HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INTELLIGENCE AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

AIR BOOKING ONLINE SYSTEM

NGUYỄN XUÂN KIÊN kien.nx215217@sis.hust.edu.vn

MAI HOÀNG ĐỨC duc.mh215195@sis.hust.edu.vn

Major:Intelligence and Communication Technology
Specialization: Global ICT

Supervisor: Dr. TRUONG THI DIEU LINH

Hanoi, 30/01/2024

PROJECT REPORT TOPIC

Full name of 1st student: Mai Hoàng Đức Student ID: 20215195
Full name of 2nd student: Nguyễn Xuân Kiên Student ID: 20215217
Cohort: 66 School: School of Intelligence and Communication Technology

1. Topic

   Air booking online system

2. Content

   Chapter 1: Introduction

   Chapter 2: Requirement analysis

   Chapter 3: Methodology

   Chapter 4: Application Design

   Chapter 5: Result

0.1 Member contribution

Member	Task	Contribution
Mai Hoàng Đức	Server, Client, System functionalities, Report, Presentation	65%
Nguyễn Xuân Kiên	Client functionality, Report	35%

Table 0.1 Member contribution

3. Topic assignment date: 01/02/2024

4. Topic complete date: 31/01/2024

January 31st, 2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In our Network Programming course, we embarked on an ambitious project to develop an FTP server-client application using the C/C++ languages. Our aim was to construct a system that enables secure and efficient file transfers. We initiated the project by delving into the intricacies of the FTP protocol, understanding its primary functions. Utilizing C/C++, we then engineered a server capable of handling multiple client requests concurrently, thus facilitating smooth file uploads and downloads.

The success of our project hinged on collaborative coding and seamless integration of components. We meticulously ensured that our system could handle file transfers securely and rapidly, enhancing user interaction with features like robust authentication, streamlined directory navigation, and effective remote file management.

In essence, the project in our Network Programming class culminated in the creation of a fully operational FTP server-client application in C/C++. This endeavor not only bolstered our understanding of network programming concepts but also provided us with hands-on experience in the field. The application is a testament to our collective effort, coordination, and commitment to become excellent in network programming.

CHAPTER 1: Introduction

1.1 Motivation

Today's society witnesses a significant rise in travelling by airplane. History of airplane started with the first flight in 1903 by Oliver Wright, then at 1918, the US used Airmail which is the service for moving mails by air, after that, the first commercial flight happenned in 1952 between London and Johnnesburg. However, the aviation only incresead exponentially in commercial flights when Boeing 707 invented in 1958 and Fedex implemented Door-to-Door service that guarantee the goods will safely send to the customers on time. With the advantage of being the fastest and the safest transportation in the world, more and more people choose airplane and buy tickets from the airlines.

As the Internet is widely used today, our groups decided to develop a online booking system which help the air tickets transaction between the customers and airplanes. The system will apply the client-server model using TCP/IP with the client is a UI for the customers or the admins, the server serves as a online manager manage the transaction and responsible for the change in airlines' database.

1.2 Objective and the scope of the project

As mentioned above, the system will be apply the client-server model using C++ with 5 main functions which are login, sign up, searching tickets with different methods, booking tickets, managing tickets and updating from the admin. For the system testing, we will also create three databases which contain users' information, tickets' information and bookings' information. The client side will show menu with different commands for users to choose. send the commands of users to the server side and the server will implement functions following the command from the client and also send back the confirmation message to the clients. In detail, the server will open users' database to verify the users' accounts or update new accounts, open tickets' database to search tickets satisfy some conditions, add or delete a tickets and also adding and deleting booking in the bookings' database. The scope of the project encompasses these objectives, aiming to deliver a functional and terminal air booking online system.

1.3 Tentative solution

After the initial requirements of the project, which is described in section 1.2, we commence the solution for the problem. Protocol design is crucial in computer network-

ing, ensuring efficient and standardized communication between devices and systems. Well-designed protocols facilitate interoperability, scalability, and security. They define rules for data exchange, error handling, and system behavior, forming the foundation for reliable and standardized interactions in the digital landscape.

1.4 Project report organization

The report is structured into four chapters, each addressing specific aspects of the development process. In Chapter 2, the Methodology is discussed, outlining the adopted modified waterfall model that guides the systematic development approach. This chapter covers the stages of requirements analysis, system design, implementation, testing, deployment, feedback, iteration, and documentation. Chapter 3 provide the detail of the system design as well as delving into the comprehensive exploration and documentation of functional and non-functional requirements for the Air booking online system. Chapter 4 encapsulates the Conclusion, summarizing key findings, lessons learned, and the overall success of the FTP program's development within the outlined methodology. This structured organization provides a coherent and detailed exploration of the project, from requirement analysis through to the concluding insights and outcome

CHAPTER 2: REQUIREMENT ANALYSIS

2.1 Overview

In Vietnam, the burgeoning flight industry features a robust selection of airline companies, each with a distinctive online presence. These websites are not merely informational but are pivotal in the provision of various services such as reservations, modifications, and cancellations of flights. Our project aims to dissect and comprehend the intricate rules and methodologies these airlines utilize in their network programming and TCP/IP communications. We will explore how these protocols can be optimized for handling the high volume of concurrent transactions inherent in flight bookings and cancellations.

In particular, we will focus on how TCP is employed to ensure data integrity and order, which are critical when dealing with sensitive transactions. This will enable us to engineer a system that is not only robust and secure but also capable of providing real-time data with minimal latency, enhancing the overall user experience.

For simplification, in our project, there is only 1 **Admin** with only one account which has the privilege to modify and view the database of the system. Furthermore, any other accounts connecting to the server are designated as user accounts within the flight management system. These accounts are equipped with the capability to book and cancel flights among other functions, but are expressly restricted from accessing the database.

2.2 Objective and the scope of the project

As mentioned above, the system will be apply the client-server model using C++ with 5 main functions which are login, sign up, searching tickets with different methods, booking tickets, managing tickets and updating from the admin. For the system testing, we will also create three databases which contain users' information, flights' information and bookings' information. The client side will show menu with different commands for users to choose. send the commands of users to the server side and the server will implement functions following the command from the client and also send back the confirmation message to the clients. In detail, the server will open users' database to verify the users' accounts or update new accounts, open tickets' database to search tickets satisfy some conditions, add or delete a tickets and also adding and deleting booking in the bookings' database. The scope of the project encompasses these objectives, aiming to deliver a functional and terminal air booking online system.

- **For the Administrator:**
  - Establish a secure connection to the server using predefined administrator credentials, with the username "admin" and password "1".
  - Add new flight schedules to the system's database to keep the offerings up-to-date.
  - Remove flights from the system that are no longer in operation, ensuring the accuracy of the database.
  - Update flight details to reflect changes in scheduling, pricing, or availability.
- **For the User:**
  - Access the server using personal account credentials. New users can create an account through a registration process.
  - Perform searches for flights that align with their travel preferences and requirements.
  - Reserve seats on desired flights, securing their travel plans.
  - Cancel bookings as necessary, providing flexibility in travel arrangements.
  - Oversee all booked tickets with options to:
    * View a comprehensive list of all tickets reserved.
    * Exchange tickets for alternatives that better suit changed plans.
    * Print the details of all tickets for personal records.
    * Print individual tickets for convenience and travel preparation.
- **Additional functionality:**
  - If users' flight is cancelled, they will receive immediately notification from the server.
  - Should there be any alterations to a flight's scheduled departure or return time, our system is designed to promptly notify users. For modifications occurring within a span of hours, users will receive a "delay notification," ensuring they are informed of the shorter-term changes to their itinerary. If the adjustments extend beyond hours and into days, a "schedule change notification" will be issued, providing users with details of the longer-term rescheduling of their flight

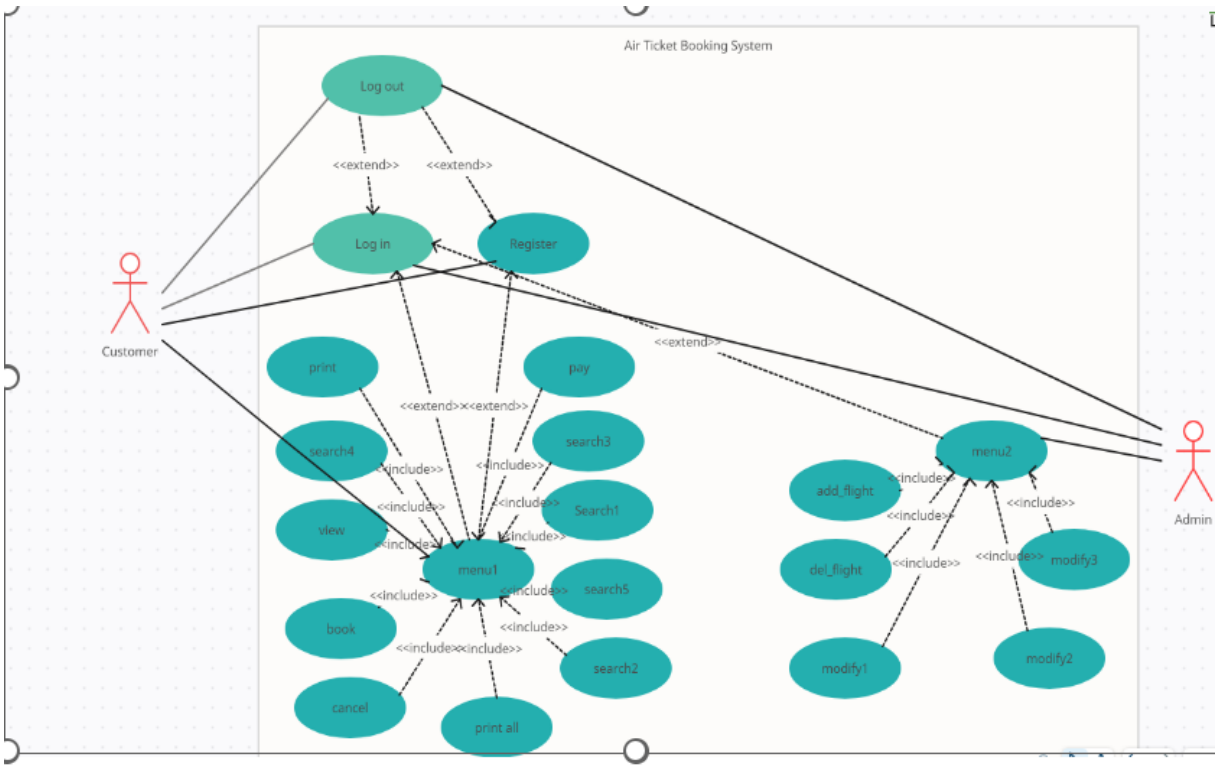

Figure 2.1 Use case diagram

2.3 Functional overview description

2.3.1 For Customers

- Description of login: Allowing user to access to the server's resource after valid login command.

