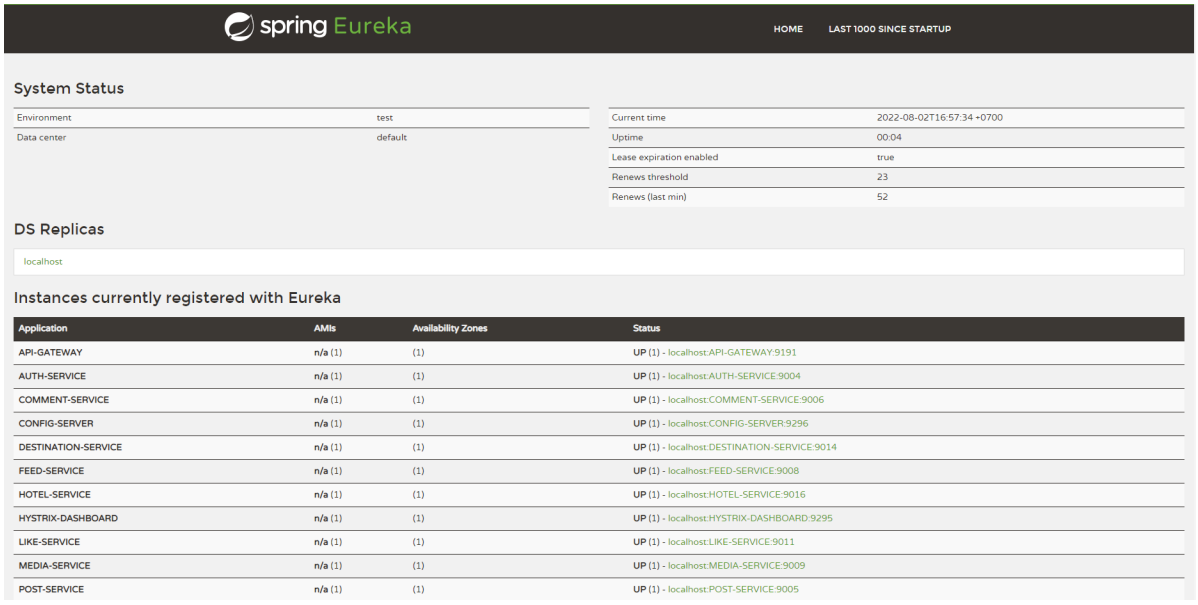

Figure 6.3: Post service configuration of gateway

I have built each service into containers using docker. In the future, this system is very convenient to deploy on Kubernetes, an open-source container orchestration platform of Google. I will add more plugins for tracing requests, logger, and monitor system.

6.2 Develop tools to reduce processing time when retrieving API data on the front-end side

6.2.1 Problem

The front end is an important part of an application, that interacts with the user. CRUD is the basic operation that most applications will be implemented. In vifrin application, there are many operators with posts, comments, destinations, and hotels,...This requires a tool that helps handle some of the lines of code that software developers have to type. This tool can support data processing from API and put it into view to display to the front-end.

6.2.2 Solution overview

Redux-saga is a library that aims to make application side effects (i.e. asynchronous things like data fetching). At this part, we can perform a function to call API using Axios, handle logic business, and dispatch results to the global state. This separates the view and logic, so we can generate code according to a certain CRUD form. Javascript is a language of any type. Therefore, after we get the returned result, we can put its value in a variable called d̈ata. The initial idea is that we will create a web page that allows entering the necessary parameters, depending on the API headers, the parameters submitted, which is the action that will generate the corresponding code. The program will also automatically create the function name, and variable name based on the action name.

6.2.3 Results and future development directions

After developing and deploying, I get a website application deployed on vercel - a platform for front-end frameworks and static sites, built to integrate with headless content, commerce, or database. After entering the necessary information, we get the code from dispatching an action to saving the data to the global state for the view to use. Below is the interface of the website:

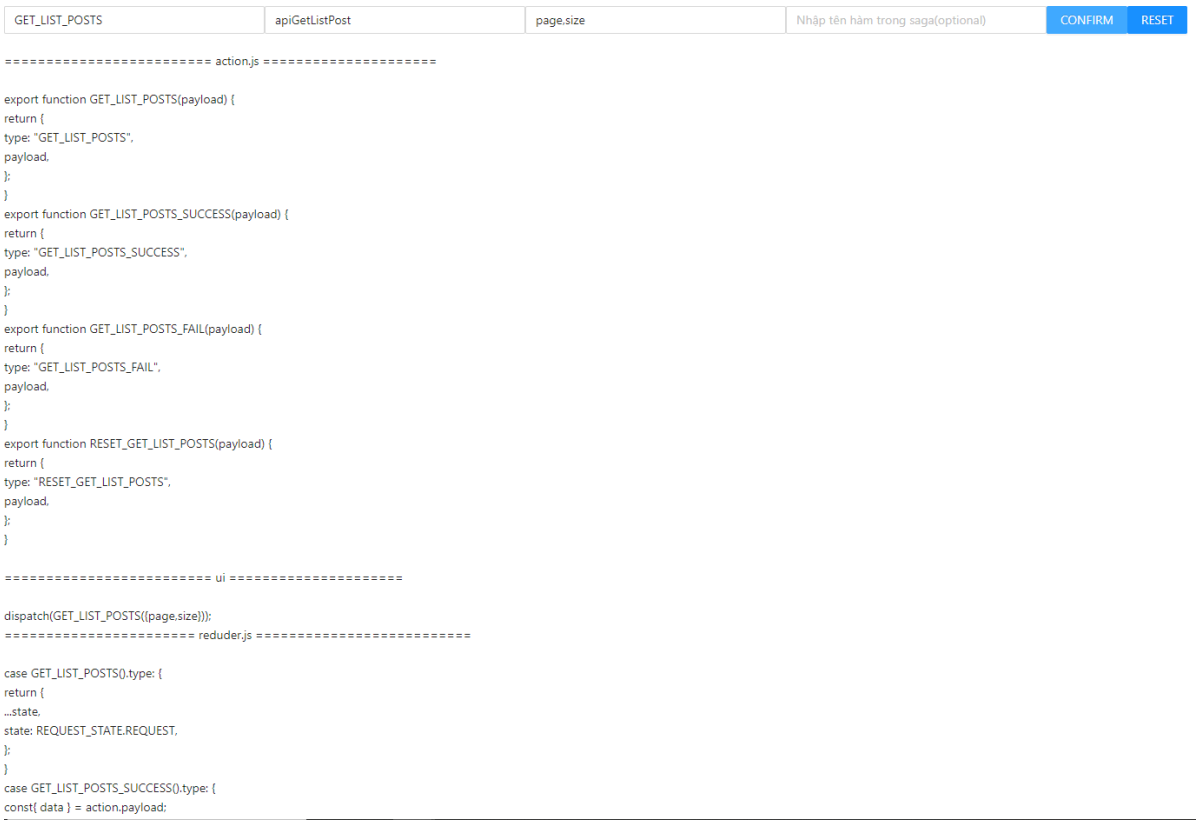

Figure 6.4: Generate redux-saga code

This tool can develop into a scripting language commands file (.sh) or even become an extension of IDE visual studio code.

6.3 Front-end performance and ability of extension

6.3.1 Problem

Performance is an issue that any large application has to deal with. React is a component-based library. As the application expands, the larger the number of components, the heavier the amount of data stored in the global state. Besides, if we are including large third-party libraries. We need to keep an eye on the code we are including in our bundle so that we don't accidentally make it so large that our app takes a long time to load. Hence we need a mechanism to only load the necessary stuff when the user needs it. This helps the client-side limit the processing of unnecessary tasks.

6.3.2 Solution overview

We will apply the code splitting technique to solve this problem. In react, there is a concept called lazy load. React supports a function called lazy() and turns the component into a dynamic component, meaning that react loads the bundle containing the returned component when this component is first rendered. Redux supports replacing existing reducers with reducers that users need. With the redux-

saga, we will gradually map the sagas associated with the pages the user wants to see into the store saga in redux.

6.3.3 Results and future development directions

We will divide the application into screens, each screen will be a module that manages its state and UI. Each of these modules will export config so that the react-router side can connect between the route and the component. That is shown in the following figure:

```
import { profileModule } from "../Profile/route";
export const homeModule = { key: "homepage", path: "Home" };
const homeRoute = {
  index: 1,
  path: "/",
  exact: true,
  isPrivate: true,
  isShowOnNav: true,
  activeIcon: <HomeFilled />,
  inActiveIcon: <HomeOutlined />,
  component: lazy(async () => {
    await initModules([homeModule, profileModule], "app");
    return import(".");
  }),
};
export default homeRoute;
```

