**MINISTRY OF EDUCATION AND TRAINING**

**HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY FOR HIGH QUALITY TRAINING**

*A logo of hands holding a book and a candle

Description automatically generated*

**GRADUATION PROJECT**

**BUILDING A WEBSITE AND AN APP FOR ACCOMODATION BOOKING**

**Advisor: NGUYỄN THIÊN BẢO, PhD.**

**Major**: **INFORMATION TECHNOLOGY**

**Members:**

1. Khúc Nguyễn Huy Cường - 19110107
2. Nguyễn Đan Trường - 19110064
3. Đặng Ngọc Trường Giang - 19110104

Ho Chi Minh City, July 2023

**SOCIALIST REPUBLIC OF VIETNAM**

**Independence - Freedom – Happiness**

**\*\*\*\*\*\*\***

Ho Chi Minh, 15/07/2023

GRADUATION THESIS TASK

Full name : Khúc Nguyễn Huy Cường Student Id: 19110107 Class: 19110CLA1

Full name: Nguyễn Đan Trường Student Id: 19110064 Class: 19110CLA1

Full name: Đặng Ngọc Trường Giang Student Id: 19110104 Class: 19110CLA1

Major: Information Technology

Instructor: Nguyễn Thiên Bảo, PhD

The date of receiving the topic: 07/03/2023 Thesis submission date: 15/07/2023

1. Project name: Building a website and an app for accomodation booking
2. Original data and documents: None
3. Content implementation of the topic:

- Learn about ReactJS, NextJS, Beego and MongoDB

* Build a website for accomodation booking
* Buid an app for accomodation booking

INDUSTRY LEADER INSTRCUTOR

Nguyễn Trần Thi Văn Nguyễn Thiên Bảo

**SOCIALIST REPUBLIC OF VIETNAM**

**Independence - Freedom – Happiness**

**\*\*\*\*\*\*\***

INSTRUCTOR’S EVALUATION

Full name : Khúc Nguyễn Huy Cường Student Id: 19110107 Class: 19110CLA1

Full name: Nguyễn Đan Trường Student Id: 19110064 Class: 19110CLA1

Full name: Đặng Ngọc Trường Giang Student Id: 19110104 Class: 19110CLA1

Major: Information Technology

Project name: Building a website and an app for accomodation booking

Instructor: Nguyễn Thiên Bảo, PhD

**Evaluation:**

1. Regarding the content of the topic and the volume of implementation:

a) Theory: ReactJS, NextJS, Nodejs, Beego and MongoDB

b) Experiments:

Conduct surveys on popular hotel booking websites such as Agoda, Airbnb. From there determine the requirements for the project, identify key points and main business - Building a website and an app for accomodation booking.

Learn and apply ReactJS, NextJS, Nodejs, Beego and MongoDB to the project.

Build back-end using MongoDB, Beego and front-end using ReactJS, NextJS, Nodejs. Deploy website on the Internet and intergrate with mobile app.

1. Strength:

Project:

* Friendly user interface
* Support online payment for many banks
* Provide 360 image for better visualization before booking
* Provide a wide range of accommodations

Group:

* Good collaboration and brainstorming, fostering the exchange of ideas and perspectives
* Divide the work evenly throughout the duration of the project
* Each team member is always on time for each assigned task

1. Drawback:

Project:

* Accommodation is currently available in Vietnam only
* Loading page speed is quite slow

Group:

* Miscommunication, delays in information sharing, and differences in communication styles
* Some member is on a job so it takes time to communicate with them

1. Approval for oral defense? (Approved or denied)

………………………………………………………………………………….

1. Overall evaluation: (Excellent, Good, Fair, Poor)

………………………………………………………………………………….

1. Mark:………………(By word:……………………………..……..............)

*Ho Chi Minh, 15/07/2023*

Lecture

*(Sign, write full name)*

**SOCIALIST REPUBLIC OF VIETNAM**

**Independence - Freedom – Happiness**

**\*\*\*\*\*\*\***

# REVIEWER’S EVALUATION

Full name : Khúc Nguyễn Huy Cường Student Id: 19110107 Class: 19110CLA1

Full name: Nguyễn Đan Trường Student Id: 19110064 Class: 19110CLA1

Full name: Đặng Ngọc Trường Giang Student Id: 19110104 Class: 19110CLA1

Major: Information Technology

Project name: Building a website and an app for accomodation booking

Reviewer teacher: Nguyễn Trần Thi Văn, MSc

**Evaluation:**

1. Regarding the content of the topic and the volume of implementation:

………………………………………………………………………………….

………………………………………………………………………………….

………………………………………………………………………………….

………………………………………………………………………………….

1. Strength:

………………………………………………………………………………….

………………………………………………………………………………….

………………………………………………………………………………….

………………………………………………………………………………….

1. Drawback:

………………………………………………………………………………….

………………………………………………………………………………….

………………………………………………………………………………….

………………………………………………………………………………….

1. Approval for oral defense? (Approved or denied)

………………………………………………………………………………………...

1. Overall evaluation: (Excellent, Good, Fair, Poor)

………………………………………………………………………………………...

1. Mark:………………(By word:……………………………..……........................)

*Ho Chi Minh, 15/07/2023*

Lecture

*(Sign, write full name)*

ACKNOWLEDGEMENT

Mr. Nguyen Thien Bao, the instructor of our course, deserves special thanks from our team. Throughout this course, you have constantly assisted us with our difficulties in the study process as well as provided us with excellent feedback and recommendations to help us grow as developers in the future. As a result, you have become our inspiration for this final project, and we would like to express our gratitude for the opportunity to learn from you in this course.

Everyone on our team has various talents, and we have discovered our shortcomings in each other over this final assignment, but we've all managed to face this challenge as a team. As a result, everyone on our team is really grateful for the opportunity to collaborate.

Thank you to the teachers in the Faculty for High Quality Training for always being enthusiastic, dedicated, and dedicated to answering our questions. Besides, we would like to thank our classmates and seniors who did not hesitate to share useful information and experiences to help us improve our thesis.

This final project is the result of our tireless efforts throughout the semester. Because we are still novices, we are aware that our design has numerous shortcomings. We welcome any feedback as well as suggestions for improving our project. We appreciate it. Sincerely!

Ho Chi Minh city, 15/07/2023

Khúc Nguyễn Huy Cường

Nguyễn Đan Trường

Đặng Ngọc Trường Giang

Table Of Content

[CHAPTER 1: OVERVIEW 1](#_Toc139884534)

[1.1 The urgency of the subject 1](#_Toc139884535)

[1.2. Purpose of project 2](#_Toc139884536)

[1.3. Object and scope of the study 2](#_Toc139884537)

[1.4. Expected results achieved 2](#_Toc139884538)

[CHAPTER 2. STATUS SURVEY AND DETERMINATION OF REQUIREMENTS 3](#_Toc139884539)

[2.1 Current status survey 3](#_Toc139884540)

[2.2 Some sample websites 4](#_Toc139884541)

[CHAPTER 3. SYSTEM ARCHITECTURE AND USED TECHNOLOGY 7](#_Toc139884542)

[3.1 Overall architecture of the system 7](#_Toc139884543)

[3.1.1 System architecture 7](#_Toc139884544)

[3.1.2. File structure 8](#_Toc139884545)

[3.1.3. Package 10](#_Toc139884546)

[3.2 ReactJS 13](#_Toc139884547)

[3.2.1 Definition 13](#_Toc139884548)

[3.2.2 Uses 13](#_Toc139884549)

[3.2.3 Strengths 14](#_Toc139884550)

[3.2.4 Drawbacks 14](#_Toc139884551)

[3.3 NextJS 15](#_Toc139884552)

[3.3.1 Definition 15](#_Toc139884553)

[3.3.2 Uses 15](#_Toc139884554)

[3.3.3 Strengths 16](#_Toc139884555)

[3.3.4 Drawbacks 16](#_Toc139884556)

[3.4 Beego 17](#_Toc139884557)

[3.4.1 Definition 17](#_Toc139884558)

[3.4.2 Uses 17](#_Toc139884559)

[3.4.3 Strengths 17](#_Toc139884560)

[3.4.4 Drawbacks 18](#_Toc139884561)

[3.5 MongoDB 18](#_Toc139884562)

[3.5.1 Definition 18](#_Toc139884563)

[3.5.2 Uses 19](#_Toc139884564)

[3.5.3 Strengths 19](#_Toc139884565)

[3.5.4 Drawbacks 20](#_Toc139884566)

[CHAPTER 4: MODELING REQUIREMENT 21](#_Toc139884567)

[4.1 User requirement 21](#_Toc139884568)

[4.1.1 Functional requirement 21](#_Toc139884569)

[4.2 Identify actors and use cases 26](#_Toc139884570)

[4.3 Detailed description of each usecase and each actor 28](#_Toc139884571)

[4.3.1 Actor description 28](#_Toc139884572)

[4.3.2 Use case description 29](#_Toc139884573)

[4.4 Use case diagram 31](#_Toc139884574)

[4.5 Sequence diagram 35](#_Toc139884575)

[4.6 Use case specification 46](#_Toc139884576)

[CHAPTER 5: DATABASE DESIGN 53](#_Toc139884577)

[5.1 Attribute table 53](#_Toc139884578)

[5.2 Entity relationship diagram 64](#_Toc139884579)

[CHAPTER 6: USER INTERFACE DESIGN 65](#_Toc139884580)

[6.1 Admin site 65](#_Toc139884581)

[6.2 Client site 77](#_Toc139884582)

[CHAPTER 7: SOFTWARE TESTING 83](#_Toc139884583)

[7.1 Test Plan 83](#_Toc139884584)

[7.1.1 Test Strategy 83](#_Toc139884585)

[7.1.2 Execution Strategy 85](#_Toc139884586)

[7.1.3 Test Environment 88](#_Toc139884587)

[7.2 Test Process 89](#_Toc139884588)

[7.2.1 Unit Test 89](#_Toc139884589)

[7.2.2 Integration Test 89](#_Toc139884590)

[7.2.3 System Test 90](#_Toc139884591)

[7.2.4 Acceptance Test 91](#_Toc139884592)

[7.2.5 White Box Testing 92](#_Toc139884593)

[7.2.6 Black Box Testing 92](#_Toc139884594)

[CHAPTER 8. CONCLUSION 94](#_Toc139884595)

[8.1 Achievements 94](#_Toc139884596)

[8.1.1 Knowledge 94](#_Toc139884597)

[8.1.2 Skills 94](#_Toc139884598)

[8.1.3 Product 94](#_Toc139884599)

[8.2 Strengths and Drawbacks 95](#_Toc139884600)

[8.2.1 Strengths 95](#_Toc139884601)

[8.2.2 Drawbacks 95](#_Toc139884602)

[8.3 Future Work 95](#_Toc139884603)

[REFERENCES 96](#_Toc139884604)

[WORKLOAD TABLE 98](#_Toc139884605)

List Of Figures

[Figure 1: Agoda website 5](#_Toc139822191)

[Figure 2: Airbnb website 6](#_Toc139822192)

[Figure 3: System architecture 7](#_Toc139822193)

[Figure 4: File structure for admin front-end 8](#_Toc139822194)

[Figure 5: File structure for client front-end 9](#_Toc139822195)

[Figure 6: File structure of back-end 10](#_Toc139822196)

[Figure 7: General use case diagram 32](#_Toc139822197)

[Figure 8: Admin use case diagram 33](#_Toc139822198)

[Figure 9: Host use case diagram 34](#_Toc139822199)

[Figure 10: Customer use case diagram 35](#_Toc139822200)

[Figure 11: Log in sequence diagram 36](#_Toc139822201)

[Figure 12: Register sequence diagram 36](#_Toc139822202)

[Figure 13: Search hotel sequence diagram 37](#_Toc139822203)

[Figure 14: Create hotel sequence diagram 38](#_Toc139822204)

[Figure 15: Create order sequence diagram 39](#_Toc139822205)

[Figure 16: Book hotel sequence diagram 40](#_Toc139822206)

[Figure 17: Create news sequence diagram 41](#_Toc139822207)

[Figure 18: Create room sequence diagram 42](#_Toc139822208)

[Figure 19: Add promotion sequence diagram 43](#_Toc139822209)

[Figure 20: Manage comment sequence diagram 44](#_Toc139822210)

[Figure 21: Manage amenity sequence diagram 45](#_Toc139822211)

[Figure 22: Export statistic sequence diagram 46](#_Toc139822212)

[Figure 23: Entity relationship diagram 64](#_Toc139822213)

[Figure 24: Login page 65](#_Toc139822214)

[Figure 25: Log in page event diagram 65](#_Toc139822215)

[Figure 26: Admin page 66](#_Toc139822216)

[Figure 27: Admin page event diagram 67](#_Toc139822217)

[Figure 28: Hotel management page 68](#_Toc139822218)

[Figure 29: Hotel management page event diagram 69](#_Toc139822219)

[Figure 30: Villa management page 69](#_Toc139822220)

[Figure 31: Villa management page event diagram 70](#_Toc139822221)

[Figure 32: Townhouse management page 70](#_Toc139822222)

[Figure 33: Townhouse management page event diagram 71](#_Toc139822223)

[Figure 34: News page 71](#_Toc139822224)

[Figure 35: News page event diagram 72](#_Toc139822225)

[Figure 36: Account management page 72](#_Toc139822226)

[Figure 37: Account management page event diagram 73](#_Toc139822227)

[Figure 38: Hotel bill page 74](#_Toc139822228)

[Figure 39: Hotel bill page event diagram 75](#_Toc139822229)

[Figure 40: Sign up page 75](#_Toc139822230)

[Figure 41: Sign up page event diagram 76](#_Toc139822231)

[Figure 42: Booking page 77](#_Toc139822232)

[Figure 43: Booking page event diagram 78](#_Toc139822233)

[Figure 44: Payment page 79](#_Toc139822234)

[Figure 45: Payment page event diagram 80](#_Toc139822235)

[Figure 46: Defect tracking process 86](#_Toc139822236)

List Of Tables

[Table 1: Advantages and disavantages of Agoda website 5](#_Toc139822285)

[Table 2: Advantages and disavantages of Airbnb website 6](#_Toc139822286)

[Table 3: Back-end packages 12](#_Toc139822287)

[Table 4: Front-end packages 13](#_Toc139822288)

[Table 5: Function business requirement 24](#_Toc139822289)

[Table 6: Function system requirement 25](#_Toc139822290)

[Table 7: Non-functional requirement 26](#_Toc139822291)

[Table 8: Actors and use cases 27](#_Toc139822292)

[Table 9: Actor description 29](#_Toc139822293)

[Table 10: Use case description 31](#_Toc139822294)