Use case ID	UC01	**Use Case Name**	Login
Actor	Customer		
Description	Customers login into their accounts		
Precondition	Connect to the server and Customer type login command		
Postcondition	Notification failed or the menu2		
Flow	Index / Actor / Action 1 / Customer / login/username/password 2 / Server / result str		
Alternative flow	If the customer type invalid command or non-existence account, customers type again		

- Description of register If the new username is checked to have no appearance in the database, the new user is accepted and allowed to access the server's resource.

<table>
<tr><td>Use case ID</td><td>UC02</td><td>Use Case Name</td><td>Register</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers register new accounts</td></tr>
<tr><td>Precondition</td><td colspan="3">Connect to the server and Customer type register command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or menu2</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>register/username/password</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command, customers type again</td></tr>
</table>

- Description of Search1

<table>
<tr><td>Use case ID</td><td>UC03</td><td>Use Case Name</td><td>Search1</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers search for flight based on departure point and destination point</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 or UC02 and Customers type Search1 command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Info of the flight</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>search1/departure point/destination point</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or search for non-existent flight, customers type again</td></tr>
</table>

- Description of Search2

<table>
<tr><td>Use case ID</td><td>UC04</td><td>Use Case Name</td><td>Search2</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers search for flight based on departure point and destination point and departure date</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 or UC02 and Customers type Search2 command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Info of the flight</td></tr>
<tr><td>Flow</td><td colspan="3"><table>
<tr><td>Index</td><td>Actor</td><td>Action</td></tr>
<tr><td>1</td><td>Customer</td><td>search2/departure point/destination point/departure date</td></tr>
<tr><td>2</td><td>Server</td><td>result str</td></tr>
</table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or search for non-existent flight, customer type again</td></tr>
</table>

- Description of Search3

<table>
<tr><td>Use case ID</td><td>UC05</td><td>Use Case Name</td><td>Search3</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers search for flight based on departure point and destination point and company</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 or UC02 and Customers type Search3 command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Info of the flight</td></tr>
<tr><td>Flow</td><td colspan="3"><table>
<tr><td>Index</td><td>Actor</td><td>Action</td></tr>
<tr><td>1</td><td>Customer</td><td>search3/departure point/destination point/company</td></tr>
<tr><td>2</td><td>Server</td><td>result str</td></tr>
</table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or search for non-existent flight, customer type again</td></tr>
</table>

- Description of Search4

<table>
<tr><td>Use case ID</td><td>UC06</td><td>Use Case Name</td><td>Search4</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers search for flight based on departure point and destination point and departure date and return date</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 or UC02 and Customers type Search4 command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Info of the flight</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>search4/departure point/destination point/departure date/return date</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or search for non-existent flight, customer type again</td></tr>
</table>

- Description of Search5

<table>
<tr><td>Use case ID</td><td>UC07</td><td>Use Case Name</td><td>Search5</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers search for flight based on departure point and destination point and departure date and return date and company</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 or UC02 and Customers type Search5 command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Info of the flight</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>search5/departure point/destination point/departure date/return date/company</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or search for non-existent flight, customer type again</td></tr>
</table>

- Description of book

<table>
<tr><td>Use case ID</td><td>UC12</td><td>Use Case Name</td><td>book</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers book the flight based on filght number and the seat class they want</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 or UC02 and Customers type book command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Info of customer’s ticket</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>book/flight_num/seat_class</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or book for non-existent ticket or book flight that sold out, customer type again</td></tr>
</table>

- Description of view

<table>
<tr><td>Use case ID</td><td>UC13</td><td>Use Case Name</td><td>view</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers view their tickets by their ticket code</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 and UC02 and Customers type view command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Info of customer’s ticket</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>view/ticket_code</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or view for non-existent ticket, customer type again</td></tr>
</table>

- Description of print

<table>
<tr><td>Use case ID</td><td>UC14</td><td>Use Case Name</td><td>print</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers print their tickets with a certain format in the screen</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 and UC02 and Customers type print command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Format of customer’s ticket</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>print/ticket_code</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr><tr><td>3</td><td>Client</td><td>print</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or ask for print for non-existent ticket, customer type again</td></tr>
</table>

- Description of print_all

<table>
<tr><td>Use case ID</td><td>UC15</td><td>Use Case Name</td><td>print_all</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers print all their tickets with a certain format in the screen</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 and UC02 and Customers type print_all command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or Format of all customer’s ticket</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>print$_all$</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr><tr><td>3</td><td>Client</td><td>print</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or ask for print for non-existent ticket, customer type again</td></tr>
</table>

- Description of cancel

<table>
<tr><td>Use case ID</td><td>UC16</td><td>Use Case Name</td><td>cancel</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers cancel their tickets</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 and UC02 and Customers cancel command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or success</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>cancel/ticket_code</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or ask for cancel non-existent ticket, customer type again</td></tr>
</table>

- Description of change

<table>
<tr><td>Use case ID</td><td>UC17</td><td>Use Case Name</td><td>change</td></tr>
<tr><td>Actor</td><td colspan="3">Customer</td></tr>
<tr><td>Description</td><td colspan="3">Customers change their tickets</td></tr>
<tr><td>Precondition</td><td colspan="3">UC01 and UC02 and Customers change command</td></tr>
<tr><td>Postcondition</td><td colspan="3">Notification failed or success</td></tr>
<tr><td>Flow</td><td colspan="3"><table><tr><td>Index</td><td>Actor</td><td>Action</td></tr><tr><td>1</td><td>Customer</td><td>change/old_ticket_code/new_flight_num/new_seat</td></tr><tr><td>2</td><td>Server</td><td>result str</td></tr></table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the customer type invalid command or ask for change non-existent ticket or change to non-existent flight, customer type again</td></tr>
</table>

2.3.2 For Admin

- Description of login: If admin login with correct administrator account (admin,1), the client is allowed to make modification to the database.

Use case ID	UC08	**Use Case Name**	Register
Actor	Admin		
Description	Admin login		
Precondition	Connect to the server and Admin type login command		
Postcondition	user name and password match the server'data		
Flow	Index / Actor / Action 1 / Admin / login/admin/1 2 / Server / Y_admin		
Alternative flow	If the admin type invalid command, admin type again		

- Description of Add flight: Admin inserts a new flight to the database

Use case ID	UC08	**Use Case Name**	Register
Actor	Admin		
Description	Admin login		
Precondition	Connect to the server and Admin type add/$_{f}lightcommand$		
Postcondition	user name and password match the server'data		
Flow	Index / Actor / Action 1 / Admin / add_flgiht/parameter 2 / Server / Y_add		
Alternative flow	If the admin type invalid command, admin type again		

- Description of Modify flight:

Use case ID	UC10	**Use Case Name**	Modify
Actor	Admin		
Description	Admin modify		
Precondition	Connect to the server and Admin type modify command		
Postcondition	The parameters matchs		
Flow	Index / Actor / Action 1 / Admin / modify_flgiht/parameter 2 / Server / Y_modified		
Alternative flow	If the admin type invalid command, admin type again		

- Description of Delete flight:

<table>
<tr><td>Use case ID</td><td>UC11</td><td>Use Case Name</td><td>Delete</td></tr>
<tr><td>Actor</td><td colspan="3">Admin</td></tr>
<tr><td>Description</td><td colspan="3">Admin delete</td></tr>
<tr><td>Precondition</td><td colspan="3">Connect to the server and Admin type delete command</td></tr>
<tr><td>Postcondition</td><td colspan="3">The parameters matchs</td></tr>
<tr><td>Flow</td><td colspan="3"><table>
<tr><td>Index</td><td>Actor</td><td>Action</td></tr>
<tr><td>1</td><td>Admin</td><td>delete_flgiht/parameter</td></tr>
<tr><td>2</td><td>Server</td><td>Y_delete</td></tr>
</table></td></tr>
<tr><td>Alternative flow</td><td colspan="3">If the admin type invalid command, admin type again</td></tr>
</table>

CHAPTER 3: Methodology

The development of the air online booking program will follow a modified waterfall model to ensure a structured and systematic approach to the project. The modified waterfall model incorporates iterative elements to enhance flexibility and accommodate potential changes in requirements. The project will be divided into several stages, each focusing on specific aspects of development.

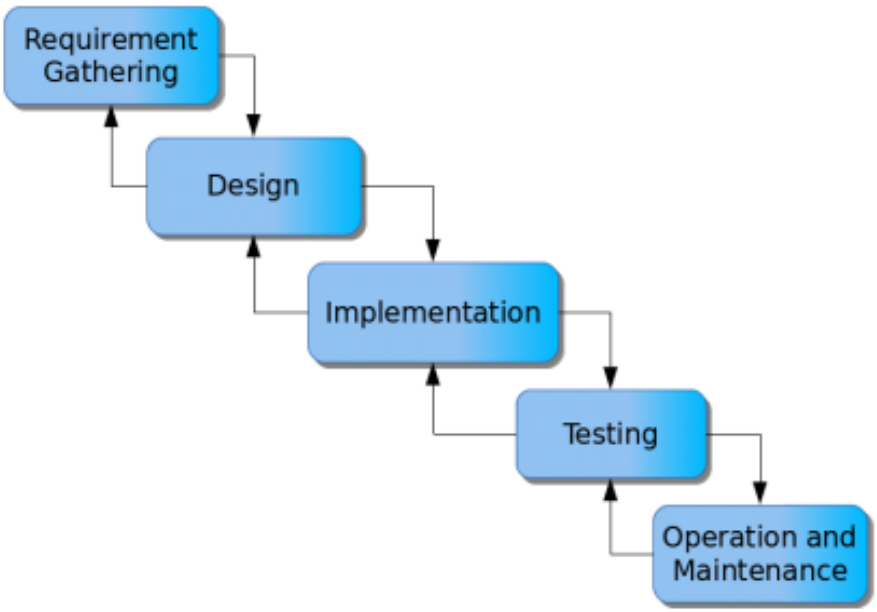

Figure 3.1 Waterfall model

3.1 Technologies:

There are many programming languages that supports socket programming such as Java, Python (which are really powerful) but above all, we believe **C++** is an optimal choice for socket applications programming due to

- **Low-level Networking and Memory Management:** C++ is renowned for its native support for intricate networking operations and its ability to efficiently manage memory. This is particularly beneficial for developers who need to implement precise control over network communication and memory usage.
- **Socket API and Performance:** With its comprehensive socket APIs, C++ is geared towards developers aiming to create high-performance, scalable networked applications. The language's emphasis on performance optimization is a cornerstone for

socket-based system development.