Figure 6.5: Export of homepage module

As we can see, each **path** corresponding to a **component**. The component is dynamic and we must wait for it to implement the necessary module before showing the UI to the screen. Hence we can see this module can import other modules to get a saga handler and a reducer handler. This reduces the amount of code to write to handle the business logic when getting data from the API. For example, in the profile screen, we have written logic to get the user's data. If on the homepage we also need this data then we just need to initialize the profileModule to the homepage. Then the homepage will reuse the logic that we wrote in profileModule. Now, we go to the details of the function **initModules**:

```
export const initModules = async (modules = [], container = "app") => {
  await Promise.all([
    modules.map(async (item) => {
      const [reducer, saga] = await Promise.all([
        import(`../screens/${container}/${item.path}/redux/reducer`),
        import(`../screens/${container}/${item.path}/redux/saga`),
      ]);
      store.injectReducer(item.key, reducer.default);
      store.injectSaga(item.key, saga.default);
    }),
  ]);

  // To ensure that modules in injected
  await delay(300);
};
```

Figure 6.6: Detail of initModules function

Based on what is exported from each screen module we can derive the corresponding saga and reducer. Then we can inject it into the redux store.

CHAPTER 7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

Currently, there are many channels online for people to find out if a certain tourist destination is attractive or not. But currently, there is no website about tourism built in the direction of social networks. Most current travel websites tend to be commercial, often suggesting hotels or airline tickets for users to buy. The interface of these websites is often difficult for people to see the information about the places they are interested in.

With the desire to build an application that helps users who love travel, like to experience a new feeling in a strange place, have more perspective on the advantages and disadvantages of places they have never been before quickly and conveniently. I have built a system of social networking sites to share the travel experiences that users have experienced, suggesting places or hotels for them to stop. When building, I tried to apply design technologies, and outstanding system architectures to be able to develop more new features or deploy the system to meet a large number of users in the future. The interface of the website is user-friendly, making it easy to see the post of their favorite destination. Besides, I also develop tools to code programming faster.

By completing this project, I was able to hone myself with valuable knowledge and skills. From the smallest steps such as coming up with ideas, and surveying other similar applications and systems, to the steps of business analysis, deployment and maintenance, and system upgrade, I have gained valuable experience. valuable, useful for building and developing other applications in the future. Besides, the learning and understanding of technologies and techniques to be able to solve the problems and problems posed by me have been applied flexibly, which is reflected in the results of the good project. this profession. This will be an important and extremely valuable suitcase and knowledge so that I can be confident and steady so that I can step out of school and develop further for my future career.

Despite spending time and trying to complete the project to the maximum extent, due to limited capacity and limited time, the problems in the system are still relatively unresolved. Typically, the system's functions are still not many. Deployment to the environment and put into actual use to get reviews from users is not yet available. In addition, there are still some functions and businesses that have been set out from the beginning but have not been implemented in this version. I have grasped these limitations and will overcome them in the future, to improve

the quality and perfection of the application.

7.2 Future work

The functions provided by the system have met the objectives of the project. However, some functions still need to be further improved to provide a better user experience. Besides, to improve the speed of functions under heavy load in web applications, it is necessary to learn and apply load balancing and parallel computing techniques to increase data processing performance. I must optimize code and remove unnecessary code. In addition, I will improve more features such as chat, notifications, or stories so that users have a better experience, and have more choices when using the application. If the application is deployed, when there is enough data, I can develop suggested models like destination suggestions, post suggestions, and user suggestions.

Above is the entire content of the project "Design and develop website for travel social network". With limited time and limited skills and experience, my project implementation process inevitably has shortcomings. I look forward to receiving the guidance and comments of the teachers so that the system can be improved in the future.

REFERENCE

[1] C. Walls, *Spring Boot in action*. Simon and Schuster, 2015.

[2] P. Webb, D. Syer, J. Long, *et al.*, "Spring boot reference guide," *Part IV. Spring Boot features*, vol. 24, 2013.

[3] J. Carlson, *Redis in action*. Simon and Schuster, 2013.

[4] J. D. Drake and J. C. Worsley, *Practical PostgreSQL*. " O'Reilly Media, Inc.", 2002.

[5] G. Deleuze, F. Guattari, and S. Miletić, *Kafka*. Literarno-umetniško društvo Literatura, 1995.

[6] C. Anderson, "Docker [software engineering]," *Ieee Software*, vol. 32, no. 3, pp. 102–c3, 2015.

[7] A. Fedosejev, *React. js essentials*. Packt Publishing Ltd, 2015.

[8] A. Vipul and P. Sonpatki, *ReactJS by Example-Building Modern Web Applications with React*. Packt Publishing Ltd, 2016.

[9] A. Banks and E. Porcello, *Learning React: functional web development with React and Redux*. " O'Reilly Media, Inc.", 2017.

[10] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, IEEE, 2016, pp. 44–51.

[11] N. Dmitry and S.-S. Manfred, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[12] J. Dong, Y. Zhao, and T. Peng, "A review of design pattern mining techniques," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 06, pp. 823–855, 2009.