[Table 11: Use case Log in specification 47](#_Toc139822295)

[Table 12: Use case Register specification 47](#_Toc139822296)

[Table 13: Use case View hotel specification 48](#_Toc139822297)

[Table 14: Use case View news specification 48](#_Toc139822298)

[Table 15: Use case Search hotel specification 48](#_Toc139822299)

[Table 16: Use case Manage account specification 49](#_Toc139822300)

[Table 17: Use case Manage hotel specification 49](#_Toc139822301)

[Table 18: Use case Manage room specification 50](#_Toc139822302)

[Table 19: Use case Manage comment specification 50](#_Toc139822303)

[Table 20: Use case Manage order specification 51](#_Toc139822304)

[Table 21: Use case Manage news specification 51](#_Toc139822305)

[Table 22: Use case Manage statistic specification 52](#_Toc139822306)

[Table 23: Use case Manage price specification 52](#_Toc139822307)

[Table 24: User model 53](#_Toc139822308)

[Table 25: Account model 53](#_Toc139822309)

[Table 26: Amenity 54](#_Toc139822310)

[Table 27: Comment model 54](#_Toc139822311)

[Table 28: Contact model 55](#_Toc139822312)

[Table 29: Holiday model 55](#_Toc139822313)

[Table 30: Hotel model 57](#_Toc139822314)

[Table 31: MonthlyFee model 57](#_Toc139822315)

[Table 32: News model 57](#_Toc139822316)

[Table 33: News content model 58](#_Toc139822317)

[Table 34: Room model 58](#_Toc139822318)

[Table 35: RoomOrder model 61](#_Toc139822319)

[Table 36: VillaTownhouse model 62](#_Toc139822320)

[Table 37: VillaTownhouseOrder model 64](#_Toc139822321)

[Table 38: Log in page component 65](#_Toc139822322)

[Table 39: Admin page component 67](#_Toc139822323)

[Table 40: Hotel management page component 68](#_Toc139822324)

[Table 41: Villa management page component 70](#_Toc139822325)

[Table 42: Townhouse management page component 71](#_Toc139822326)

[Table 43: News page component 72](#_Toc139822327)

[Table 44: Account management page component 73](#_Toc139822328)

[Table 45: Hotel bill page component 74](#_Toc139822329)

[Table 46: Sign up page component 76](#_Toc139822330)

[Table 47: Booking page component 78](#_Toc139822331)

[Table 48: Payment page component 79](#_Toc139822332)

[Table 49: Type of testing 84](#_Toc139822333)

[Table 50: Error classification 86](#_Toc139822334)

[Table 51: Workload table 95](#_Toc139822335)

# CHAPTER 1: OVERVIEW

## 1.1 The urgency of the subject

With the advent of technology and the internet, the traditional method of booking accommodation has undergone a substantial transformation. Online accommodation booking platforms have emerged as powerful tools that revolutionize the way people find and reserve lodging options for their travel needs.

Online platforms provide travelers with access to an extensive range of accommodation choices, ranging from hotels, resorts, guesthouses, vacation rentals, and more. These platforms often aggregate options from multiple providers, giving users a comprehensive selection tailored to their preferences and budget. The ability to compare various accommodations empowers travelers to make informed decisions based on their specific needs.

Online booking platforms offer real-time availability information. Travelers can instantly check if a particular accommodation is available on their desired dates, ensuring transparency and avoiding potential disappointment due to overbooking or unavailability. This feature enables users to plan their trips with greater accuracy and flexibility.

It also allow travelers to access authentic user reviews and ratings. This valuable feedback from previous guests helps prospective travelers gain insights into the quality, cleanliness, service, and overall experience of a particular accommodation. This transparency promotes trust and helps users make informed decisions based on the experiences of others.

In summary, the urgency of online accommodation booking lies in its ability to simplify the booking process, provide convenience, offer extensive choices, save costs, and enhance the overall travel experience. With the rapid advancements in technology and increasing reliance on digital solutions, online accommodation booking has become an essential tool for travelers worldwide, catering to their needs efficiently and effectively.

From idea generation to status survey, requirements definition, system design analysis, interface design, etc., to program installation, testing and completion, this article will present Full description of the project implementation process.

## 1.2. Purpose of project

* Create a website that is convenient for users.
* A website with all basic and stand-out functions of an accommodation booking website.
* Ensure stability and security for users.

## 1.3. Object and scope of the study

The research was conducted around two focus groups of subjects including: subjects with technological knowledge and little knowledge of technology.

In which, the group of technology knowledge subjects includes 4 compulsory research subjects: ReactJS Library, NextJS library MongoDB database and Beego framework. Incorporating more in the group of research subjects on technology knowledge are a number of valuable Javascript libraries

The scope of the research is set out at a general level, the researcher understands the general knowledge of the research content and can apply each knowledge content to the actual product, not placing heavy emphasis on theories.

## 1.4. Expected results achieved

Build an accommodation booking website with full functions and most convenient for users

# 

# CHAPTER 2. STATUS SURVEY AND DETERMINATION OF REQUIREMENTS

## 2.1 Current status survey

The survey examines key aspects such as user experience, features, mobile accessibility, customer reviews, and industry trends.

**User experience:**

* The user experience of hotel booking websites has improved over the years, with a focus on intuitive interfaces, streamlined search functionality, and easy navigation.
* Many platforms now offer advanced filtering options, allowing users to specify preferences such as price range, location, amenities, and accommodation type.
* Interactive maps, high-quality images, and detailed descriptions help users make informed decisions.

**Features:**

* Hotel booking websites typically provide features like real-time availability, instant booking confirmation, and secure payment gateways.
* Some platforms offer additional features such as package deals, flight bookings, car rentals, and vacation rentals, providing a comprehensive travel booking experience.
* Loyalty programs and membership benefits are becoming more prevalent, rewarding frequent users with exclusive discounts and perks.

**Mobile Accessibility:**

* Hotel booking websites have prioritized mobile accessibility to cater to the growing number of users who book accommodations using smartphones and tablets.
* Many platforms have developed dedicated mobile apps or responsive websites, ensuring a seamless booking experience across various devices.
* Mobile features often include location-based searches, push notifications, and mobile-exclusive discounts.

**Customer Reviews and Ratings:**

* User-generated reviews and ratings play a crucial role in the decision-making process for potential customers.
* Hotel booking websites typically provide aggregated customer reviews and ratings for each listed property, helping users assess the quality and suitability of accommodations.
* Traveler feedback serves as a valuable resource for future travelers, fostering transparency and accountability within the industry.

**Industry Trends:**

* Integration of artificial intelligence (AI) and machine learning technologies is increasingly prominent, providing personalized recommendations based on user preferences and behavior.
* Direct partnerships between hotels and booking platforms are growing, allowing for exclusive rates and inventory, bypassing third-party distributors.
* Sustainable and eco-friendly accommodations are gaining traction, with booking platforms highlighting environmentally conscious options.

## 2.2 Some sample websites

<https://www.agoda.com/>

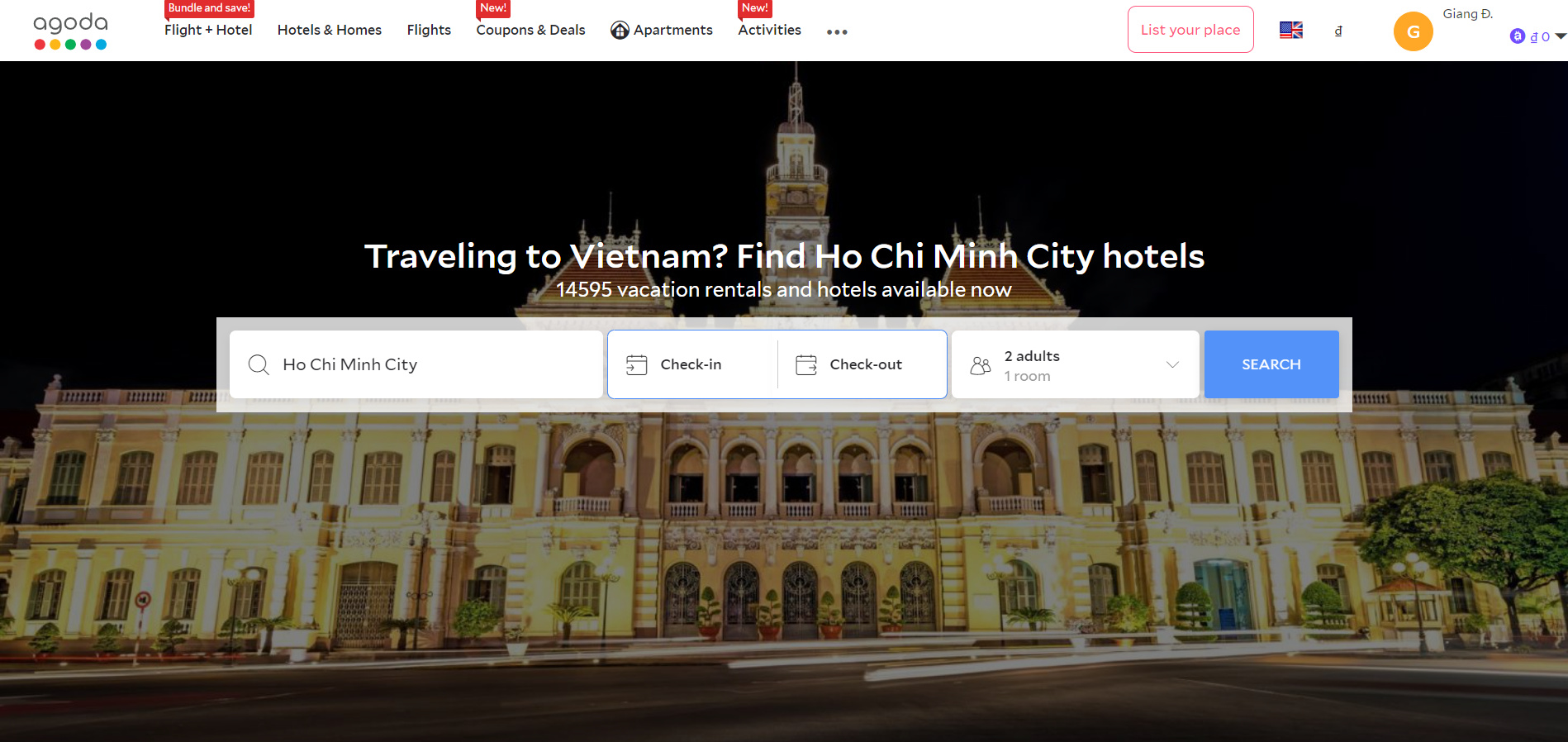


Figure 1: Agoda website

|  |  |
| --- | --- |
| Advantages | * Full information display * Wide range of filter * Multi languages applied * Multi currency applied * Allow multinational booking |
| Disavantages | * Too much information is displayed. This may lead to confusion for the user |

Table 1: Advantages and disavantages of Agoda website

<https://www.airbnb.com/>

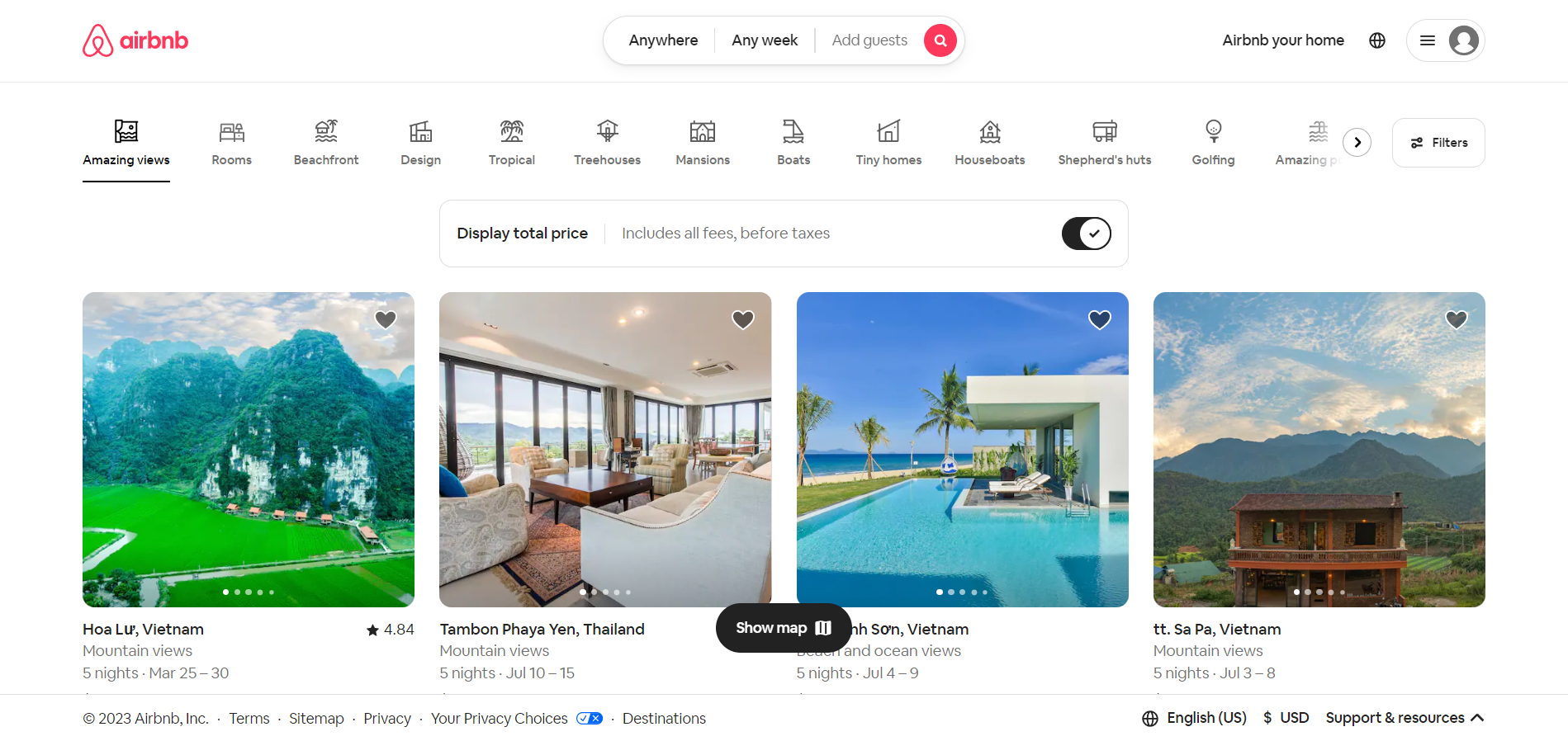


Figure 2: Airbnb website

|  |  |
| --- | --- |
| Advantages | * Information displayed is carefully filtered * Wide range of filter * Multi languages applied * Multi currency applied * Allow multinational booking * Reliable rating and review system * Allow host to post their own hotels, villas, townhouses … |
| Disavantages | * The user interface is quite simple |