- **Portability:** The portability of C++ code is a significant advantage, especially in network programming. It allows applications to run seamlessly across diverse operating systems, which is crucial for the interoperability of networked systems.
- **Socket Programming:** Utilizing the standard C++ library's networking APIs enables developers to construct robust applications capable of establishing connections and exchanging data between machines through sockets, which are essential for network communication.

Our project take good advantage **GitHub** to simplify teamwork. Its collaborative platform is conducive to group work, making it easier for team members to contribute to the codebase, track each other's changes, and merge contributions seamlessly.

- **Version and Project Management with GitHub:** For project versioning and management, GitHub is our platform of choice. Its intuitive interface, coupled with excellent collaboration tools, makes it an ideal ecosystem for developers working together on complex projects.
- **Collaborative Development:** GitHub enhances the collaborative development process by offering features such as pull requests and issue tracking. This streamlines the workflow and helps maintain high standards of code quality.
- **Community and Open-Source Contributions:** The social aspect of coding on GitHub encourages community engagement and has significantly contributed to the growth of open-source development. It stands as a central hub for developers globally, promoting shared learning and collaboration.
- **Integration with Git:** GitHub's seamless integration with Git provides a robust framework for tracking changes and managing code across the development lifecycle, making it a powerful tool for developers.

CHAPTER 4: APPLICATION DESIGN

After thorough research and careful consideration of the pros and cons of various frameworks, we have determined that the **Client-Server Architecture** is the most suitable structure for our application.

4.1 Architecture

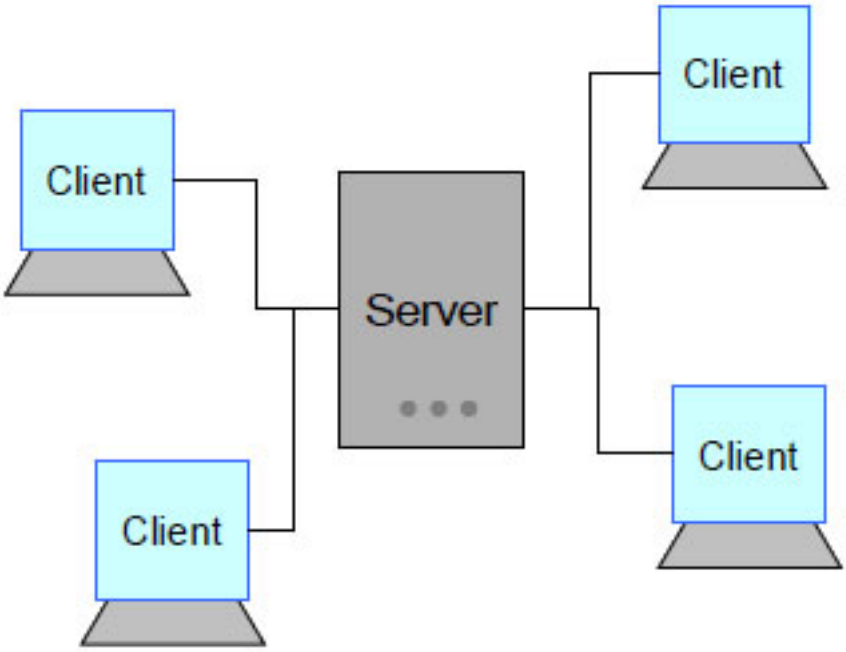

Figure 4.1 Server- Clients

Server plays a significant role in managing all incoming connection from clients by creating each thread for each connection in order to handle all of them at once.

Whenever a **Client** seeks to utilize the server's functionalities, it initiates a connection request and subsequently leverages the server's functionalities.

4.2 Protocol Design:

Every time a **Client** needs to interact with the server, it initiates a connection request, establishing a session that enables access to the server's services. This interaction is facilitated by **TCP**, which brings a host of advantages, including:

- Guaranteed Delivery: TCP ensures that data packets are reliably delivered to the

correct destination without loss.

- Data Integrity: It provides checksums for each data packet, safeguarding against corruption during transmission.
- Sequencing: Packets are reassembled in the correct order at the destination, even if they arrive out of sequence.
- Flow Control: By managing the data packet flow, TCP prevents the sender from overwhelming the receiver.

  With our enhancement of TCP implementation, our system excels in managing multiple client connections concurrently, ensuring stability and reliability.

4.3 Message format:

4.3.1 Client message

1. Admin

COMMAND	MESSAGE	DESCRIPTION
login	login\admin\1	Admin log in to system
log out	logout	Admin log out of system
add flight	add_flight\10 parameters	Insert new flight
delete flight	del_flight\flightnum	delete a flight
modify flight	modify(123)\flightnum(\newde_date)(\newre_date)	modify a flight's schedule

Table 4.1 Message format of Admin Client

2. User

COMMAND	MESSAGE	DESCRIPTION
log in	login\username\password	User log in to system
log out	logout	User log out to system
register	register\username\password	New user creates account
search	search(12345)\(comp)\dep\des (\dedate)(\redate)	search for flight
book	book\flightnum\seat class	process booking
view	view	view all booked ticket
cancel	cancel\flightnum	cancel ticket
change	change\tick\flightnum\seatclass	change ticket

Table 4.2 Message format of User Client

4.3.2 Server message:

1. To Admin

Table 4.3 Admin Functions and Responses

Function	Response
login	N_login, Y_admin
modify1	N_modify, Y_modify
modify2	N_modify, Y_modify
Del_flight	N_del, Y_del
Add_flight	N_add, Y_add
modify3	N_modify, Y_modify

2. To Users

Table 4.4 User Functions and Responses

Function	Response
login	Y_login, N_login
register	Y_register, N_register
search1	Y_found\result_str, N_found
search2	Y_found\result_str, N_found
search3	Y_found\result_str, N_found
search4	Y_found\result_str, N_found
search5	Y_found\result_str, N_found
book	N_book_avail, N_book, N_flight_not_found, success_book
cancel	cancel_success, N_cancel_err, N_cancel_notfound
change	N_book_avail, N_book, N_flight_not_found, success_book
view	Y_view\result_str, N_view

4.4 Sequence Diagram

4.4.1 Server's add flight

The sequence diagram demonstrate the insertion of a new flight into our database, with the message type is add_flight and parameters represent the flight's information.

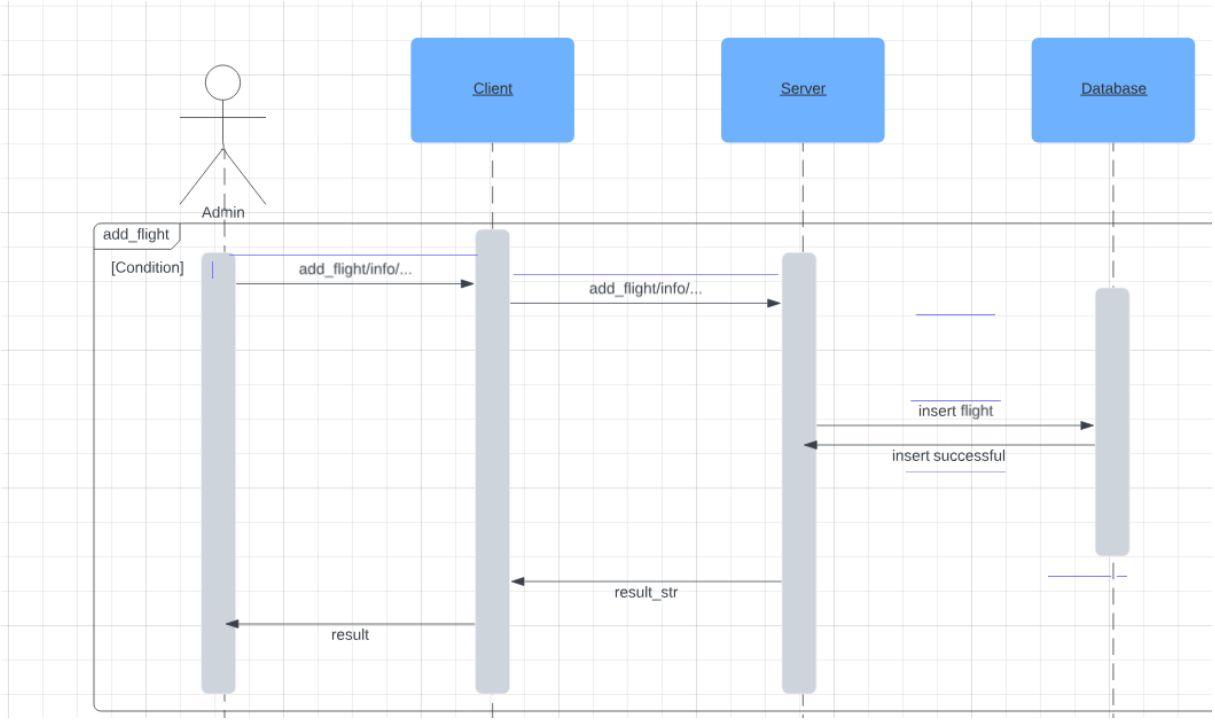

Figure 4.2 Add Flight

4.4.2 Client's book

This sequence diagram show us how a booking is process, with message type is book and parameters represent the information of the flight user want.