Table 2: Advantages and disavantages of Airbnb website

# CHAPTER 3. SYSTEM ARCHITECTURE AND USED TECHNOLOGY

## 3.1 Overall architecture of the system

### 3.1.1 System architecture

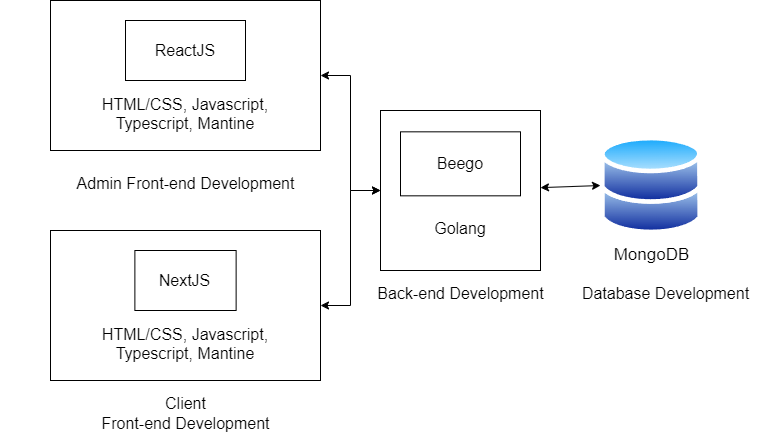


Figure 3: System architecture

### 3.1.2. File structure

#### 3.1.2.1. FrontEnd

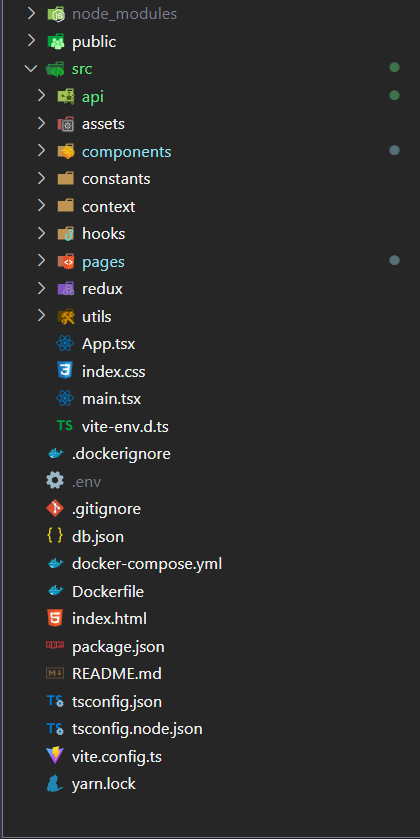


Figure 4: File structure for admin front-end

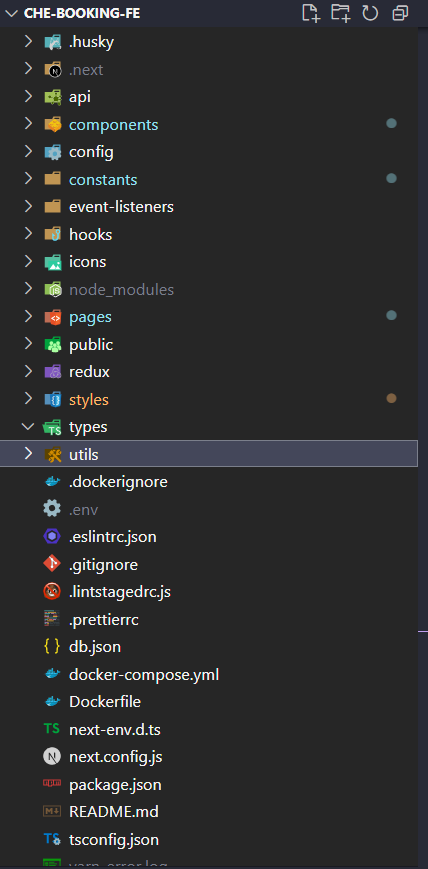


Figure 5: File structure for client front-end

#### 3.1.2.2. BackEnd

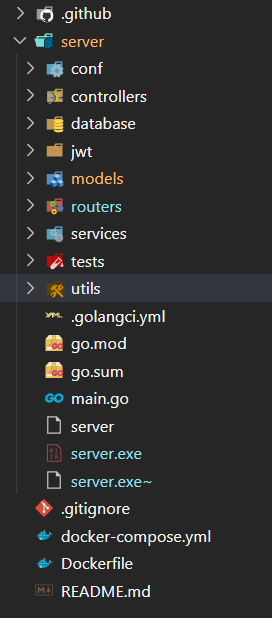


Figure 6: File structure of back-end

### 3.1.3. Package

#### 3.1.3.1. Back-end

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Package** | **Version** | **Uses** |
| 1 | godotenv | 1.4.0 |  |
| 2 | goconvey | 1.6.4 |  |
| 3 | Mongo Driver | 1.10.2 |  |
| 4 | Crypto | 0.3.0 |  |
| 5 | Oauth2 | 0.0.0 |  |
| 6 | Compute | 1.12.1 |  |
| 7 | Pretty | 0.3.0 |  |
| 8 | check | 1.0.0 |  |
| 9 | yaml | 3.0.1 |  |
| 10 | Metadata | 0.2.1 |  |
| 11 | Perks | 1.0.1 |  |
| 12 | xxhash | 2.2.0 |  |
| 13 | Protobuf | 1.5.2 |  |
| 14 | Snappy | 0.0.1 |  |
| 15 | Gopherjs | 0.0.0 |  |
| 16 | Golang lru | 0.5.4 |  |
| 17 | gls | 4.20.0 |  |
| 18 | Compress | 1.13.6 |  |
| 19 | Mapstructure | 1.5.0 |  |
| 20 | Stats | 0.0.0 |  |
| 21 | Errors | 0.9.1 |  |
| 22 | Client Golang | 1.14.0 |  |
| 23 | Client Model | 0.3.0 |  |
| 24 | Common | 0.37.0 |  |
| 25 | Procfs | 0.8.0 |  |
| 26 | Ansicolor | 0.0.0 |  |
| 27 | Assertions | 0.0.0 |  |
| 28 | Testify | 1.8.1 |  |
| 29 | pbkdf2 | 1.0.0 |  |
| 30 | Scram | 1.1.1 |  |
| 31 | Stringprep | 1.0.3 |  |
| 32 | pkcs8 | 0.0.0 |  |
| 33 | net | 0.3.0 |  |
| 34 | sync | 0.1.0 |  |
| 35 | sys | 0.3.0 |  |
| 36 | text | 0.5.0 |  |
| 37 | appengine | 1.6.7 |  |

Table 3: Back-end packages

#### 3.1.3.2. Front-end

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Package** | **Version** | **Uses** |
| 1 | React Redux | 8.0.2 |  |
| 2 | React Image Gallery | 1.2.11 |  |
| 3 | React Flag Kit | 1.1.1 |  |
| 4 | React Dom | 18.2.0 |  |
| 5 | Query string | 8.1.0 |  |
| 6 | Pannellum React | 1.2.4 |  |
| 7 | Next Transpile Modules | 10.0.0 |  |
| 8 | Next Redux Wrapper | 8.0.0 |  |
| 9 | Moment | 2.29.4 |  |
| 10 | Embla carousel react | 7.0.3 |  |
| 11 | Embla carousel autoplay | 7.1.0 |  |
| 12 | Dayjs | 1.11.5 |  |
| 13 | CSS Loader | 6.8.1 |  |
| 14 | Crypto JS | 4.1.1 |  |
| 15 | Axios | 1.2.2 |  |

Table 4: Front-end packages

## 3.2 ReactJS

### 3.2.1 Definition

ReactJS, also known as React, is an open-source JavaScript library maintained by Facebook. It is a component-based framework used for building user interfaces (UI) in web applications. React focuses on creating reusable UI components that efficiently update and render as data changes, providing a fast and responsive user experience.

### 3.2.2 Uses

**Single-Page Applications (SPA):** React is commonly used for building SPAs where the entire application runs within a single web page, enhancing performance and providing a seamless user experience.

**Component Reusability:** React promotes the development of reusable UI components, allowing developers to build complex UI structures by combining smaller, self-contained components. This reusability improves development efficiency and code maintainability.

**Interactive User Interfaces:** React enables the creation of interactive and dynamic user interfaces through its virtual DOM (Document Object Model) and efficient rendering methods. This makes it suitable for applications that require frequent UI updates, such as real-time dashboards or collaborative tools.

### 3.2.3 Strengths

**Virtual DOM:** React's virtual DOM enables efficient updates by minimizing direct manipulations to the actual browser DOM. This results in improved performance and responsiveness.

**Component-Based Architecture:** React's component-based approach promotes code reusability, modularity, and maintainability. Components can be easily composed and nested, making it easier to manage complex UI structures.

**One-Way Data Flow:** React follows a unidirectional data flow, making it easier to understand and debug the application's state changes. This enhances predictability and helps prevent bugs caused by unexpected state mutations.

**Large and Active Community:** React has a vast and active community, providing extensive documentation, tutorials, and third-party libraries. This community support ensures access to resources, knowledge-sharing, and continuous updates.

### 3.2.4 Drawbacks

**Learning Curve:** React has a learning curve, especially for developers who are new to component-based frameworks or JavaScript libraries. Understanding concepts such as JSX syntax and virtual DOM may require some initial investment of time and effort.

Configuration Overhead: React is a library, not a full-fledged framework. To build a complete application, developers may need to configure additional tools, libraries, and build systems. This can increase the setup and configuration overhead.

**Boilerplate Code:** React alone provides the view layer of an application, so developers often need to integrate it with other libraries or frameworks to handle other aspects like routing or state management. This can lead to some boilerplate code and complexity.

Performance Impact: While React's virtual DOM offers performance benefits, it may not be as efficient as directly manipulating the browser DOM in certain scenarios. However, this impact is usually negligible for most applications unless dealing with extremely complex UI structures.

It's important to note that while React has its strengths and drawbacks, its popularity and adoption by major companies demonstrate its value and effectiveness in building modern web applications.

## 3.3 NextJS

### 3.3.1 Definition

Next.js is a JavaScript framework that extends React and provides additional features to simplify the development of server-rendered React applications. It allows developers to build modern web applications with server-side rendering, static site generation, client-side rendering, and API routes, among other capabilities.

### 3.3.2 Uses

**Server-Side Rendering (SSR):** Next.js excels at server-side rendering, which means the initial rendering of pages happens on the server before sending them to the client. This approach improves the page load time and facilitates better SEO performance.

**Static Site Generation (SSG):** Next.js supports static site generation, enabling developers to pre-render static HTML pages at build time. This approach is ideal for content-driven websites, blogs, and landing pages, as it allows for improved performance and reduced server load.

**Single-Page Applications (SPAs):** Next.js can be used to build client-side rendered SPAs, where the initial page is rendered on the client-side and subsequent interactions are handled by React.

**API Routes:** Next.js provides a simple way to create serverless API routes within your application, enabling easy integration with external services or building custom backend functionality.

### 3.3.3 Strengths

**Performance:** Next.js optimizes website performance through features like server-side rendering and static site generation, resulting in faster page loads and better user experience.

**SEO-Friendly:** Server-side rendering and static site generation improve search engine optimization (SEO) by providing pre-rendered HTML content that search engines can easily crawl and index.

**Developer Experience:** Next.js offers a great development experience with features like automatic code splitting, hot module replacement, and built-in TypeScript support. It provides a smooth development workflow, allowing developers to focus on building applications rather than configuring complex setups.

**Ecosystem and Community:** Next.js benefits from a large and active community, which contributes to its extensive ecosystem of plugins, libraries, and resources. This support makes it easier to find solutions, learn from others, and leverage existing tools.

### 3.3.4 Drawbacks

**Learning Curve:** While Next.js is built on React, it has additional concepts and features to learn, especially if you're new to server-side rendering or static site generation. It may require some time and effort to grasp the framework's intricacies.

Complexity for Simple Projects: Next.js is powerful but may be overkill for simple applications or websites that do not require server-side rendering or static site generation. Using Next.js in such cases might add unnecessary complexity to the project.

**Limited Server Control:** Next.js abstracts away many server details, which can be advantageous for most applications. However, if you require granular control over the server, you may find Next.js somewhat limiting.

Overall, Next.js is a versatile framework that excels in server-side rendering, static site generation, and building performant React applications. It offers a range of benefits, but it's essential to consider the specific needs of your project and evaluate whether Next.js aligns with those requirements.

## 3.4 Beego

### 3.4.1 Definition

Beego is an open-source web application framework written in the Go programming language. It follows the Model-View-Controller (MVC) architectural pattern and provides developers with a set of tools and libraries to build efficient and scalable web applications.

### 3.4.2 Uses

**Web Application Development:** Beego is primarily used for building web applications in Go. It offers a structured approach to development, making it easier to organize code and build scalable applications.

**API Development:** Beego can be used to develop RESTful APIs, providing a framework for routing, request parsing, response formatting, and authentication.

### 3.4.3 Strengths

**Easy to Get Started:** Beego offers a simple and straightforward learning curve, making it accessible for developers new to Go or web development frameworks.

**MVC Architecture:** Beego follows the MVC architectural pattern, promoting separation of concerns and allowing for clean code organization.

**Built-in Features:** Beego provides a range of built-in features, including automatic routing, form validation, logging, caching, and internationalization support. These features help developers save time and effort by reducing the need to build common functionalities from scratch.

**Scalability:** Beego is designed to handle high-traffic and concurrent requests efficiently. It leverages Goroutines and channels in Go to handle multiple requests concurrently, making it suitable for building scalable applications.

### 3.4.4 Drawbacks

**Limited Community:** Compared to some other web frameworks in different programming languages, Beego has a smaller community and ecosystem. This might result in fewer available resources, plugins, and community support compared to more popular frameworks.

**Documentation:** Although Beego has documentation, it may not be as comprehensive or up-to-date as some other frameworks. Developers may need to rely on the Beego source code or community forums to find solutions to specific problems.

**Flexibility vs. Opinionated:** Beego is opinionated in its approach, which means it may enforce certain conventions and limit flexibility compared to more flexible frameworks. This can be a drawback for developers who prefer a high degree of customization and flexibility.