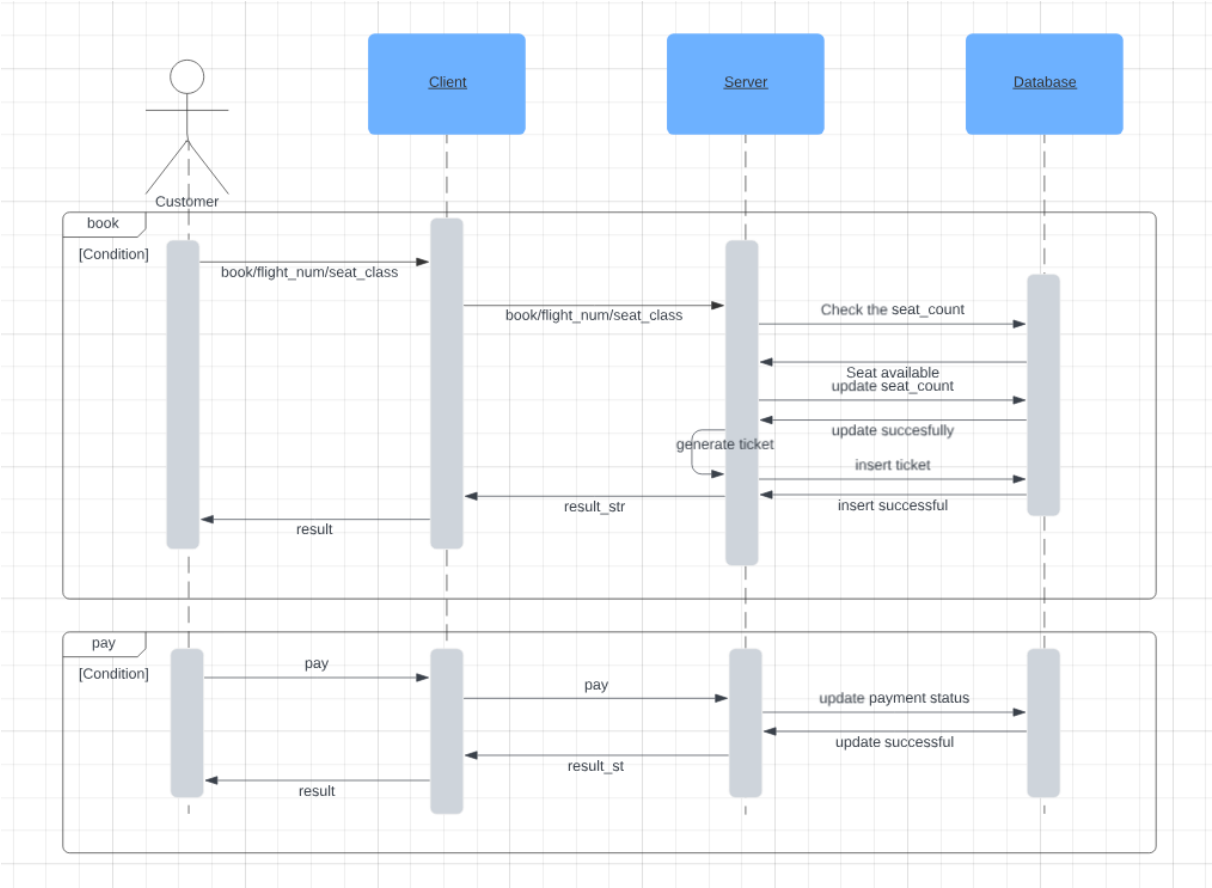

Figure 4.3 Book

4.4.3 Client's Cancel

The sequence diagram illustrates the process when a client requests to cancel a booking. The message type in this interaction is 'cancel' with parameters detailing the booking to be cancelled.

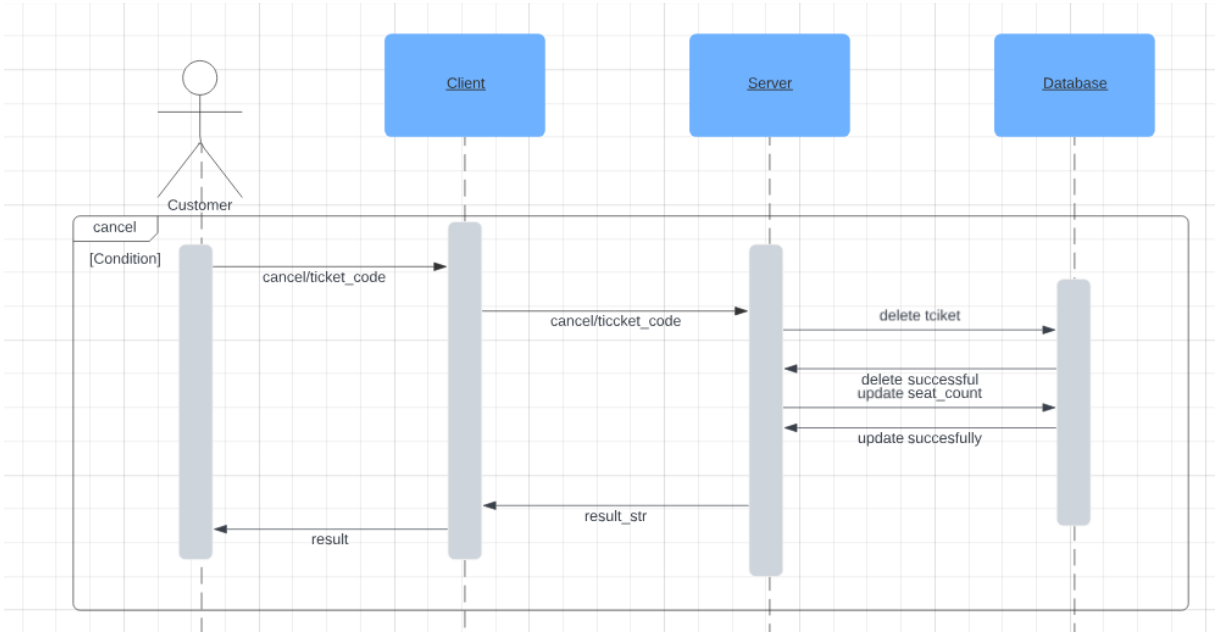

Figure 4.4 Cancel

4.4.4 Server's delete flight

Here, the sequence diagram depicts the deletion of a flight from the database. The message type for this operation is 'delete_flight', with parameters specifying the flight to be removed.

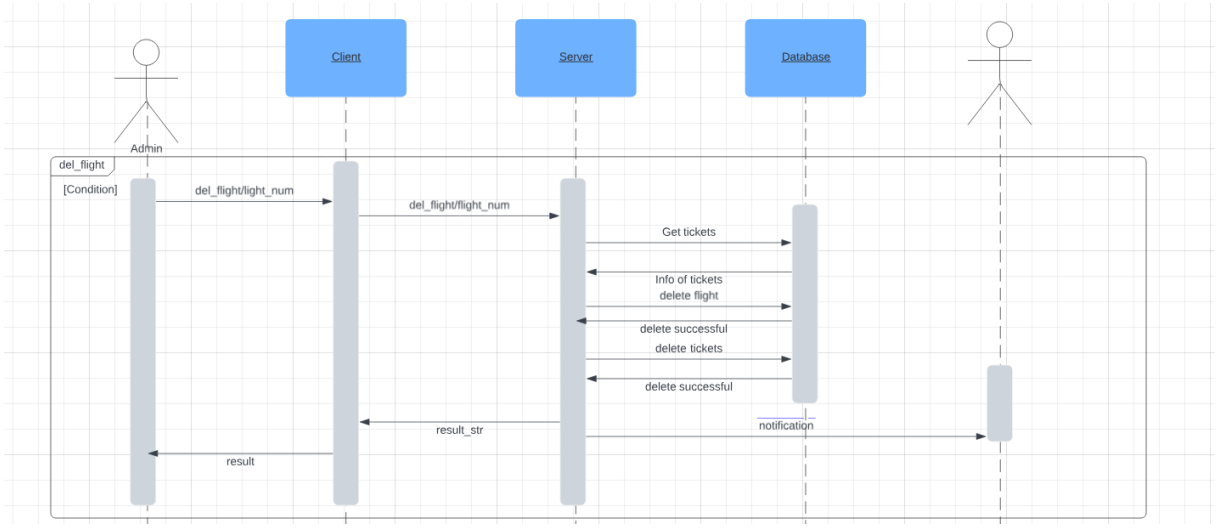

Figure 4.5 Delete Flight

4.4.5 Client's login

This sequence diagram shows the login process from the client's perspective. The message type is 'login', with parameters containing the user's credentials.

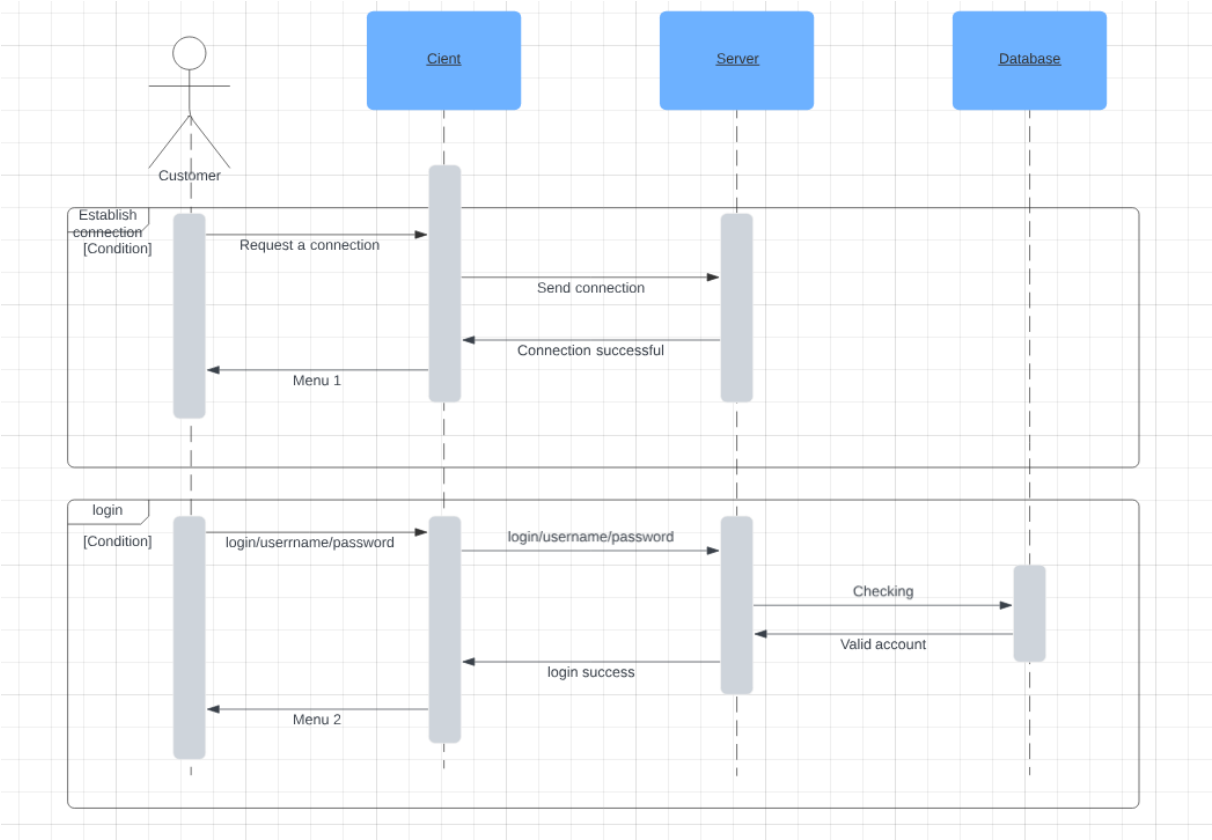

Figure 4.6 Login

4.4.6 Server's modify1

This sequence diagram details the first modification process on the server side. The message type for this operation is 'modify1', with parameters detailing the changes.