Beego is a feature-rich web application framework for building web applications and APIs in Go. It offers a simple learning curve, built-in features, and follows the MVC architectural pattern. While it has its strengths in terms of ease of use and scalability, it also has limitations in terms of community size and flexibility. It's important for developers to evaluate their specific requirements and preferences before choosing Beego or any other web framework.

## 3.5 MongoDB

### 3.5.1 Definition

MongoDB is a popular open-source NoSQL database management system. It falls under the category of document-oriented databases, designed to store, retrieve, and manage unstructured and semi-structured data in a flexible and scalable manner. MongoDB stores data in flexible, JSON-like documents, making it easy to represent complex hierarchical relationships and accommodate evolving data models.

### 3.5.2 Uses

**Flexible Data Model:** MongoDB's document-oriented approach makes it suitable for handling a wide range of data types and structures, making it ideal for use cases with evolving or unpredictable data models.

**Scalability:** MongoDB offers horizontal scalability by allowing data to be distributed across multiple servers, enabling high-performance and efficient scaling of read and write operations as data volumes increase.

**Real-time Analytics:** MongoDB's ability to store and query large volumes of data in near real-time makes it well-suited for applications that require fast and complex data analysis, such as real-time analytics, logging, and event-driven systems.

**Content Management Systems:** MongoDB's flexible schema and ability to handle a variety of data types make it a popular choice for content management systems, allowing developers to store and retrieve rich, unstructured content efficiently.

### 3.5.3 Strengths

**Flexibility:** MongoDB's flexible document-based model allows for easy and dynamic schema changes, providing agility and adaptability to evolving data requirements.

**Scalability and Performance:** MongoDB's architecture is designed to scale horizontally, allowing for efficient distribution of data across multiple servers and delivering high-performance read and write operations.

**Rich Query Language:** MongoDB's query language (MongoDB Query Language or MQL) offers a comprehensive set of querying capabilities, including support for complex filtering, aggregation, indexing, and geospatial queries.

**Replication and Fault Tolerance:** MongoDB supports replica sets, providing data redundancy and automatic failover, ensuring high availability and fault tolerance in case of server failures.

### 3.5.4 Drawbacks

**Lack of Transactions:** MongoDB, until version 4.0, did not support multi-document ACID transactions, which could be a limitation for applications requiring complex transactional operations.

**Memory and Storage Requirements:** MongoDB's data storage efficiency can be lower compared to traditional relational databases, requiring more memory and disk space for storing indexes and document structures.

**Complex Configuration:** MongoDB's flexibility can lead to complex configuration and management, especially in distributed environments, requiring careful planning and expertise to optimize performance and ensure data consistency.

**Learning Curve:** MongoDB's query language and data model may require a learning curve for developers accustomed to relational databases, as it deviates from the traditional SQL-based approach.

It's worth noting that MongoDB has been continuously evolving, and some of the drawbacks mentioned above may have been addressed or improved in newer versions. It's recommended to refer to the official MongoDB documentation and stay updated with the latest releases to fully understand its capabilities and limitations.

# CHAPTER 4: MODELING REQUIREMENT

## 4.1 User requirement

### 4.1.1 Functional requirement

#### 4.1.1.1 Functional business requirement

|  |  |  |  |
| --- | --- | --- | --- |
| **Order** | **Requirement** | **Type of requirement** | **Note** |
| 1 | Store hotels | Storage | Store information of hotel such as: name, location, images, etc |
| 2 | Store news | Storage | Store name and content of news |
| 3 | Store rooms | Storage | Store information of room such as: price, number of bed, fee policy, etc |
| 4 | Store comments | Storage | Store content, post date, rating star of comment and information about user who post the comment |
| 5 | Store accounts | Storage | Store information of account such as: user name, password, etc |
| 6 | Store order | Storage | Store information of orser such as: check in and check out date, ordered rooms, user who ordered, etc |
| 7 | Store amenities | Storage | Store icon and description of amenity |
| 8 | Search for hotels | Search | Search by location, check in and check out date, maximum of guest and additional filter |
| 9 | Calculate statistic | Calculation | Calculate statistic by day and month. Data will be refresh after a year |
| 10 | Calculate invoice base on order | Calculation | Calculate the total amount of money of an order base on number of ordered room, ordered day, guest, type of room, ordered hotel |

Table 5: Function business requirement

#### 4.1.1.2 Functional system requirement

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Content | Description | Note |
| 1 | Management | Admin: View account info, view all available hotels and rooms, view posted news, view all comments, view statistic, view all orders  Host: View host’s hotel and room, view posted news, view all comments of host’s hotel, view statistic, view all orders of host’s hotel  Customer: View all available hotels and rooms, view posted news, view comments of hotels |  |

Table 6: Function system requirement

4.1.2 Non-functional requirement

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Content** | **Standard** | **Note** |
| 1 | The software can be edited and upgraded to suit the requirement and needs of host. | Evolution |  |
| 2 | * User-friendly. * Easy to manipulate. * The software interacts well with many devices. * The function buttons are arranged not too complicated and easy to see. * Functional monitors interact well with each other, creating convenience for users. | Convenience |  |
| 3 | * Databases are stored safely and easily retrieved. * The software operates stably, fast access and processing speed. | Effectiveness |  |
| 4 | The software can meet the user's requirements without affecting the operation of other machines. | Compatibility |  |
| 5 | The design and functionality of the software can be reused for later developments.  The code can be used many times and can be applied to many different programs without having to change the code too much. | Reusability |  |

Table 7: Non-functional requirement

## 4.2 Identify actors and use cases

|  |  |
| --- | --- |
| **Actor** | **Use case** |
| Admin | * Manage account * Manage hotel * Manage room * Manage comment * Manage order * Manage news * Manage statistic * Manage price |
| Host | * Login * Register * Manage hotel * Manage room * Manage comment * Manage order * Manage news * Manage statistic * Manage price |
| Customer | * Book hotel * Add comment * View hotel * View news * Login * Register * Search hotel |

Table 8: Actors and use cases

## 4.3 Detailed description of each usecase and each actor

### 4.3.1 Actor description

|  |  |
| --- | --- |
| Actor | Use case |
| Admin | * Add account * Update account * Delete account * Add hotel * Update hotel * Delete hotel * Add room * Update room * Delete room * Add comment * Update comment * Delete comment * Add order * Update order * Delete order * Add news * Update news * Delete news * View statistic * Export statistic * Add price * Update price * Delete Price |
| Host | * Login * Register * Update hotel * Add room * Update room * Delete room * Add comment * Update comment * Delete comment * Add order * Update order * Delete order * Add news * Update news * Delete news * View statistic * Export statistic * Add price * Update price * Delete Price |
| Customer | * Book hotel * Add comment * View hotel * View news * Login * Register * Search hotel |

Table 9: Actor description

### 4.3.2 Use case description

|  |  |  |
| --- | --- | --- |
| No. | Use case | Description |
| 1 | Login | Login to client/admin side to access functions of customer/host |
| 2 | Register | Create a account |
| 3 | View hotel | View a list hotels on a specific location |
| 4 | View news | View news that are posted by host |
| 5 | Search hotel | Search hotel by location, check in and check out date, number of guest and additional filter |
| 6 | Manage account | Add, update, delete user account |
| 7 | Manage hotel | Add, update, delete hotel and rooms of the hotel |
| 8 | Manage room | Add, update, delete room and fee policy of the room |
| 9 | Manage comment | Add reply comment to customer’s comment, update comment, delete reported comment |
| 10 | Manage order | Add, update, delete order |
| 11 | Manage news | Add, update, delete news |
| 12 | Manage statistic | View statistic by day and month; export statistic to excel file |
| 13 | Manage price | Add, update, delete price (price by day, by event or percent promotion) |

Table 10: Use case description

## 4.4 Use case diagram

**General use case diagram**

****

Figure 7: General use case diagram

**Admin use case diagram**

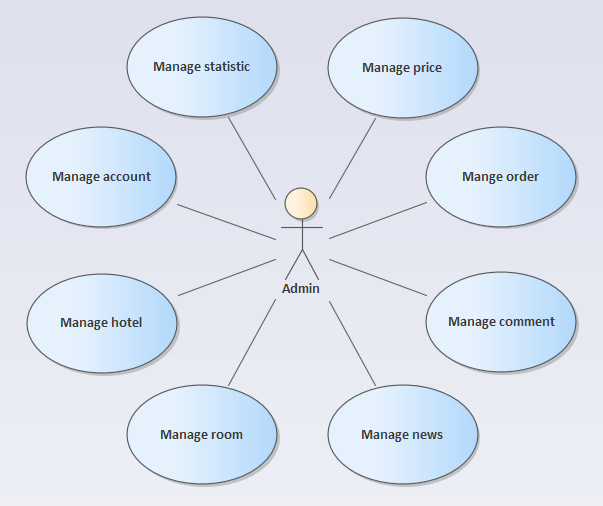
****

Figure 8: Admin use case diagram

**Host use case diagram**

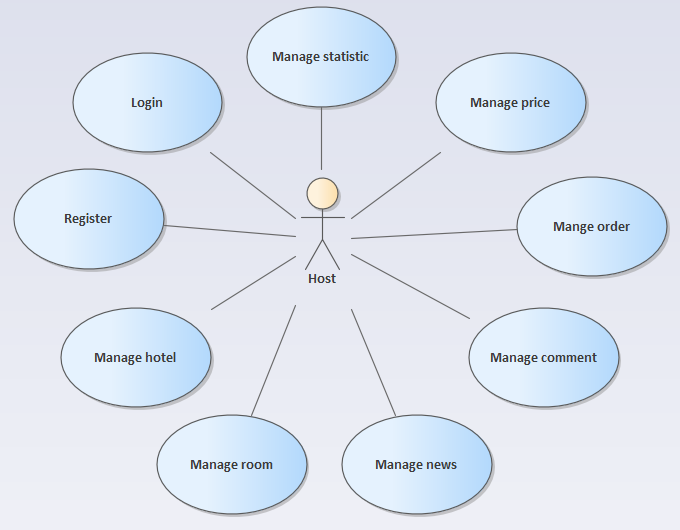
****

Figure 9: Host use case diagram

**Customer use case diagram**

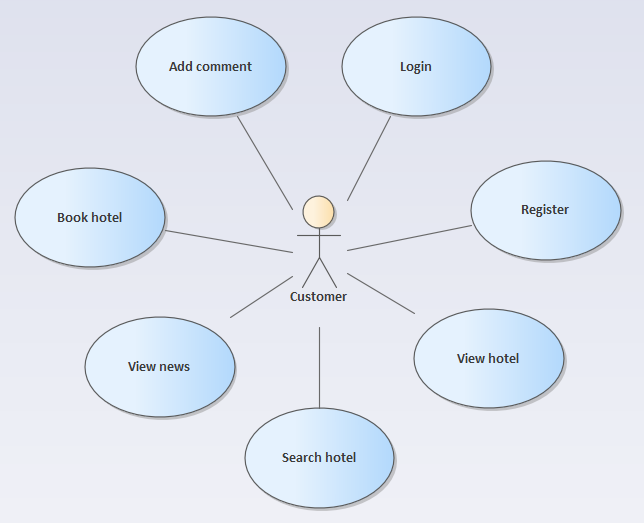
****

Figure 10: Customer use case diagram

## 4.5 Sequence diagram

1. Use case Log in

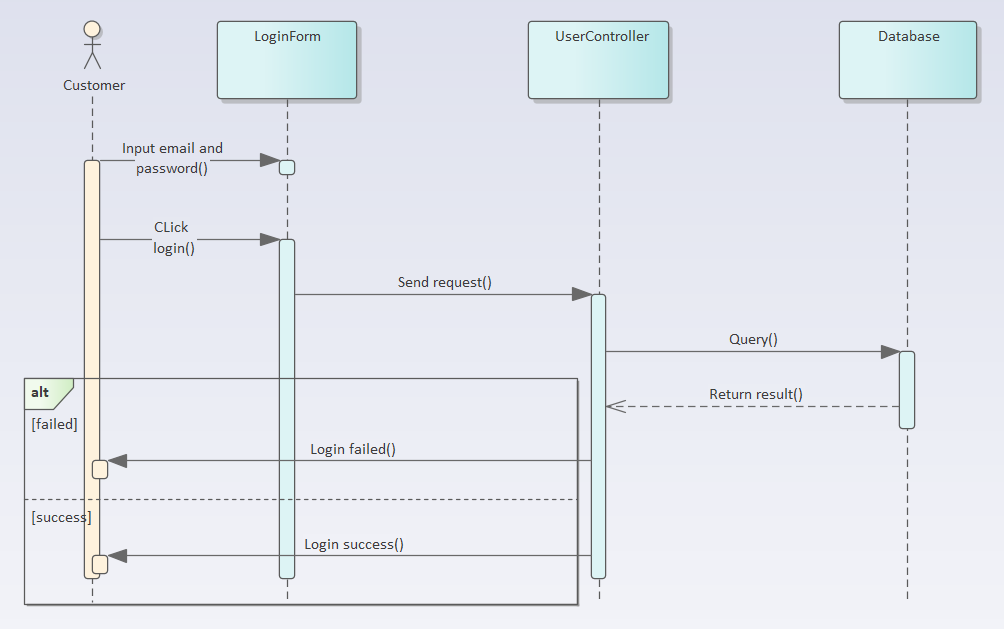


Figure 11: Log in sequence diagram

2. Use case Register

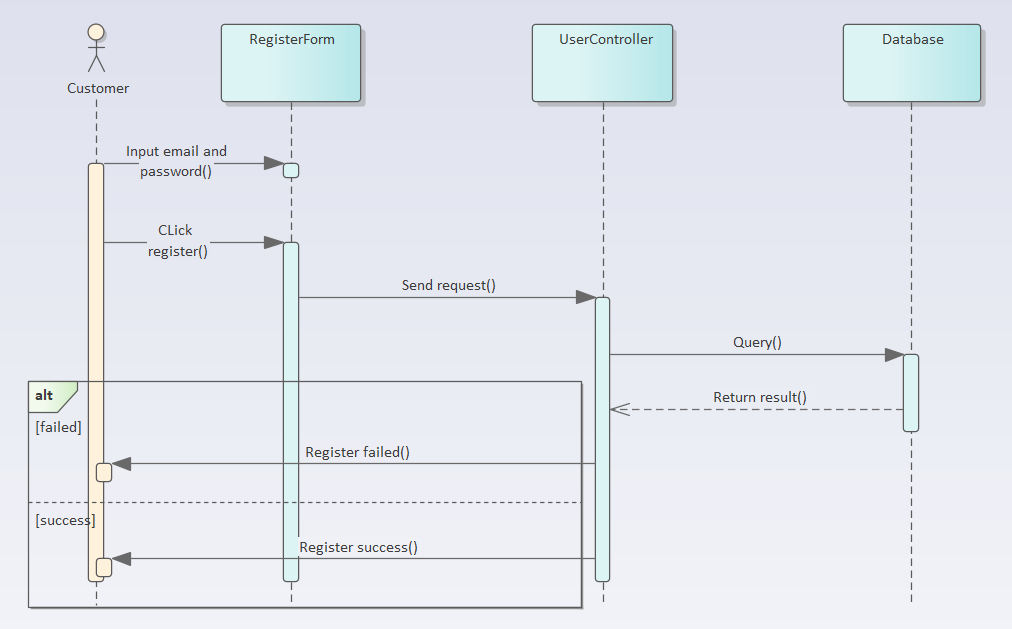


Figure 12: Register sequence diagram

3. Use case Search hotel

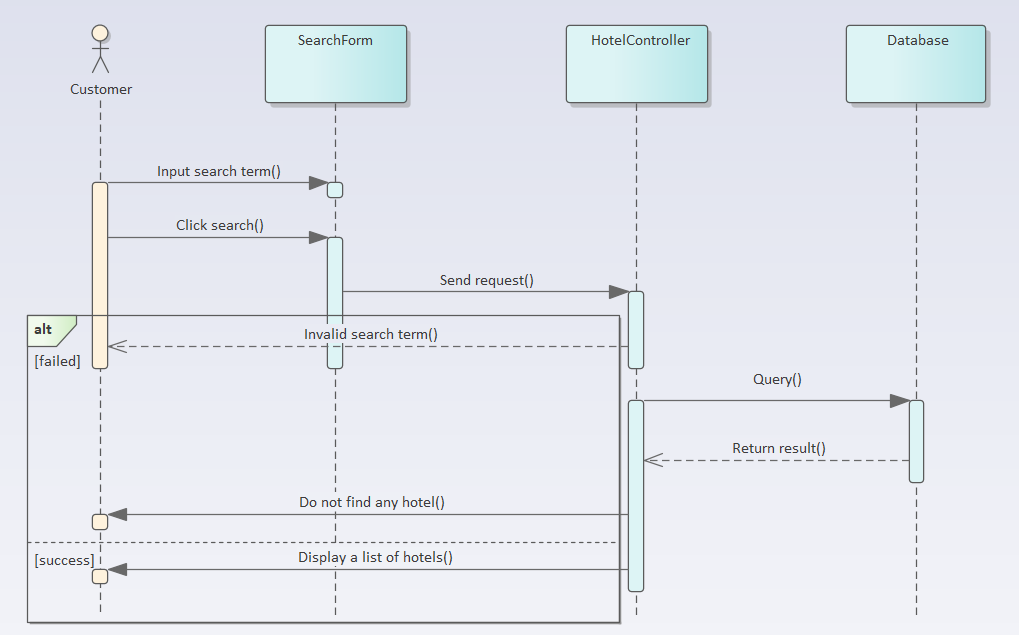


Figure 13: Search hotel sequence diagram

4. Use case Create hotel

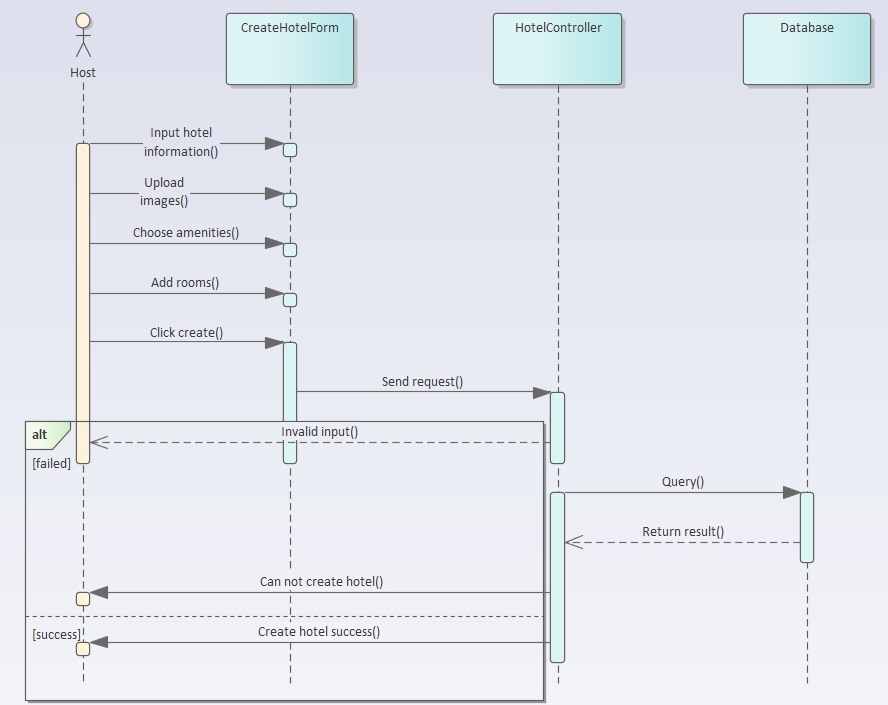


Figure 14: Create hotel sequence diagram

5. Use case Create order

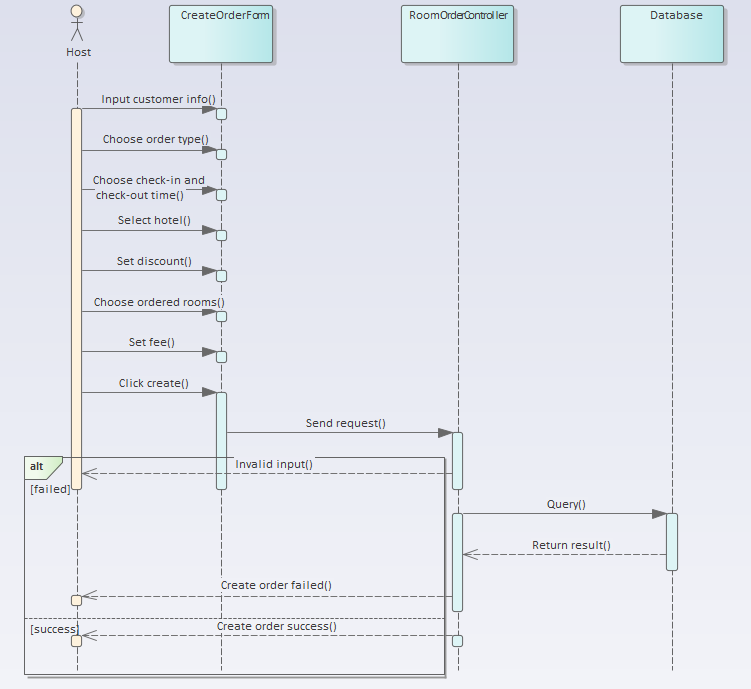


Figure 15: Create order sequence diagram

6. Use case Book hotel

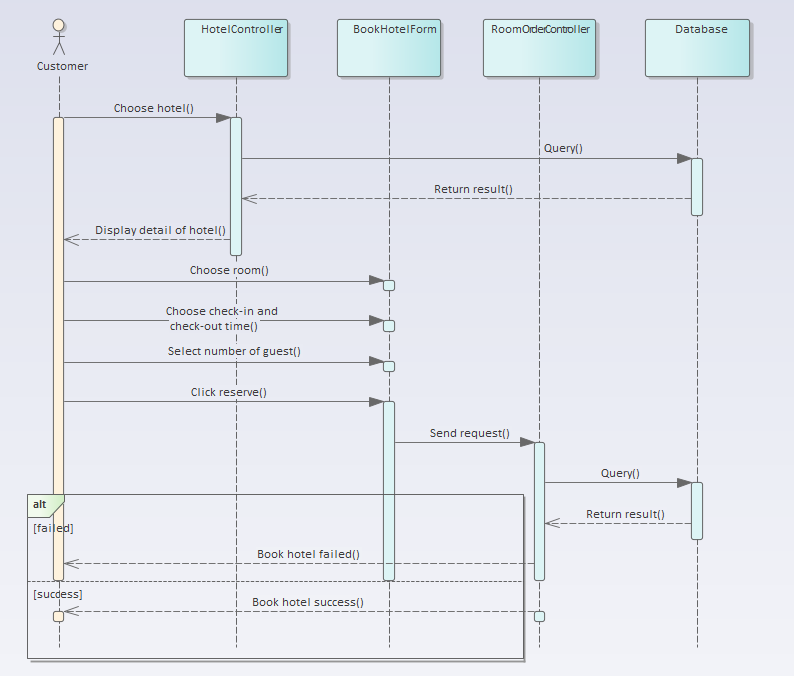


Figure 16: Book hotel sequence diagram

7. Use case Create news

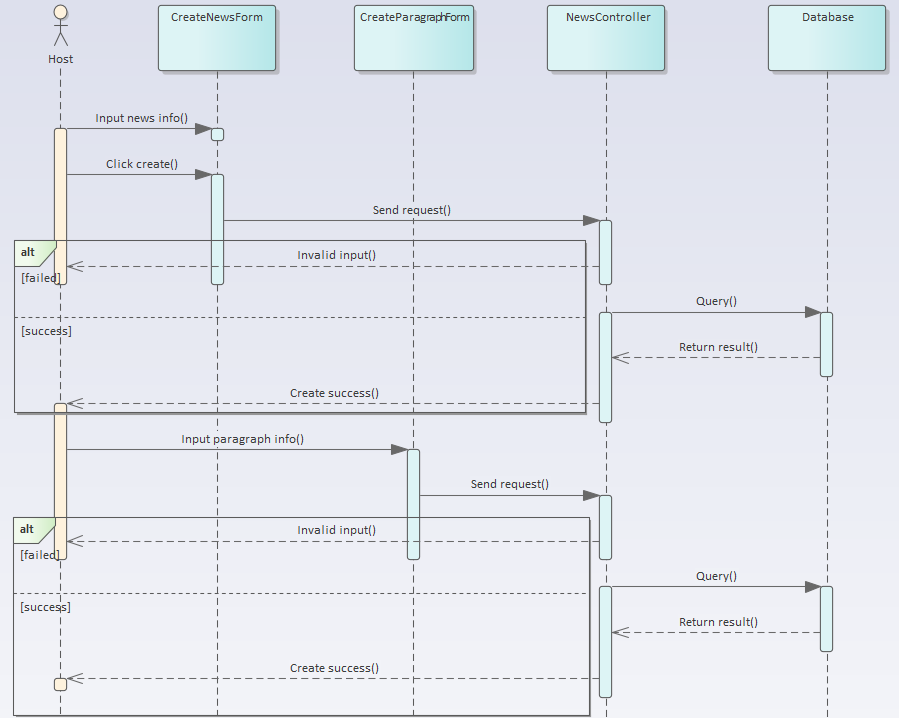


Figure 17: Create news sequence diagram

8. Use case Create room

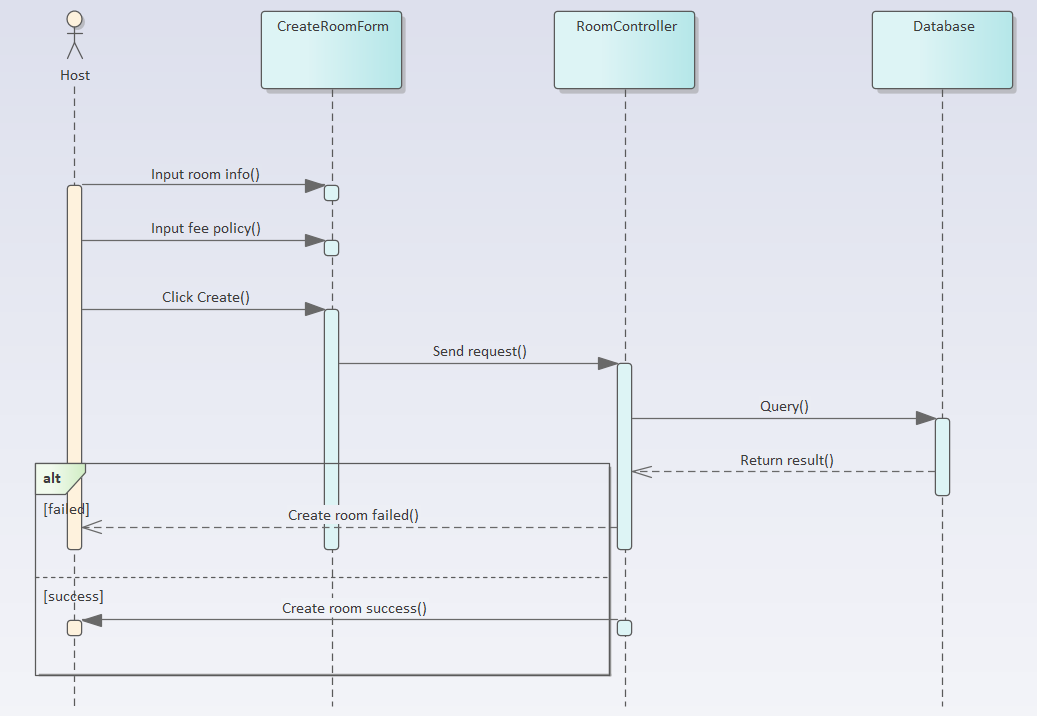


Figure 18: Create room sequence diagram

9. Use case Add promotion

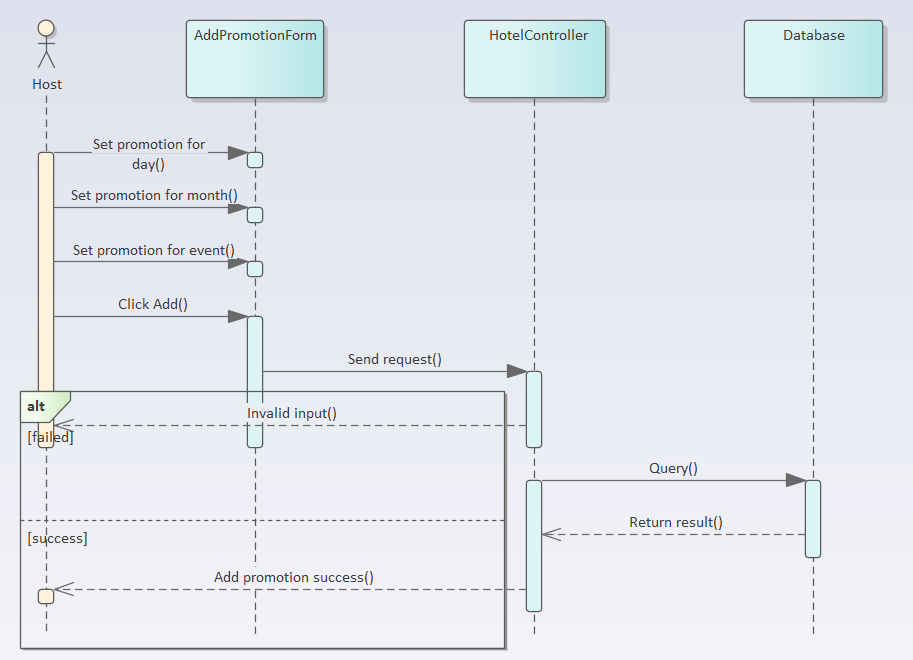


Figure 19: Add promotion sequence diagram

10. Use case Manage comment



Figure 20: Manage comment sequence diagram

11. Use case Manage amenity

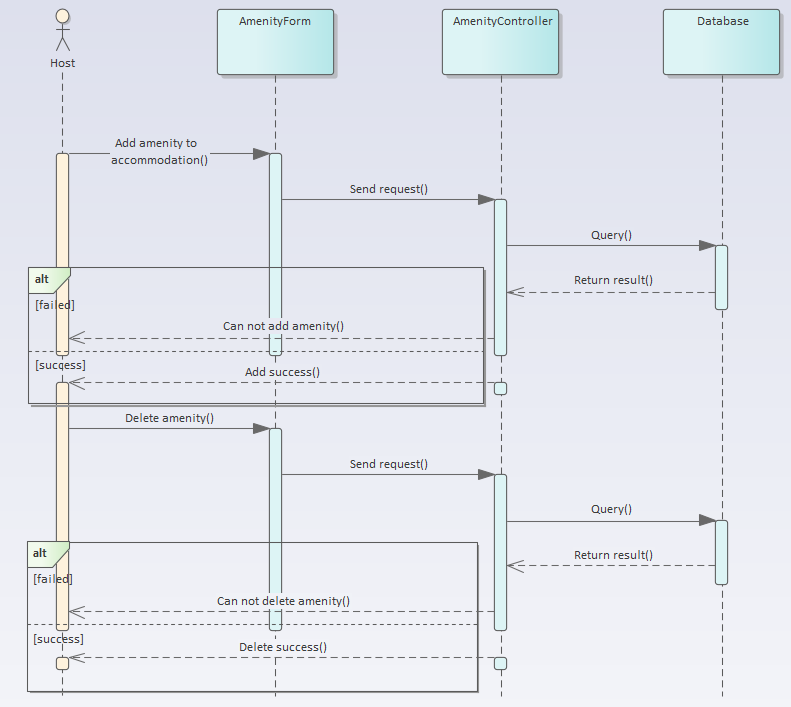


Figure 21: Manage amenity sequence diagram

12. Use case Export statistic

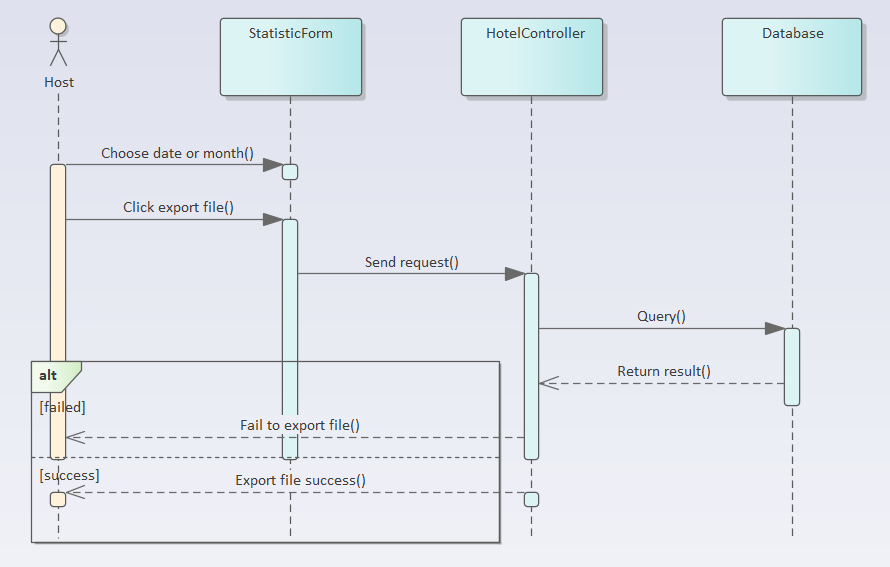


Figure 22: Export statistic sequence diagram

## 4.6 Use case specification

**1. Use case Log in**

|  |  |
| --- | --- |
| **Use Case Login** | |
| **Describe** | Allow actor to log into the system. |
| **Actor** | Customer |
| **Pre-Conditional** | Actor know the account and login password |
| **Steps** | 1. Actor open website. 2. Input user name and password 3. Press the "Sign in" button or press Enter 4. If the login fails, the message "Enter the wrong account name or password". The Customer/Manager re-entered the information and logged in again. 5. If the login is successful. Direct to homepage of the website. |

Table 11: Use case Log in specification

**2. Use case Register**

|  |  |
| --- | --- |
| **Use Case Register** | |
| **Describe** | Actor registers an account in the website |
| **Actor** | Customer |
| **Pre-Conditional** |  |
| **Steps** | 1. Actor open website. 2. Actor click to the Register (Sign up) Button and will be navigated to the Register Screen (Page). 3. Actor need to fill in all required information and have that information checked for validation before sign up. 4. Actor click on the Register to create a new account (after checking whether the account has existed or the password is correct or not). |

Table 12: Use case Register specification

**3. Use case View hotel**

|  |  |
| --- | --- |
| **Use Case View hotel** | |
| **Describe** | Actor view a list of hotel in a specific location |
| **Actor** | Customer |
| **Pre-Conditional** |  |
| **Steps** | 1. Actor open website. 2. Select a desire location. 3. Direct to view page and show a list of hotels that match the location. |

Table 13: Use case View hotel specification

**4. Use case View news**

|  |  |
| --- | --- |
| **Use Case View news** | |
| **Describe** | Actor view a list of posted news. |
| **Actor** | Customer |
| **Pre-Conditional** |  |
| **Steps** | 1. Actor open website. 2. Go to News page. 3. Display a list of news. |

Table 14: Use case View news specification

**5. Use case Search hotel**

|  |  |
| --- | --- |
| **Use Case Search hotel** | |
| **Describe** | Actor search for hotels base on certain criteria. |
| **Actor** | Customer |
| **Pre-Conditional** | Actor know the search term |
| **Steps** | 1. Actor open website. 2. Enter the search criteria then click on “Search” button or press “Enter”. If the search criteria is not valid, a warning will pop up tell the user to enter search criteria again. 3. Direct to view page and show a list of hotels that match the search criteria. |

Table 15: Use case Search hotel specification

**6. Use case Manage account**

|  |  |
| --- | --- |
| **Use Case Manage account** | |
| **Describe** | Actor add a new account or update/delete the existed one. |
| **Actor** | Admin |
| **Pre-Conditional** |  |
| **Steps** | 1. Actor open website. 2. Go to manage account page. 3. Choose according function (add/update/delete) to continue. 4. Depend on function that the actor choosed, enter the required field if needed and click on Add/Update/Delete button. |

Table 16: Use case Manage account specification

**7. Use case Manage hotel**

|  |  |
| --- | --- |
| **Use Case Manage hotel** | |
| **Describe** | Actor add a new hotel or update/delete the existed one. |
| **Actor** | Host |
| **Pre-Conditional** | Actor logged in the website |
| **Steps** | 1. Actor open website. 2. Go to manage hotel page. 3. Select a hotel. 4. Choose according function (add/update/delete) to continue. 5. Depend on function that the actor choosed, enter the required field if needed and click on Add/Update/Delete button. |

Table 17: Use case Manage hotel specification

**8. Use case Manage room**

|  |  |
| --- | --- |
| **Use Case Manage room** | |
| **Describe** | Actor add new rooms or update/delete the existed one. |
| **Actor** | Host |
| **Pre-Conditional** | Actor logged in the website |
| **Steps** | 1. Actor open website. 2. Go to manage hotel page. 3. Select a hotel. 4. Go to manage room tab. 5. Depend on function that the actor choosed, enter the required field if needed and click on Add/Update/Delete button. |

Table 18: Use case Manage room specification

**9. Use case Manage comment**

|  |  |
| --- | --- |
| **Use Case Manage comment** | |
| **Describe** | Actor add reply comment to customer’s comment or update existed one or delete reported comment. |
| **Actor** | Host |
| **Pre-Conditional** | Actor logged in the website |