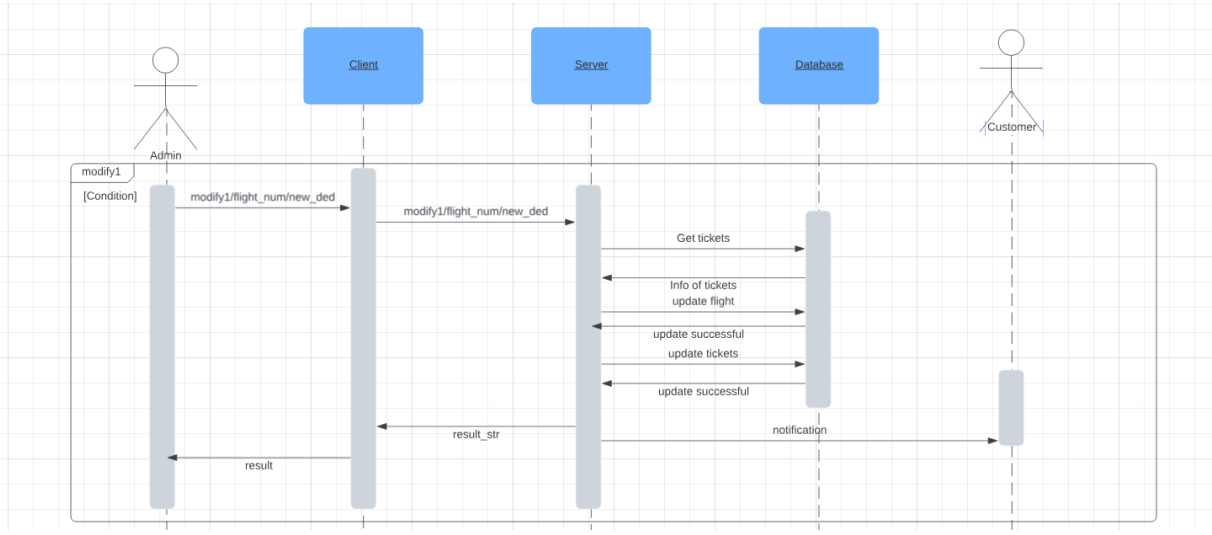

Figure 4.7 Modify 1

4.4.7 *Server's modify2*

The sequence diagram for the second modification by the server is shown here. The message type is 'modify2', with parameters that include the modification details.

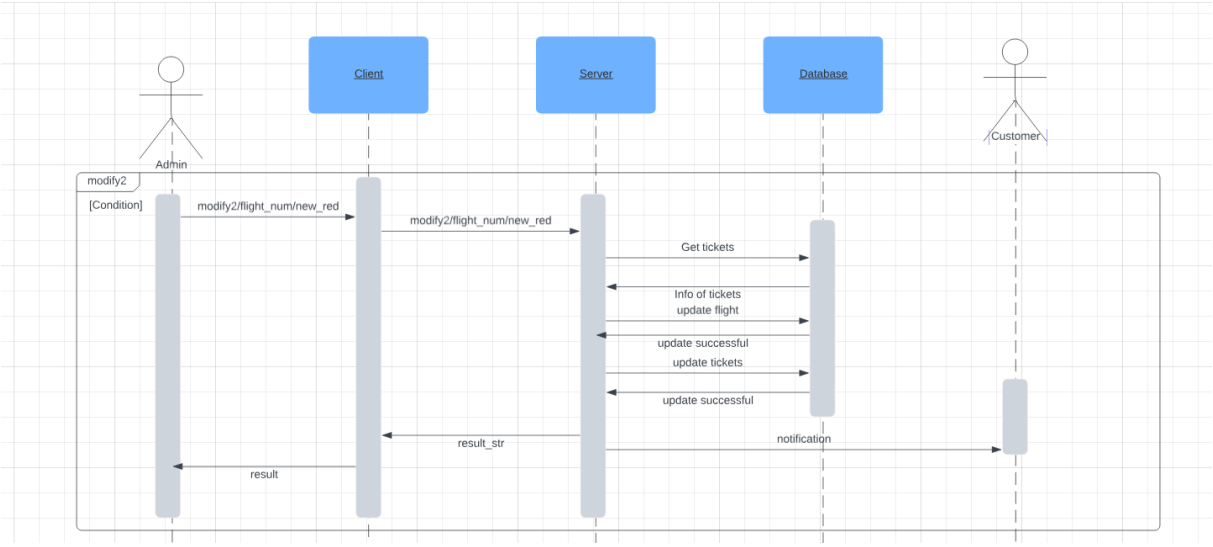

Figure 4.8 Modify 2

4.4.8 *Server's modify3*

Here the sequence diagram represents the third modification process on the server. The message type is 'modify3', with parameters that specify the modification data.

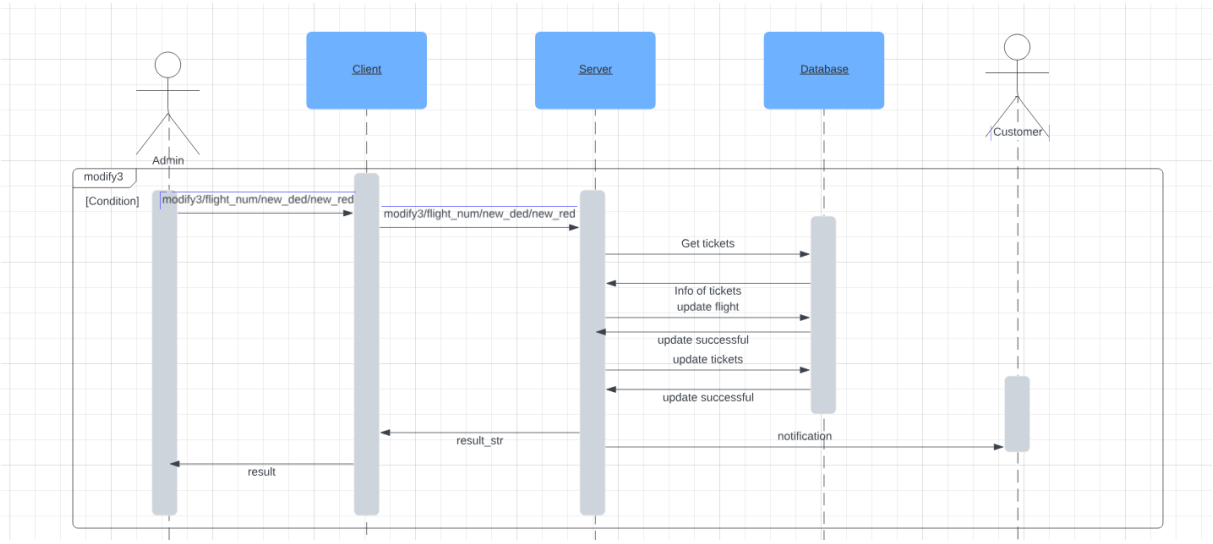

Figure 4.9 Modify 3

4.4.9 *Client's print all*

This sequence diagram shows the process by which a client can print all tickets. The message type is 'print_all', with parameters likely including the user ID or booking

details.

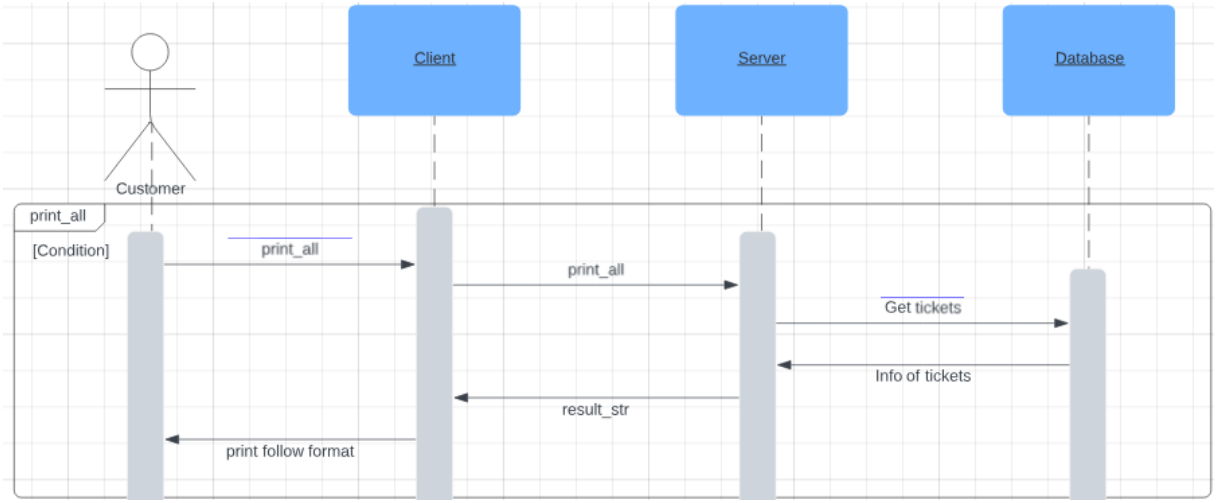

Figure 4.10 Print All

4.4.10 Client's print single ticket

The sequence diagram illustrates how a client prints a single ticket. The message type for this is 'print', and the parameters will identify the specific ticket.