| **Steps** | 1. Actor open website. 2. Go to manage hotel page. 3. Select a hotel. 4. Go to manage comment tab. 5. Depend on function that the actor choosed, enter the required field if needed and click on Add/Update/Delete button. |

Table 19: Use case Manage comment specification

**10. Use case Manage order**

|  |  |
| --- | --- |
| **Use Case Manage order** | |
| **Describe** | Actor add/update/delete direct or online order. |
| **Actor** | Host |
| **Pre-Conditional** | Actor logged in the website |
| **Steps** | 1. Actor open website. 2. Go to manage order page. 3. Click on an order. 4. Depend on function that the actor choosed, enter the required field if needed and click on Add/Update/Delete button. |

Table 20: Use case Manage order specification

**11. Use case Manage news**

|  |  |
| --- | --- |
| **Use Case Manage news** | |
| **Describe** | Actor post a new news or update/delete the existed one. |
| **Actor** | Host |
| **Pre-Conditional** | Actor logged in the website |
| **Steps** | 1. Actor open website. 2. Go to manage news page. 3. Depend on function that the actor choosed, enter the required field if needed and click on Add/Update/Delete button. |

Table 21: Use case Manage news specification

**12. Use case Manage statistic**

|  |  |
| --- | --- |
| **Use Case Manage statistic** | |
| **Describe** | View statistic by day and month; export statistic to excel file. |
| **Actor** | Host |
| **Pre-Conditional** | Actor logged in the website |
| **Steps** | 1. Actor open website. 2. Go to manage hotel page. 3. Select a hotel 4. Go to manage statistic tab. 5. View/exported statistic by filter. |

Table 22: Use case Manage statistic specification

**13. Use case Manage price**

|  |  |
| --- | --- |
| **Use Case Manage price** | |
| **Describe** | Add price by event or month, update/delete existed one. |
| **Actor** | Host |
| **Pre-Conditional** | Actor logged in the website |
| **Steps** | 1. Actor open website. 2. Go to manage hotel page. 3. Select a hotel 4. Go to manage price tab. 5. Depend on function that the actor choosed, enter the required field if needed and click on Add/Update/Delete button. |

Table 23: Use case Manage price specification

# CHAPTER 5: DATABASE DESIGN

## 5.1 Attribute table

1. User

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | avatar | string |  | Avatar of user |  |
| 3 | name | string |  | Name of user |  |
| 4 | phoneNumber | string |  | Phone number of user |  |
| 5 | address | string |  | Address of user |  |
| 6 | reviewCount | uint |  | Number of times a user can comment |  |

Table 24: User model

2. Account

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | username | string |  | Username of account |  |
| 3 | password | string |  | Password of acocunt |  |
| 4 | email | string |  | Email of account |  |
| 5 | role | string |  | Role of account |  |
| 6 | userID | ObjectID |  | Identity of user |  |
| 7 | wishListHotelIDs | slice |  | List of hotel’s identities which added to wishlist |  |
| 8 | wishListVillaTownhouseIDs | slice |  | List of villa/townhouse’s identities which added to wishlist |  |

Table 25: Account model

3. Amenity

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | description | string |  | Description of amenity |  |
| 3 | icon | string |  | Icon of amenity |  |

Table 26: Amenity

4. Comment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | hotelID | ObjectID |  | Identity of hotel |  |
| 3 | date | Datetime |  | The date comment was addes |  |
| 4 | content | string |  | Content of comment |  |
| 5 | userID | ObjectID |  | Identity of user |  |
| 6 | userName | string |  | Name of user who post comment |  |
| 7 | userAvatar | string |  | Avatar of user who post comment |  |
| 8 | phoneNumber | string |  | Phone number of user who post comment |  |
| 9 | parentID | ObjectID |  | Identity of preceded comment |  |
| 10 | starRating | float32 |  | Number of star that user give within comment |  |
| 11 | level | uint8 |  | Level of comment |  |
| 12 | replyComment | slice |  | List of reply comments |  |

Table 27: Comment model

5. Contact

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | facebook | string |  | Facebook of contact |  |
| 3 | email | string |  | Email of contact |  |
| 4 | phonenumber | string |  | Phone number of contact |  |
| 5 | address | string |  | Address of contact |  |
| 6 | youtube | string |  | Youtube of contact |  |
| 7 | instagram | string |  | Instagram of contact |  |

Table 28: Contact model

6. Holiday

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | hotelID | ObjectID |  | Identity of hotel |  |
| 3 | villaID | ObjectID |  | Identity of villa |  |
| 4 | townHouseID | ObjectID |  | Identity of townhouse |  |
| 5 | name | string |  | Name of holiday |  |
| 6 | date | Datetime |  | Date of holiday |  |
| 7 | fee | float32 |  | Fee of holiday |  |
| 8 | promotion | float32 |  | Promotion of holiday |  |

Table 29: Holiday model

7. Hotel

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | ownerID | ObjectID |  | Identity of owner |  |
| 3 | name | string |  | Name of hotel |  |
| 4 | address | string |  | Address of hotel |  |
| 5 | description | string |  | Description about hotel |  |
| 6 | images | slice |  | List of images of hotel |  |
| 7 | star | float32 |  | Number of stars of hotel |  |
| 8 | totalRoom | int64 |  | Number of rooms of hotel |  |
| 9 | dayOrderMaxRoom | int |  | Number of maximum rooms can ordered in one day |  |
| 10 | promotionDescription | string |  | Description of promotion of hotel |  |
| 11 | needToContact | bool |  | Check whether need to contact to book hotel | false: Don’t need to contact  true: need to contact |
| 12 | amenities | slice |  | List of amenities of hotel |  |
| 13 | rank | float32 |  | Number of review star of hotel |  |
| 14 | contactInfor | string |  | Contact info of hotel |  |
| 15 | minRoomPrice | float32 |  | Minimum price of hotel |  |
| 16 | deposit | float32 |  | The percent of deposit of hotel |  |
| 17 | images360 | slice |  | List of 360 images of hotel |  |
| 18 | lat | float32 |  | Latitude of hotel |  |
| 19 | lng | float32 |  | Longtitude of hotel |  |

Table 30: Hotel model

8. MonthlyFee

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | hotelID | ObjectID |  | Identity of hotel |  |
| 3 | villaID | ObjectID |  | Identity of villa |  |
| 4 | townHouseID | ObjectID |  | Indentity of townhouse |  |
| 5 | feeType | int |  | Type of monthly fee |  |
| 6 | month | int |  | Month that the fee is activated |  |
| 7 | sundayFee | float32 |  | Fee on Sunday |  |
| 8 | saturdayFee | float32 |  | Fee on Saturday |  |
| 9 | normalDayFee | float32 |  | Fee on normal day |  |

Table 31: MonthlyFee model

9. News

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | title | string |  | Title of news |  |
| 3 | thumbnail | string |  | Thumbnail of news |  |
| 4 | time | Datetime |  | The time when news is post |  |
| 5 | tag | string |  | Tag of news |  |

Table 32: News model

10. NewsContent

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | newsID | ObjectID |  | Identity of news |  |
| 3 | header | string |  | Header of news content |  |
| 4 | image | string |  | Image of news content |  |
| 5 | highLight | string |  | Highlight of news content |  |
| 6 | text | string |  | Content of news content |  |

Table 33: News content model

11. Room

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | hotelID | ObjectID |  | Identity of hotel |  |
| 3 | roomNo | string |  | Room number |  |
| 4 | dayPrice | float32 |  | Price by day of room |  |
| 5 | numberOfBed | uint8 |  | Number of beds of room |  |
| 6 | hourFeePolicies | slice |  | Hour fee policy of room |  |
| 7 | blocked | bool |  | Decide whether a room is allowed for online ordering or not |  |

Table 34: Room model

12. RoomOrder

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | checkIn | DateTime |  | Check-in time of order |  |
| 3 | checkOut | DateTime |  | Check-out time of order |  |
| 4 | gmail | string |  | Email of user who book order |  |
| 5 | phoneNumber | string |  | Phone number of user who book order |  |
| 6 | userName | string |  | Username of user who book order |  |
| 7 | numberOfCustomer | uint8 |  | Number of customers of order |  |
| 8 | isFullyPaid | bool |  | Check whether user paid all fee for order | true: pay all fee  false: not pay all fee |
| 9 | isGroupOrder | bool |  | Check whether order is booked for group tenants or not | true: order booked by group tenants  false: order is not booked by group tenants |
| 10 | createdBy | string |  | Order is created by client or admin |  |
| 11 | mustPayDeposit | float32 |  | Amount of deposit that customer must pay |  |
| 12 | paidDeposit | float32 |  | Amount of money that customer paid |  |
| 13 | discountInPercentage | float32 |  | Percentage of discount of order |  |
| 14 | discountInCash | float32 |  | Amount of discount in cash of order |  |
| 15 | typeOfDiscount | int |  | Type of discount of order | 0: by percentage  1: by cash |
| 16 | vat | float32 |  | Percentage of VAT of order |  |
| 17 | vatInPrice | float32 |  | VAT of order in cash |  |
| 18 | orderType | uint |  | Type of order | 0: hour order  1: day order |
| 19 | hotelID | ObjectID |  | Identity of hotel |  |
| 20 | roomIDs | slice |  | List of room IDs of order |  |
| 21 | userID | ObjectID |  | Identity of user |  |
| 22 | roomSurcharges | slice |  | List of surcharges of order |  |
| 23 | totalPrice | float32 |  | Total price of order |  |
| 24 | remain | float32 |  | Remain fee of order |  |
| 25 | roomPrice | float32 |  | Default price of ordered rooms |  |

Table 35: RoomOrder model

13. VillaTownhouse

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | name | string |  | Name of villa/townhouse |  |
| 3 | address | string |  | Address of villa/townhouse |  |
| 4 | images | slice |  | List of images of villa/townhouse |  |
| 5 | description | string |  | Description of villa/townhouse |  |
| 6 | price | float32 |  | Price of villa/townhouse |  |
| 7 | surchargeFee | float32 |  | Surcharge of villa/townhouse |  |
| 8 | promotion | float32 |  | Promotion of villa/townhouse |  |
| 9 | star | float32 |  | Star of villa/townhouse |  |
| 10 | cancelFee | float32 |  | Cancel fee of villa/townhouse |  |
| 11 | depositPrice | float32 |  | Percentage of deposit of villa/townhouse |  |
| 12 | date | slice |  |  |  |
| 13 | numberOfCustomer | uint8 |  | Maximum number of customers that can order villa/townhouse |  |
| 14 | type | uint8 |  |  |  |
| 15 | commentIDs | slice |  | List of comments of villa/townhouse |  |
| 16 | promotionDescription | string |  | Promotion description of villa/townhouse |  |
| 17 | needToContact | bool |  | Check whether need to contact villa/townhoue’s host for ordering |  |
| 18 | contactInfor | string |  | Contact info of villa/townhouse |  |
| 19 | available | bool |  | Check whether a villa/townhouse is ordered or not |  |
| 20 | amenities | slice |  | List of amenities of villa/townhouse |  |
| 21 | images360 | slice |  | List of 360 images of villa/townhouse |  |
| 22 | lat | float32 |  | Latitude of villa/townhouse |  |
| 23 | lng | float32 |  | Longtitude of villa/townhouse |  |

Table 36: VillaTownhouse model

14. VillaTownhouseOrder

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Type** | **Range** | **Meaning** | **Note** |
| 1 | \_id | ObjectID |  | Identity |  |
| 2 | checkIn | DateTime |  | Check-in time of order |  |
| 3 | checkOut | DateTime |  | Check-out time of order |  |
| 4 | gmail | string |  | Email of user who book order |  |
| 5 | phoneNumber | string |  | Phone number of user who book order |  |
| 6 | userName | string |  | Username of user who book order |  |
| 7 | numberOfCustomer | uint8 |  | Number of customers of order |  |
| 8 | isFullyPaid | bool |  | Check whether user paid all fee for order |  |
| 9 | isGroupOrder | bool |  | Check whether order is booked for group tenants or not |  |
| 10 | createdBy | string |  | Order is created by client or admin |  |
| 11 | mustPayDeposit | float32 |  | Amount of deposit that customer must pay |  |
| 12 | paidDeposit | float32 |  | Amount of money that customer paid |  |
| 13 | discountInPercentage | float32 |  | Percentage of discount of order |  |
| 14 | discountInCash | float32 |  | Amount of discount in cash of order |  |
| 15 | typeOfDiscount | int |  | Type of discount of order |  |
| 16 | vat | float32 |  | Percentage of VAT of order |  |
| 17 | vatInPrice | float32 |  | VAT of order in cash |  |
| 18 | villaID | ObjectID |  | Identity of villa |  |
| 19 | townhouseID | ObjectID |  | Identity of townhouse |  |
| 20 | userID | ObjectID |  | Identity of user |  |
| 21 | surcharges | slice |  | List of surcharges of order |  |
| 22 | villaTownhousePrice | float32 |  | Default price of villa/townhouse |  |
| 23 | totalPrice | float32 |  | Total price of order |  |
| 24 | remain | float32 |  | Remain fee of order |  |
| 25 | orderType | uint8 |  | Type of order |  |

Table 37: VillaTownhouseOrder model

## 5.2 Entity relationship diagram



Figure 23: Entity relationship diagram

# CHAPTER 6: USER INTERFACE DESIGN

## 6.1 Admin site

**1. Log in page**

A screenshot of a website

Description automatically generated

**1**

**3**

**2**

Figure 24: Login page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Input (Account) | Input account to sign in |
| 2 | Input (Password) | Input password to sign in |
| 3 | Button (Sign in) | Sign in the Admin page |

Table 38: Log in page component

A green square with white text

Description automatically generated

Figure 25: Log in page event diagram

**2. Admin page**

A screenshot of a computer

Description automatically generated

**13**

**12010**

**11**

**10010**

**9**

**6**

**8**

**7**

**4**

**5**

**3**

**2**

**1**

Figure 26: Admin page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button (Hotel management ) | User move to hotel management features |
| 2 | Button (Villa management) | User move to villa management features |
| 3 | Button ( House management) | User move to town house management features |
| 4 | Button (news management) | User move to News management features |
| 5 | Button (Account management) | User move to account management features |
| 6 | Button (Hotel bill) | User move to hotel bill management features |
| 7 | Button ( Villa bill) | User move to villa bill management features |
| 8 | Button (Town house bill) | User move to town house bill management features |
| 9 | Setting | User move to setting features |
| 10 | Button ( New hotel addition) | Create new hotel |
| 11 | Button ( Edit) | Move to Edit hotel page for edit information of existed hotel |
| 12 | Button ( Deleting) | Delete information of existed hotel |
| 13 | Button ( Show room ) | Show the room exist in the hotel |

Table 39: Admin page component

A green rectangular object with white text

Description automatically generated

Figure 27: Admin page event diagram

**3. Hotel management page**

A screenshot of a computer

Description automatically generated

**8**

**7**

**6**

**5**

**4**

**3**

**2**

**1**

Figure 28: Hotel management page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button (Edit hotel info) | Move to edit hotel information part |
| 2 | Button (Room management) | Move to room management part |
| 3 | Button (Price management) | Move to price management part |
| 4 | Button (Comment management) | Move to comment management part |
| 5 | Button ( Amenity management) | Move to amenity management part |
| 6 | Button ( Statistical) | Move to statistical part |
| 7 | Button ( Cancel changes) | Use to cancel the changes |
| 8 | Button (Save change) | Use to save the information have been updated later |

Table 40: Hotel management page component

A green rectangular box with white text

Description automatically generated

Figure 29: Hotel management page event diagram

**4. Villa management page**

A screenshot of a computer

Description automatically generated

**2**

**3**

**1**

Figure 30: Villa management page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button ( New Villa addition) | Add new villa |
| 2 | Button ( Editing) | Edit the information of existed villa |
| 3 | Button (Deleting ) | Delete the information of existed villa |

Table 41: Villa management page component

A diagram of a computer program

Description automatically generated

Figure 31: Villa management page event diagram

**5. Townhouse management page**

A screenshot of a computer

Description automatically generated

**2\**

**3**

**1**

Figure 32: Townhouse management page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button ( House addition) | Add new town house |
| 2 | Button ( Editing ) | Edit the information of existed town house |
| 3 | Button (Deleting ) | Delete the information of existed town house |

Table 42: Townhouse management page component

A diagram of a computer program

Description automatically generated

Figure 33: Townhouse management page event diagram

**6. News page**

A screenshot of a computer

Description automatically generated

**3**

**2**

**1**

Figure 34: News page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button ( News addition) | Add news information |
| 2 | Button ( Editing) | Edit information of existed news |
| 3 | Button (Deleting ) | Delete information of existed news |

Table 43: News page component

A diagram of a computer program

Description automatically generated

Figure 35: News page event diagram

**7. Account management page**

A screenshot of a computer

Description automatically generated

**2**

**3**

**1**

Figure 36: Account management page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button ( Account addition) | Add new account |
| 2 | Button ( Deleting) | Delete information of existed account |
| 3 | Button (Editing) | Edit information of existed account |

Table 44: Account management page component

A diagram of a computer program

Description automatically generated

Figure 37: Account management page event diagram

**8. Hotel bill page**

A screenshot of a computer

Description automatically generated

**4**

**3**

**2**

**1**

Figure 38: Hotel bill page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button ( Hotel choice ) | Choose the hotel want to see bill |
| 2 | Button ( New order addition) | Create new order |
| 3 | Button ( Modifying) | Edit information of existed order |
| 4 | Button (Deleting ) | Delete the existed order |

Table 45: Hotel bill page component

A diagram of a computer program

Description automatically generated

Figure 39: Hotel bill page event diagram

## 6.2 Client site

**1. Sign up page**

A screenshot of a computer

Description automatically generated

**6**

**5**

**4**

**3**

**2**

**1**

Figure 40: Sign up page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button ( Sign up with Google) | Use account google to sign up |
| 2 | Input (Email) | Type email to sign up |
| 3 | Input (Password) | Type password to sign up |
| 4 | Input ( Retype password) | Retype password to sign up |
| 5 | Button (Sign in) | Sign in if users have account already |
| 6 | Button (Sign up) | Sign up after type enough information |

Table 46: Sign up page component

A diagram of a sign up button

Description automatically generated

Figure 41: Sign up page event diagram

**2. Booking page**

A screenshot of a website

Description automatically generated

**12**

**11**

**10**

**9**

**8**

**7**

**6**

**5**

**4**

**3**

**2**

**1**

Figure 42: Booking page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button (Homepage) | Move to home page |
| 2 | Input(Place to go) | Type the place users want to go |
| 3 | Button (Choose checkin date) | Choose the checkin date |
| 4 | Button (Choose checkout date) | Choose the checkout date |
| 5 | Button (Choose number of people) | Choose the number of people |
| 6 | Button (Search) | After type the information then search |
| 7 | Button ( News ) | Move to News page |
| 8 | Button (Contact) | Move to contact page |
| 9 | Button (User) | Move to users page or sign up and sign in page |
| 10 | Button ( 360 image) | Open 360 degree image of the hotel/villa/town house |
| 11 | Button (Map) | Open map where the hotel/villa/town house locate |
| 12 | Button ( Booking) | Go to the detail page for booking |

Table 47: Booking page component

A screenshot of a computer screen

Description automatically generated

Figure 43: Booking page event diagram

**3. Payment page**

A screenshot of a computer

Description automatically generated

**6**

**5**

**4**

**3**

**2**

**1**

Figure 44: Payment page

|  |  |  |
| --- | --- | --- |
| Order | Type | Meaning |
| 1 | Button(Edit date) | Edit the checkin and checkout date |
| 2 | Button(Edit number of people) | Edit the number of people |
| 3 | Input (Full name) | Type full name |
| 4 | Input (Phone numbers) | Type phone numbers |
| 5 | Input (Email) | Type email to receive confirmation mail |
| 6 | Button (Choose payment method) | Choose the payment method to pay |

Table 48: Payment page component

A diagram of a button

Description automatically generated

Figure 45: Payment page event diagram

# CHAPTER 7: SOFTWARE TESTING

## 7.1 Test Plan

### 7.1.1 Test Strategy

#### 7.1.1.1 Test Objectives

The objective of the test is to verify that the functionality of the Website For Selling Train Ticket works according to the specifications.

The test will execute and verify the test scripts, identify, fix and retest all high and medium severity defects per the entrance criteria, prioritize lower severity defects for future fixing.

The final product of the testing is:

* A complete and ready-to-go website.
* A set of stable test scripts that can be reused for Functional.