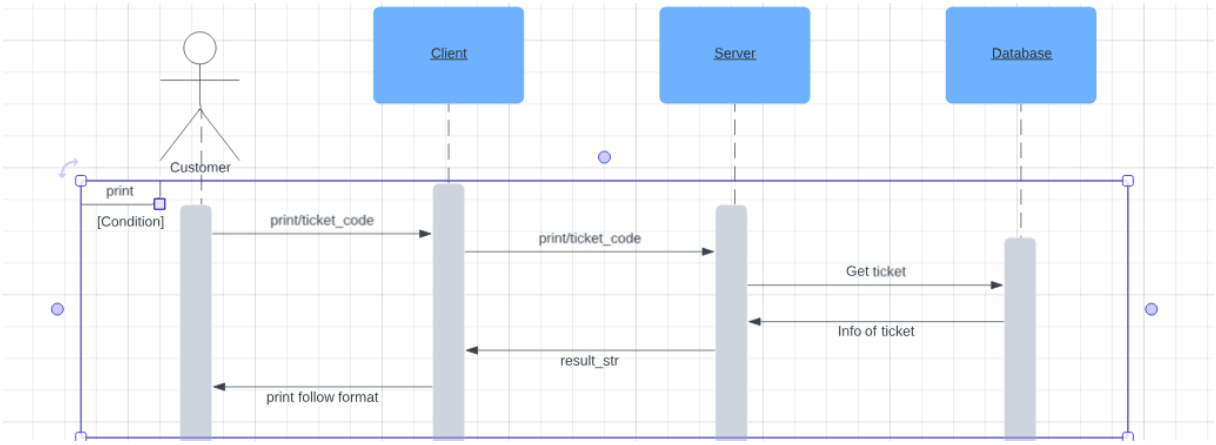

Figure 4.11 Print

4.4.11 Client's search1

This diagram shows the first type of search functionality available to the client. The message type is 'search1', with parameters defining the search criteria.

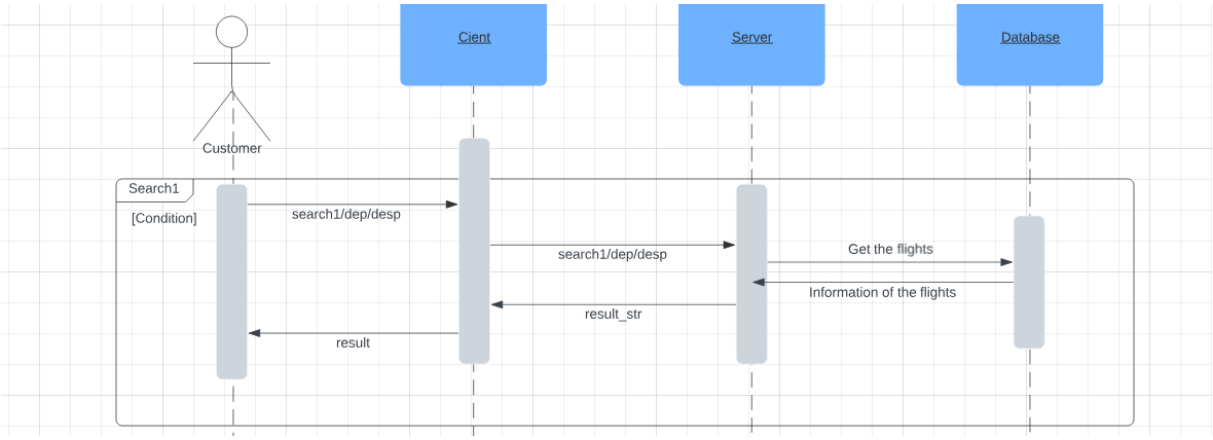

Figure 4.12 Search 1

4.4.12 Client's search2

The sequence diagram for the second search function is depicted here. The message type is 'search2', with parameters that detail the search query.

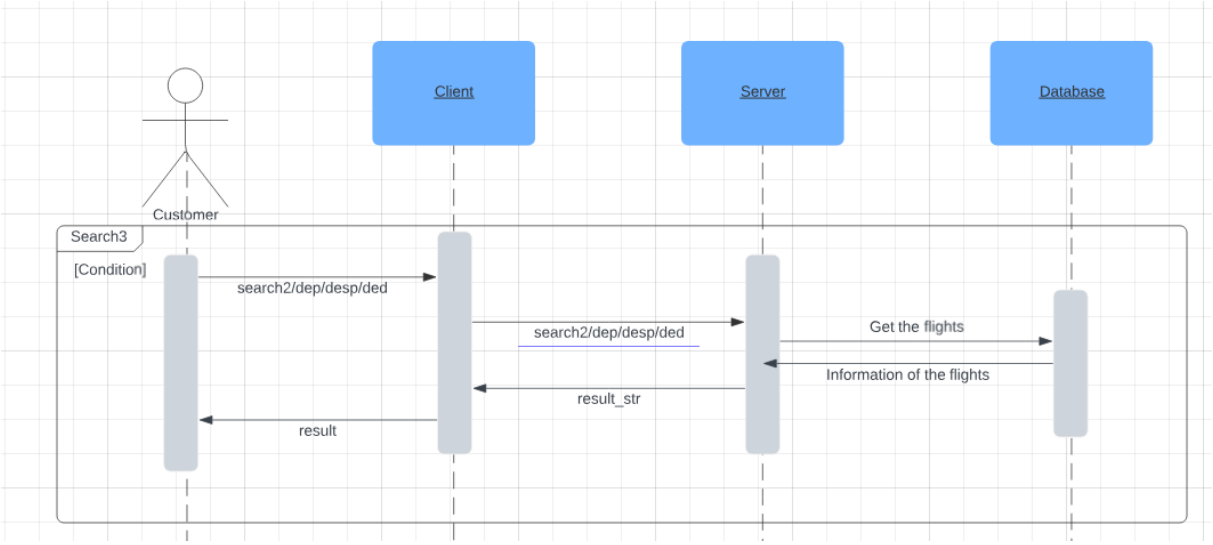

Figure 4.13 Search 2

4.4.13 Client's search3

The sequence diagram for the second search function is depicted here. The message type is 'search3', with parameters that detail the search query.

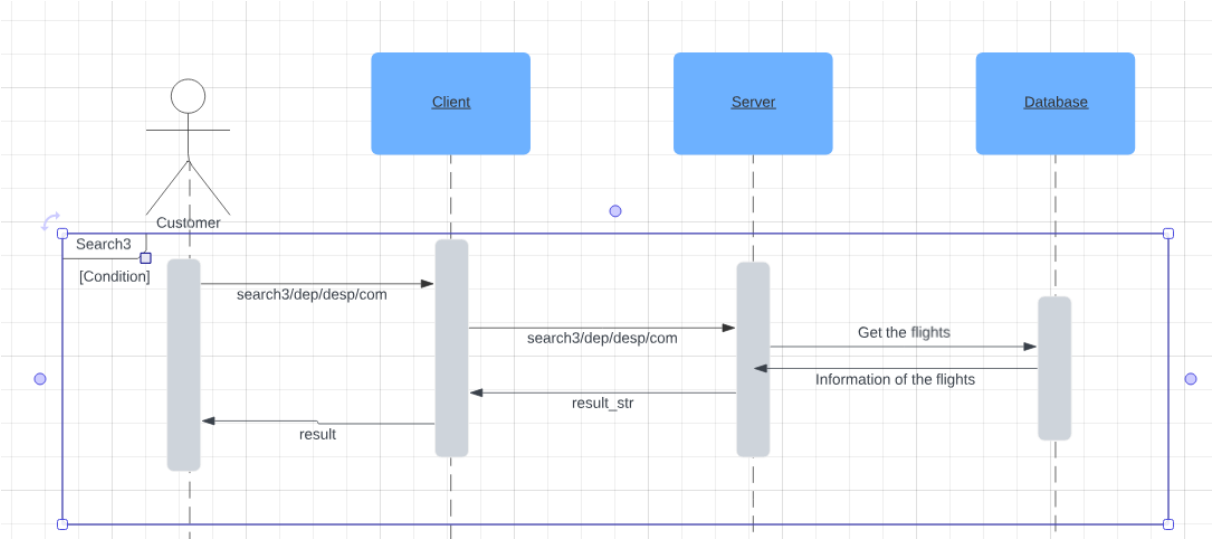

Figure 4.14 Search 3

4.4.14 Client's search4

The sequence diagram for the second search function is depicted here. The message type is 'search4', with parameters that detail the search query.

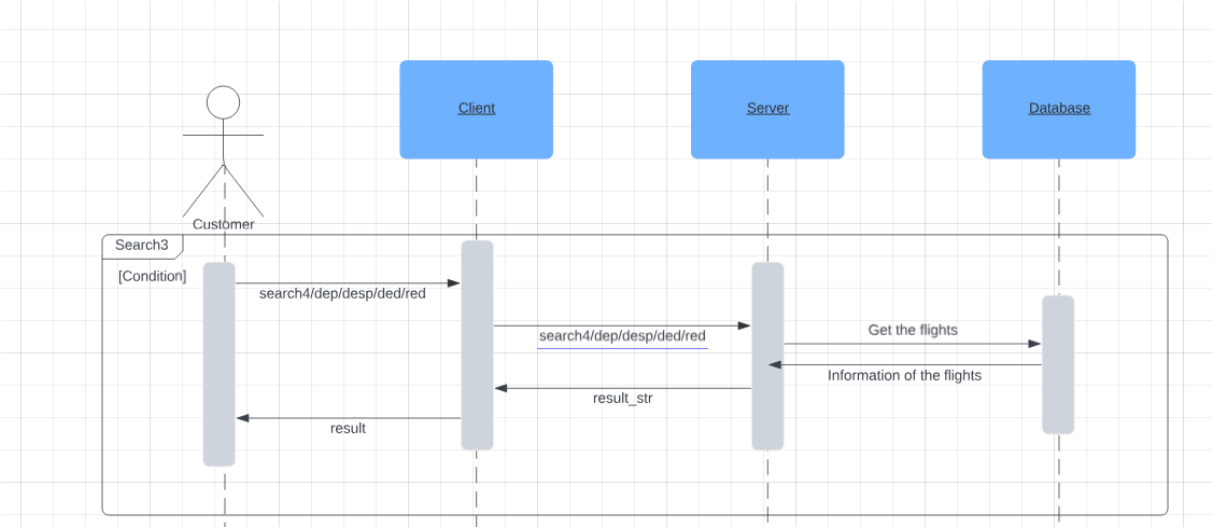

Figure 4.15 Search 4

4.4.15 Client's search5

The sequence diagram for the second search function is depicted here. The message type is 'search5', with parameters that detail the search query.