#### 7.1.1.2 Test Assumptions

**Main assumption**

* Ticket issuance is the same as the requested data and is available in the system before the Functional Test starts

**General**

* Exploratory Testing would be carried out once the build is ready for testing
* Performance testing is not considered for this estimation.
* All the defects would come along with a snapshot JPEG format
* The Test Team will be provided with access to Test environment via VPN connectivity
* The Test Team assumes that all necessary inputs required during the design and execution of the Test will be appropriately supported by someone with experience.
* Test case design activities will be performed by each people in the team
* Test environment and preparation activities will be owned by Dev Team
* Dev team will provide Defect fix plans based on the Defect meetings during each cycle to plan.
* The same will be informed to Test team prior to start of Defect fix cycles
* Any defect fixes planned will be shared with Test Team prior to applying the fixes on the Test environment
* Project Manager will review and sign-off all test deliverables
* The project will provide test planning, test design and test execution support
* Test team will manage the testing effort with close coordination with Project Manager

**Analyst**

* Project team has the knowledge and experience necessary, or has received adequate training in the system, the project and the testing processes.
* The system will be treated as a black box; if the information shows online and correct with the report, it will be assumed that the database is working properly.

**Functional Testing**

* During Functional testing, testing team will use preloaded data which is available on the system at the time of execution
* The Test Team will be perform Functional testing only on the Website For Selling Train Ticket

#### 7.1.1.3 Test Principles

* Testing will be focused on meeting the business objectives, and quality.
* There will be common, consistent procedures for all teams supporting testing activities.
* Testing processes will be well defined, yet flexible, with the ability to change as needed.
* Testing activities will build upon previous stages to avoid redundancy or duplication of effort.
* Testing environment and data will emulate a production environment as much as possible.
* Testing will be a repeatable, quantifiable, and measurable activity.
* Testing will be divided into distinct phases, each with clearly defined objectives and goals.
* There will be entrance and exit criteria.

#### 7.1.1.4 Data Approach

In functional testing, test data will be stored in Excel file for testing activities.

### 7.1.2 Execution Strategy

#### 7.1.2.1 Test Stages

There will be two cycles for functional testing. Each cycle will execute all the scripts.

The objective of the first cycle is to identify any blocking, critical defects, and most of the high defects. It is expected to use some work-around in order to get to all the scripts.

The objective of the second cycle is to identify remaining high and medium defects, remove the work-around from the first cycle, correct gaps in the scripts and obtain performance results.

#### 7.1.2.2 Type of Testing

**Functional testing**

|  |  |
| --- | --- |
| **Purpose of testing** | Ensure the functions tested work correctly according to the required specification |
| **Technique** | Execute all possible scenarios for each functional group, using valid and invalid data to determine:  - Expected results when valid data is used  - Appropriate warning appears when invalid data is used |
| **Standard stop** | All designed test cases are executed.  All errors found are clearly documented to help the developer fix them. |
| **Responsible for testing** | Test Designer / Tester |
| **How to test** | Manual manual testing, sequentially following the steps defined in the testcase |
| **Exception handling** | List all related issues that arise during test execution. |

Table 49: Type of testing

#### 7.1.2.3 Defect Tracking

##### 7.1.2.3.1 Error Classification

|  |  |
| --- | --- |
| **Severity** | **Impact** |
| 1 (Critical) | + This bug is critical enough to crash the system, cause file corruption, or cause potential data loss  + It causes an abnormal return to the operating system (crash or a system failure message appears).  + It causes the application to hang and requires re-booting the system |
| 2 (High) | + It causes a lack of vital program functionality with workaround. |
| 3 (Medium) | + This Bug will degrade the quality of the System. However there is an intelligent workaround for achieving the desired functionality - for example through another screen.  + This bug prevents other areas of the product from being tested. However other areas can be independently tested. |
| 4 (Low) | + There is an insufficient or unclear error message, which has minimum impact on product use. |
| 5(Cosmetic) | + There is an insufficient or unclear error message that has no impact on product use. |

Table 50: Error classification

##### 7.1.2.3.2 Defect Tracking Process

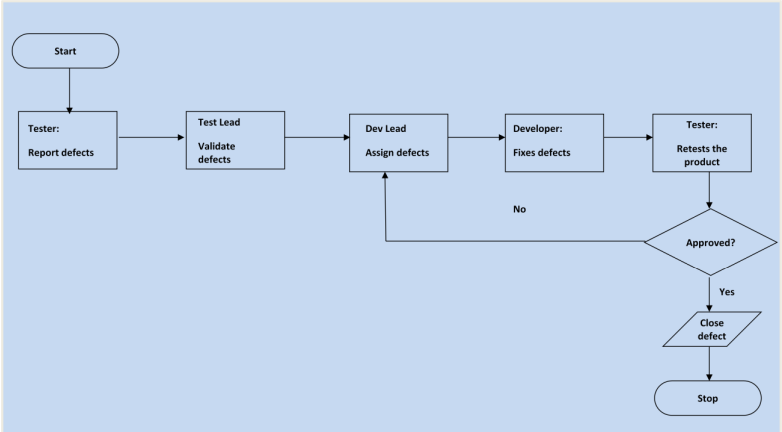


Figure 46: Defect tracking process

### 7.1.3 Test Environment

Personal computer (no Internet needed).

Functions of the application will be tested in Google Chrome browser (Version 107.0.5304.107).

Operating system: Windows 10 Home Single Language.

## 7.2 Test Process

### 7.2.1 Unit Test

* A unit is the smallest piece of software that can be tested.
* Unit testing refers to tests to verify the functionality of a particular part of code, usually at the function level. In an object-oriented environment, unit testing is often used at the class level and testing the smallest units including constructors and destructors.
* Unit testing is usually done by programmers. This should be done as early as possible in the coding phase and throughout the software development cycle. A function can have multiple tests, to catch cases or branches in the code.

We are using this technique for testing the combination of functions in our project. We have demoed this technique in Chapter 4. The testing process will check whether the functions work together correctly based on the user requirement.

### 7.2.2 Integration Test

Integration testing combines the components of an application and tests as a finished application. While Unit Test tests individual components and Units, Integration Test brings them together and tests the communication between them.

Integration Test has 2 main goals:

* Detect communication errors occurring between Units.
* Integrate single units into small systems (subsystems) and finally a complete system (system) preparing for system-level testing (System Test).

There are 4 types of tests in Integration Test:

* Structure Test: Similar to White Box Test (testing to ensure that the internal components of a program run properly), it focuses on the operation of internal structural elements of the program such as instructions and internal branches.
* Functional Test: Similar to Black Box Test (testing only focuses on the function of the program, not interested in the internal structure), only examines the function of the program according to the technical requirements.
* Performance Test: Test the operation of the system.
* Stress Test: Test the system's limits.

We are using this technique for testing functions in our project. We have demoed this technique in 6.2.4. The testing process will check whether the functions separately work correctly based on the user requirement.

### 7.2.3 System Test

System testing (ST) includes a series of tests to verify that all the components of a system are properly integrated. The purpose of ST is to make sure the whole system works as the customer wants.

Usually, this type of check takes a lot of effort and time. In many cases, testing requires some auxiliary equipment, specific software or hardware, especially real-time applications, distributed systems, or embedded systems. At the system level, testers also look for defects, but the focus is on evaluating the performance, operation, reliability and other quality related requirements of the whole system.

System test usually has the following types of tests:

* Functional Test: ensure that the behavior of the system meets the design requirements.
* Performance Test: ensure optimal allocation of system resources (eg memory) to achieve targets such as processing time or query response...
* Stress Test or Load Test: Make sure the system operates properly under high pressure
* Configuration Test
* Security Test: ensure the integrity and security of the data and of the system.
* Recovery Test: ensuring the system is capable of recovering to a previous stable state in the event of a loss of resources or data; especially important for transaction systems such as online banking

In this project, we create many constraints, transactions, … For the transaction, we create and test it in database to ensure the recovery event when losing data. Furthermore, we test each function according to the user requirement to ensure that the behavior of project meet the design requirement. Because we use local database, and the source is not too big so it could work at an optimal way in some respects

### 7.2.4 Acceptance Test

There are two methods for acceptance test:

* Alpha Test: carried out where the organization develops but is not tested by the development team, but by the customer.
* Beta Test: made by customers or potential customers power in their place

Because we are developer team, and this is a course project so we do not have customer or organization. However, to demo this kind of test, some members work as a stranger/customer and test our project and we also invite our friends to use our project

### 7.2.5 White Box Testing

White box testing, also known as structural testing or clear box testing, is a software testing technique that involves testing the internal workings of an application or system. In this approach, the tester has access to the code and is aware of the internal structure, design, and implementation of the system.

The primary goal of white box testing is to ensure that the code is functioning as intended and meets the specified requirements. This type of testing focuses on verifying the internal logic, flow of data, and algorithms used in the code.

Some of the commonly used techniques for white box testing include code coverage analysis, path testing, control flow testing, data flow testing, and mutation testing.

White box testing can be used for various purposes, including:

* Verifying the correctness of the code,
* Identifying defects early,
* Improving code quality,
* Ensuring code maintainability,
* Ensuring code security,
* Optimizing code performance.

We use white box testing to ensure that the code is reliable, robust, and performs as intended, while also improving its quality and maintainability.

### 7.2.6 Black Box Testing

Black box testing is a software testing technique that focuses on testing the functionality of an application or system without any knowledge of its internal workings, design, or implementation. In this approach, the tester is only concerned with the inputs and outputs of the system and how it behaves under different conditions.

Black box testing is typically performed from a user's perspective, where the tester assumes the role of a typical user and tests the system based on its expected behavior.

Some of the commonly used techniques for black box testing include equivalence partitioning, boundary value analysis, decision table testing, state transition testing, and exploratory testing.

In this project, black box testing help ensure that the system meets the specified requirements and functions as expected under various scenarios, while also identifying potential usability and functionality issues.

# CHAPTER 8. CONCLUSION

## 8.1 Achievements

### 8.1.1 Knowledge

* Gained a deep understanding of software development processes and techniques, including agile development methodologies, software testing and quality assurance, and user experience design.
* Understand more about project management principles and practices to software development projects to ensure that they are completed on time, within budget, and to the required quality standards.
* Improves the ability to work effectively in a team. Furthermore, the process teaches effective project planning, including how to initiate and distribute tasks evenly among team members.

### 8.1.2 Skills

* Developed a wide range of technical, interpersonal, and teamwork skills throughout the project.
* Strengthened our problem-solving skills by identifying and resolving technical issues, managing project risks, and adapting to changing project requirements.

### 8.1.3 Product

* Developed a accommodation booking website allows customers to search for and book accommodations quickly and easily, providing them with a seamless user experience.
* The website is reliable and performs well, ensuring that customers have a positive experience when using it.
* The website delivered a wide range of accommodation options, real-time availability, competitive pricing, and convenient booking features, meeting the needs and expectations of both travelers and accommodation providers.

## 8.2 Strengths and Drawbacks

### 8.2.1 Strengths

* The website is designed with ease of maintenance and future development in mind, ensuring that any updates or changes can be made quickly and efficiently.
* The website also provide a mobile app version for more convenient for customers.
* The website included an extensive choice of accommodations, convenience and efficiency in the booking process, real-time availability information, competitive pricing, and user reviews and ratings for informed decision-making
* Effective communication, agile development methodologies, and attention to detail

### 8.2.2 Drawbacks

* The website currently does not provide services for many places
* The project experienced delays due to unforeseen technical issues that required additional time and resources to resolve.
* We also encountered some communication breakdowns between team members that led to misunderstandings and mistakes.

## 8.3 Future Work

* Enhancing the website's search functionality to allow customers to search for hotels by additional criteria, such as: distance to center, bed type deals and discounts, etc.
* Expanding website to allow booking in all over the world.

# REFERENCES

1. TypeScript Quickly. (2020, February 10). Scribd. <https://www.scribd.com/book/511816861/TypeScript-Quickly>
2. Crockford, D. (2006). The application/json Media Type for JavaScript Object Notation (JSON). <https://doi.org/10.17487/rfc4627>
3. Qing, F., Jiao, L., Dai, C., Ziqiang, D., & Zhang, R. (2019). Golang-Based POI Discovery and Recommendation in Real Time. <https://doi.org/10.1109/mdm.2019.00114>
4. Li, C., & Yang, W. (2014). The distributed storage strategy research of remote sensing image based on Mongo DB. <https://doi.org/10.1109/eorsa.2014.6927858>
5. Liu, S. S., & Mattila, A. S. (2017). Airbnb: Online targeted advertising, sense of power, and consumer decisions. International Journal of Hospitality Management, 60, 33–41. <https://doi.org/10.1016/j.ijhm.2016.09.012>
6. Tích hợp cổng thanh toán Vn-Pay với ứng dụng của bạn - NTechDevelopers. NTechDevelopers - Programs must be written for people to read, and only incidentally for machines to execute. <https://blog.ntechdevelopers.com/tich-hop-cong-thanh-toan-vn-pay-voi-ung-dung-cua-ban/>
7. Set up online and mobile payment gateway integration. (n.d.). <https://developer.paypal.com/docs/online/>
8. System Design VN. (2023). Thiết Kế Hệ Thống Airbnb - Viblo. Viblo. <https://viblo.asia/p/thiet-ke-he-thong-airbnb-x7Z4DYX2JnX?fbclid=IwAR3b3YZ9WohLrKXLtyulZTCstJrMtqCfNj6ajEK1vsf11zHMZL794pNze24>
9. YvonneDev. (2022, August 12). Meet Mantine: A TS-Based Open-Source React Components Library. Medium. <https://javascript.plainenglish.io/new-popular-replacement-ui-mantine-829aca3be18a>
10. Vitejs. (n.d.). GitHub - vitejs/vite-plugin-react: The all-in-one Vite plugin for React projects. GitHub. <https://github.com/vitejs/vite-plugin-react>
11. Agrawal, A. (2021, December 12). Building an API Using Beego + Swagger - MDG Space - Medium. Medium. <https://medium.com/mobile-development-group/how-beego-swagger-can-make-coding-an-api-easier-227b56055cbc>
12. Reduxjs. (n.d.). GitHub - reduxjs/redux-thunk: Thunk middleware for Redux. GitHub. <https://github.com/reduxjs/redux-thunk>

# 

# WORKLOAD TABLE

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name** | **Workload** | **Percentage** |
| 19110107 | Khúc Nguyễn Huy Cường | - Determine requirements  - Design user interface  - Connect API | 35% |
| 19110064 | Nguyễn Đan Trường | - Analyze requirements  - Distribute tasks to team members  - Design responsive interface | 30% |
| 19110104 | Đặng Ngọc Trường Giang | - Design model  - Design database  - Write API | 35% |

Table 51: Workload table