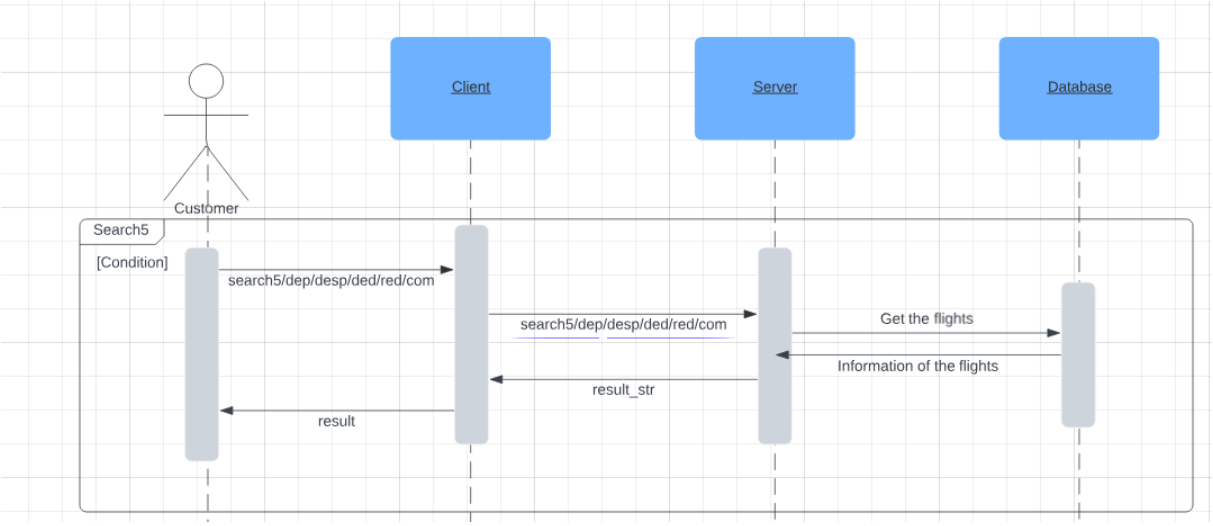

Figure 4.16 Search 5

4.4.16 Client's Sign Up

The sequence diagram for the second search function is depicted here. The message type is 'signup', with parameters that detail the user information

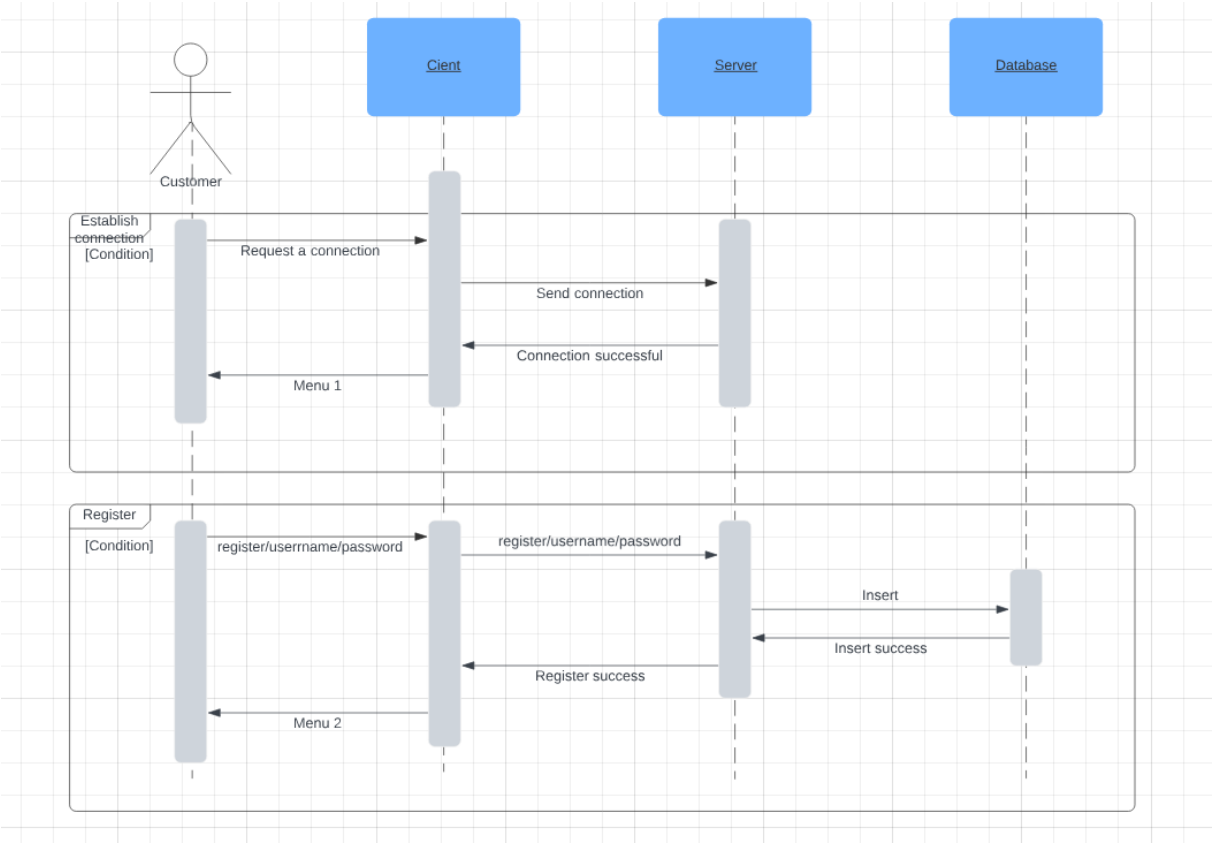

Figure 4.17 Sign Up

4.4.17 Client's View

The sequence diagram for the second search function is depicted here. The message type is 'view'

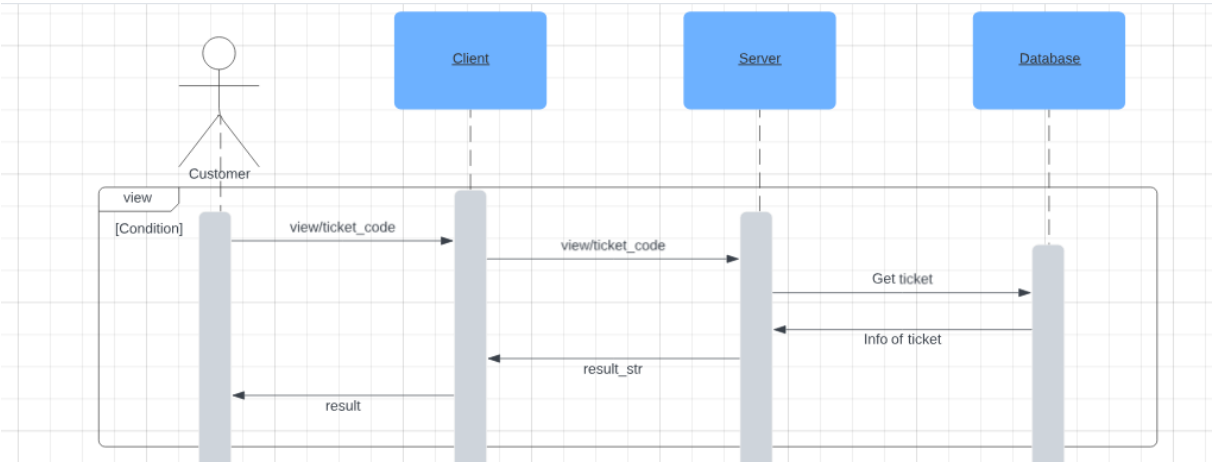

Figure 4.18 View

4.5 State Machine

4.5.1 Admin

Below is the state machine of Admin, which demonstrate clearly the admin's progress in the system. As an Admin, client has to log in first, then it can have access to the authority of administrator.

After that it can modify the time of departure date or return date of a flight, also delete an available flight or simply inserting a new flight.

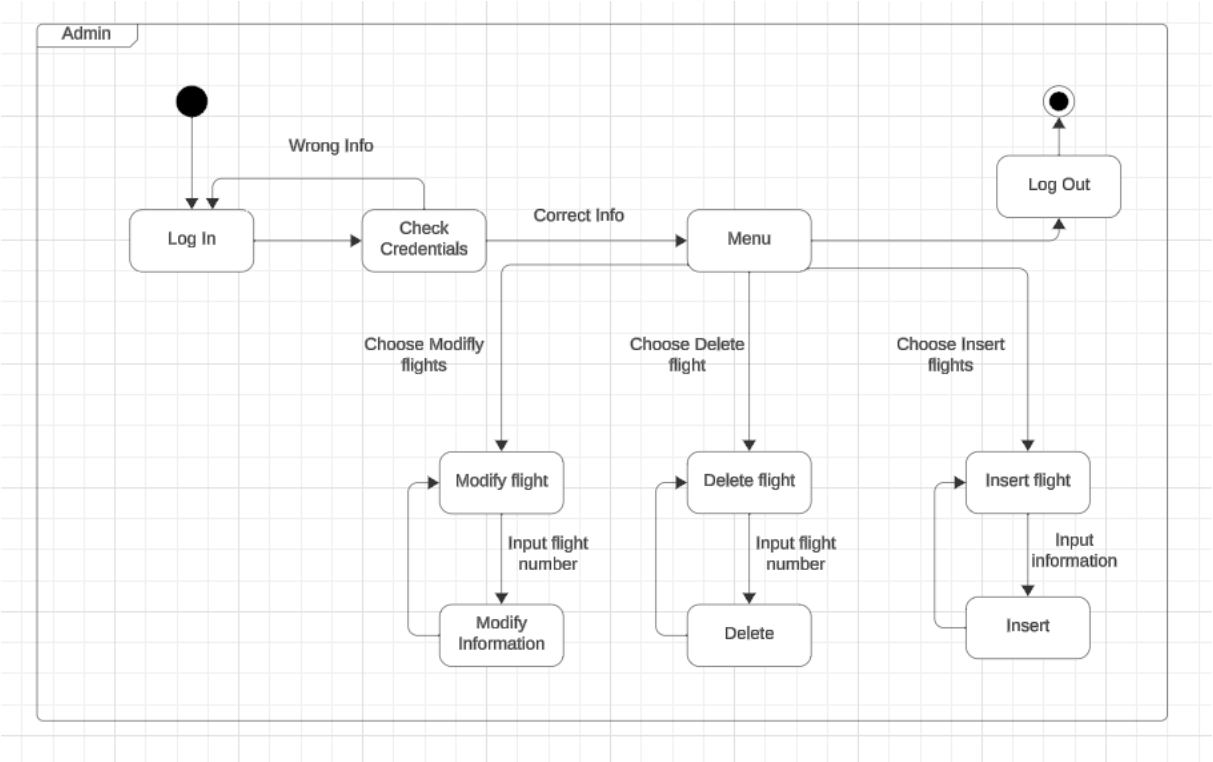

Figure 4.19 Admin_state

4.5.2 Users

This is the state machine of User, which demonstrate clearly the user's progress in the system. After logged in or registered successfully, the user can have access to the functionalities which server offers to user.

Users can search for flights based on various criterias, book a flight (need flight number and seat class), view booked ticket, cancel a ticket, change a ticket or make payment (with the same parameter as cancel: ticket_code) for those unpaid ticket of theirs.

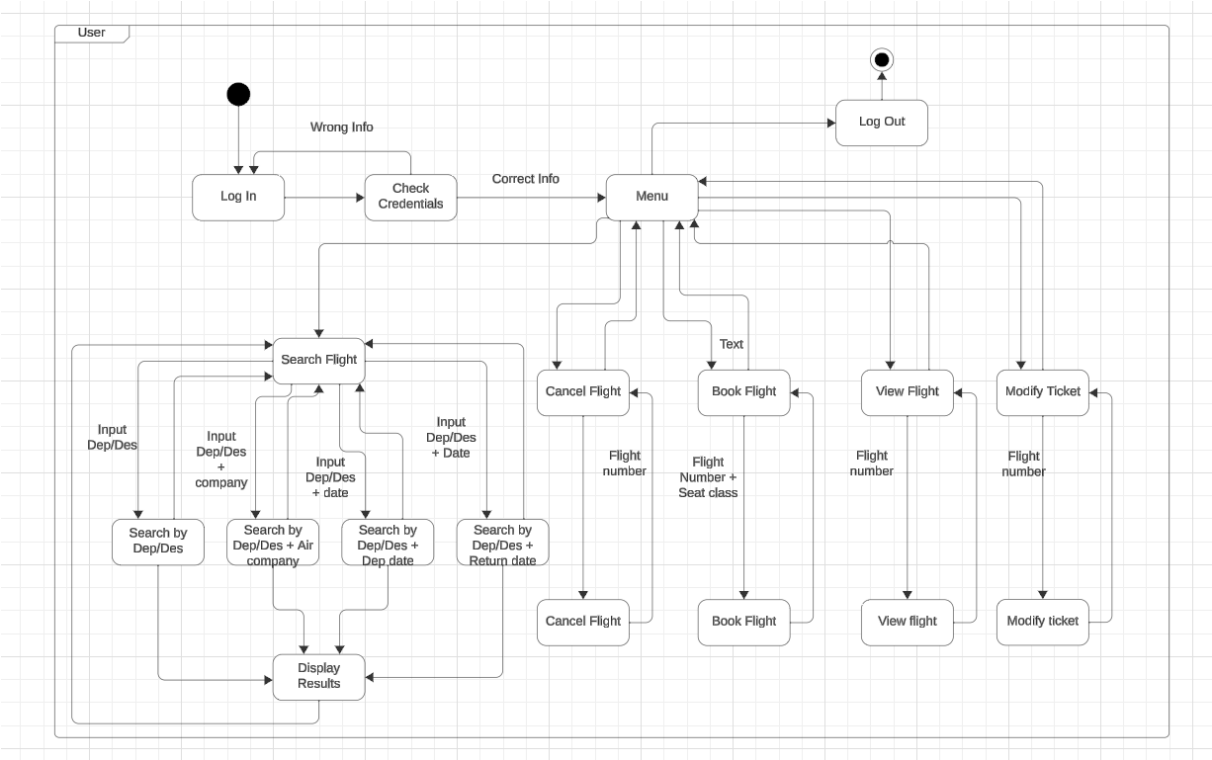

Figure 4.20 User_state

4.6 Database design

Our database is designed with 3 tables: Flights, Users, Tickets

Absolutely, here's a more detailed rewrite:

- Flights: The Flights table serves as a comprehensive repository of flight-related information, cataloging the complete list of flights that the airline offers. Each entry is uniquely defined by its flight_num, which acts as the primary key, ensuring quick access to the flight's details, such as the airline company, class availability, pricing, and scheduled dates and points of departure and arrival.
- Users: The Users table is dedicated to storing the personal and authentication details of the platform's users. The 'user_id' is the cornerstone of this table, providing a unique identifier for each user, which is crucial for maintaining user privacy and security while enabling personalized services and experiences.
- Tickets: Acting as a bridge between the Flights and Users tables, the Tickets table records the transactions of users booking flights. It details which user (referenced by 'user_id') has booked which flight (referenced by 'flight_num'), along with the class of the seat, the price paid, and the payment status. This table is pivotal for tracking bookings, ensuring proper allocation of seats, and managing financial transactions related to the flight services.

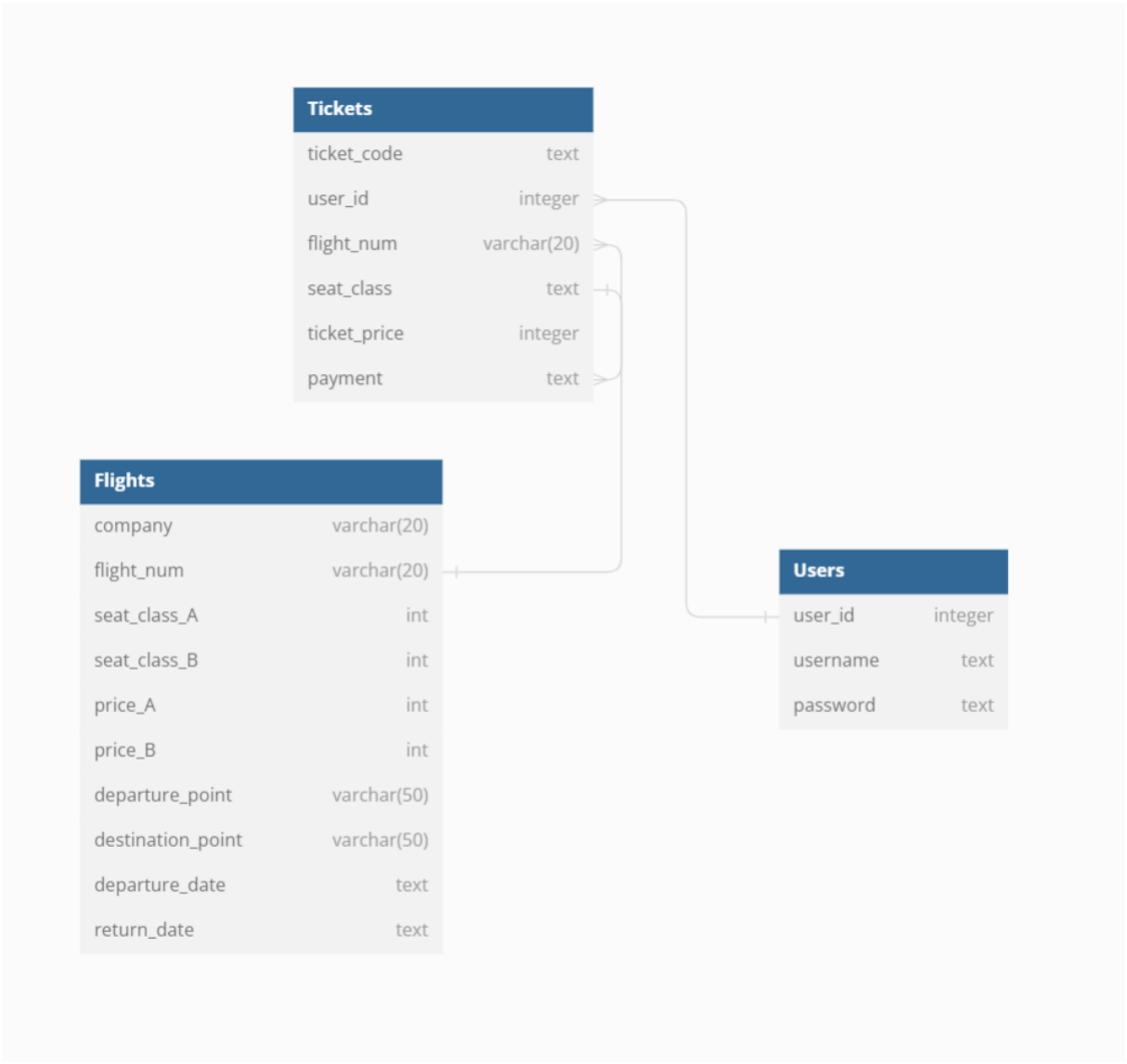

Figure 4.21 Database

CHAPTER 5:Result

We dedicated considerable effort to this project, culminating in the development of a robust client-server architecture. After a few weeks of designing, implementing the code, searching for availble technologies and finally testing, we establish a server-clients **Air tickets management systems** which meet the required demand:

- Establish a client-server connection using sockets
- Log received and sent messages on the server
- User account registration and login
- Search for flights based on various criteria
- Compare ticket prices and flight times from different airlines
- Book flight tickets and make online payments.
- Receive electronic ticket codes (via email or text message)
- Manage booked tickets: view ticket details, cancel tickets, change tickets, and print tickets
- Receive flight notifications: schedule changes, flight cancellations, delays

Our project demands a high level of concentration and comprehensive collaboration among all team members, as well as their relentless effort to continuously explore and learn.

Below are the specific contributions made by each team member.

Member	**Task**	**Contribution**
Mai Hoàng Đức	Server, Client, System functionalities, Report, Presentation	65%
Nguyễn Xuân Kiên	Client functionality, Report	35%

Table 5.1 Member contribution